Understanding TOCTTOU in the Windows Kernel Font Scaler Engine

wang yu

pjf bugvuln

Black Hat - USA, 2014

Part One

Introduction

Introduction

- About me (wangyu@360.cn)

- Background

The Font Scaler engine is widely used in Microsoft Windows and Mac operating systems for rendering TrueType fonts. To improve the performance of the Windows NT, Microsoft decided to move the engine from user mode to kernel mode. This enhancement does improve the performance, but it also brings security issues.

Many things make the font engine vulnerable. Such as the complexity of font file format, the assumptions about the interactions between the font engine and its clients (win32k.sys), and the existence of font cache.

Among these vulnerabilities, TOCTTOU (Time-of-Check to Time-of-Use) is the most critical type.

Introduction

- Outline

1. Design and implementation of Font Scaler Client Interface
2. Discuss how to implement Font Scaler engine and its client in User Mode – Project fs-engine
3. The famous CVE-2011-3402 and how Win32k.sys works as a client of the Font Scaler engine in kernel mode
4. Bochspwn and TOCTTOU problems in the Windows Kernel Font Scaler engine
5. More finding about TOCTTOU problems
6. Introduce the Architecture of our tool – digTool

Part Two

Smashing the Font Scaler Client Interface

Why It Is

- Why it's significant

1. Font engine is the must-have component of any OS with user interaction

2. It succeeded in solving the low-resolution problem in font rendering

- Why it's difficult

1. Many data structures lacking details

2. The TrueType font developers even strive to resolve low-resolution problems by introducing a set of instructions and TrueType Rasterizer Interpreter

Why It Is

- Why it's worth

1. The integration of font engine into windows kernel greatly facilitates the vulnerability exploitation

2. Font could be embedded into Office file and PDF file, and could be also embedded into web page, indicating vulnerabilities could be exploited remotely

- My approach and Disclaimer

1. Static Analysis and Reversing

2. Dynamic Tracing

3. White-Box Analysis !

The Profile of the Font Scaler Client Interface

The minimum set of the interface

Routine	Description
fs_OpenFonts	opens the font scaler
fs_Initialize	initializes the font scaler
fs_NewSfnt	specifies the SFNT data structure
fs_NewTransformation	specifies the point size, the transformation matrix, the pixel diameter, and the device resolution
fs_NewGlyph	computes the glyph index from the character code
fs_ContourGridFit / fs_ContourNoGridFit	converts the glyph description into an outline with or without executing the instructions (integrated into fs_NewContourGridFit)
fs_FindBitMapSize	determines the amount of memory to create a bitmap of the glyph
fs_ContourScan	converts the outline into a bitmap
fs_CloseFonts	closes the font scaler

Table 1. The minimum set of the FSCI

The discarded routines of the interface

Routine	Description
fs_GetAdvanceWidth	extracts information about the advance width of a glyph
fs_GetScaledAdvanceWidths	returns the hinted advance widths for the range of glyphs specified

Table 2. The discarded routines of the FSCI

The Profile of the Font Scaler Client Interface

Routine	Description
fs_SizeOfOutlines	calculates the amount of memory that the Font Scaler needs to cache the outline
fs_SaveOutlines	stores the outline data in the outline cache
fs_RestoreOutlines	recovers the outline data stored in the outline cache
fs_FindGraySize	calculate gray scale scan conversion memory requirements
fs_FindBandingSize	calculate memory requirements for banding
fs_FindGrayBandingSize	calculate gray scale memory requirements for banding
fs_ContourGrayScan	generate a gray scale bitmap

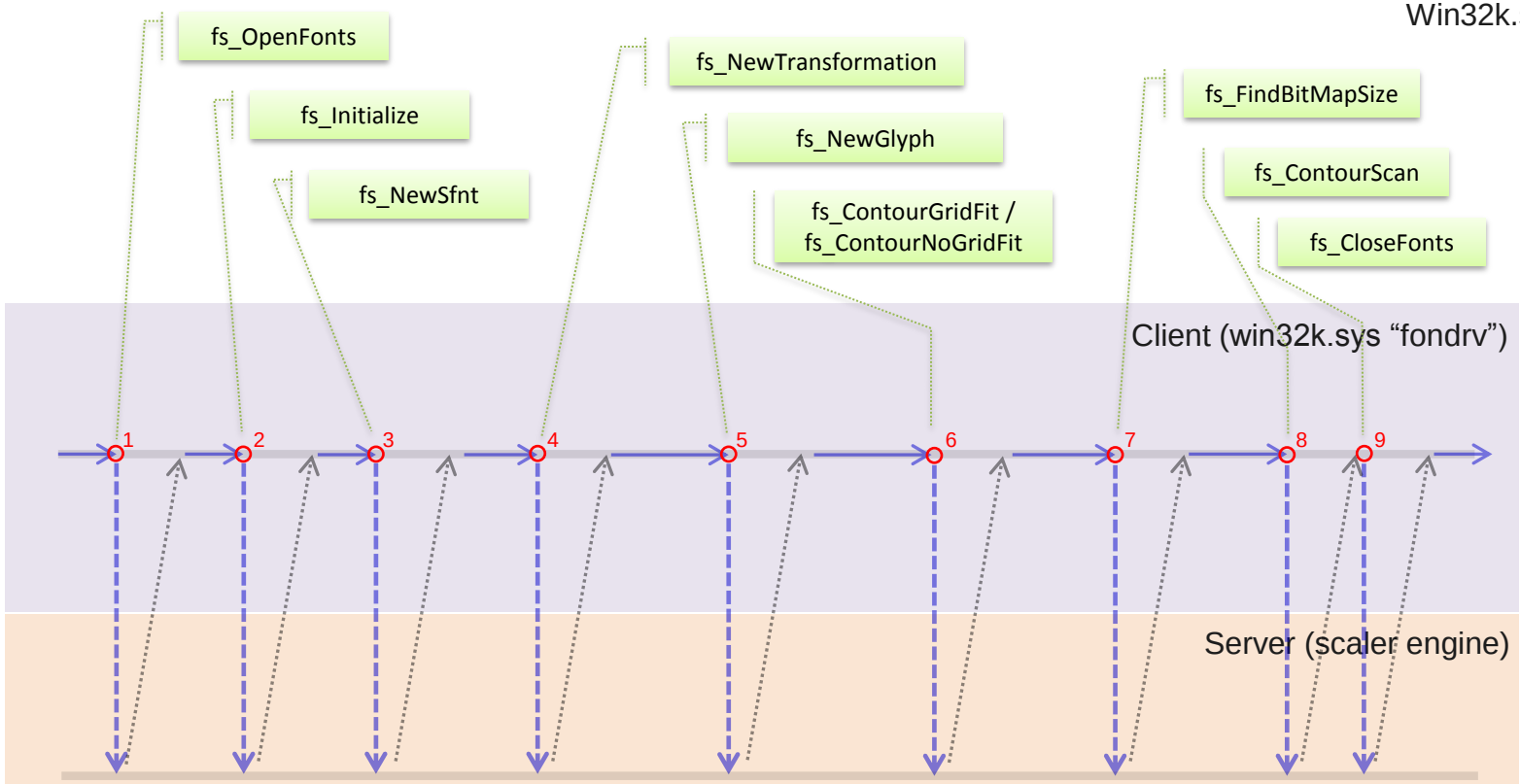
Table 2 (continued). The discarded routines of the FSCI

Routine Prefixes

fs_ / _fs_	engine export interface
fsc_	engine converter routine
fsg_	glyph related routine
sbit_	bitmap support routine
itrp_	TT rasterizer interpreter
sfac_	font format parser routine
mth_	math routine
fnterr_	error support routine

Table 3. Routine Prefixes

The Sequence of the Interface



Function Prototypes

```
----------------------------
fs_OpenFonts(fs_GlyphInputType *inputPtr, fs_GlyphInfoType *outputPtr);
fs_Initialize(fs_GlyphInputType *inputPtr, fs_GlyphInfoType *outputPtr);
fs_NewSfnt(fs_GlyphInputType *inputPtr, fs_GlyphInfoType *outputPtr);
fs_NewTransformation(fs_GlyphInputType *inputPtr, fs_GlyphInfoType *outputPtr);
.............
fs_CloseFonts(fs_GlyphInputType *inputPtr, fs_GlyphInfoType *outputPtr);
```

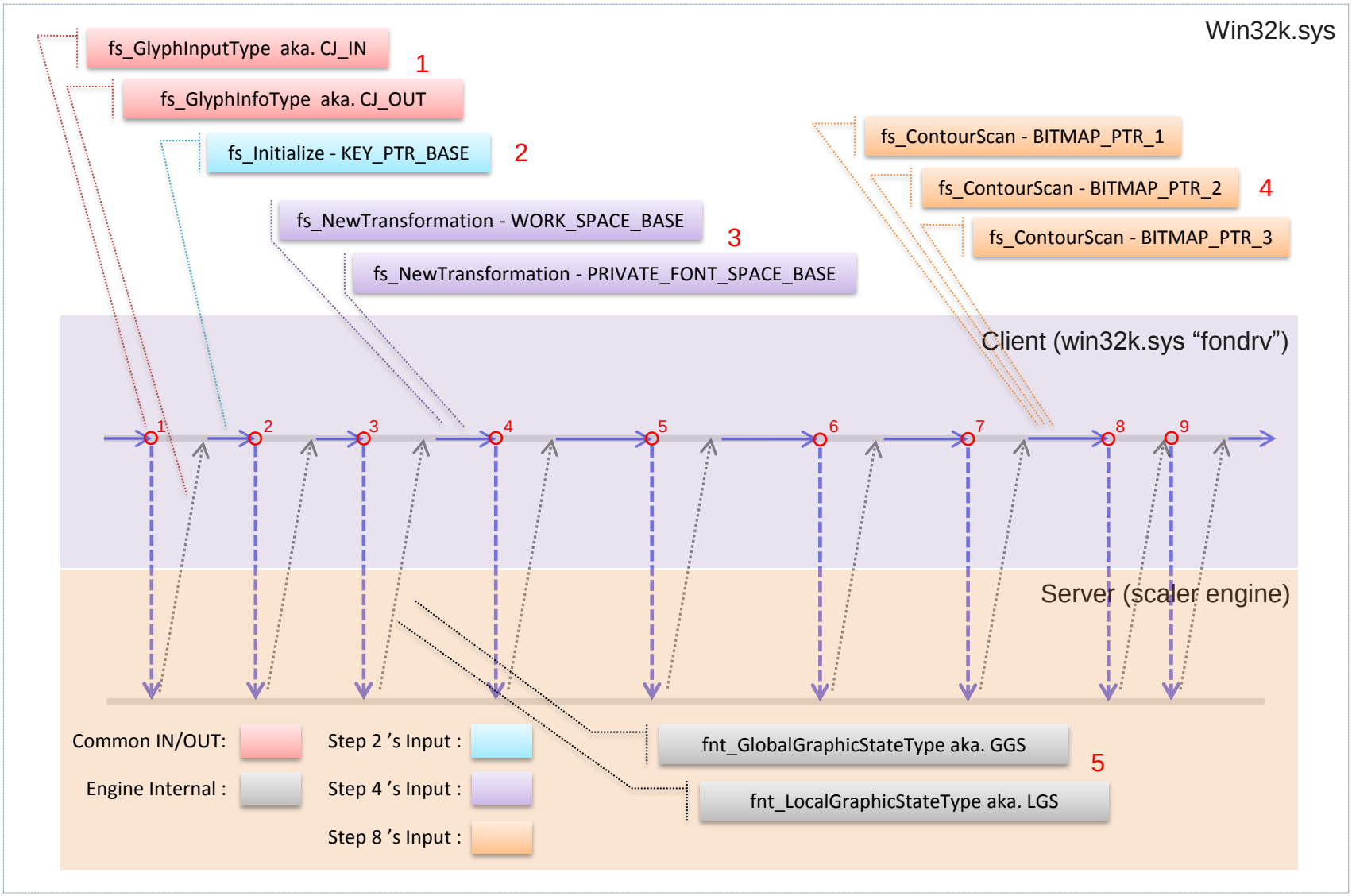
Base Data Structures of the Engine

Based on White-Box analysis,
Data structures is not difficult for me

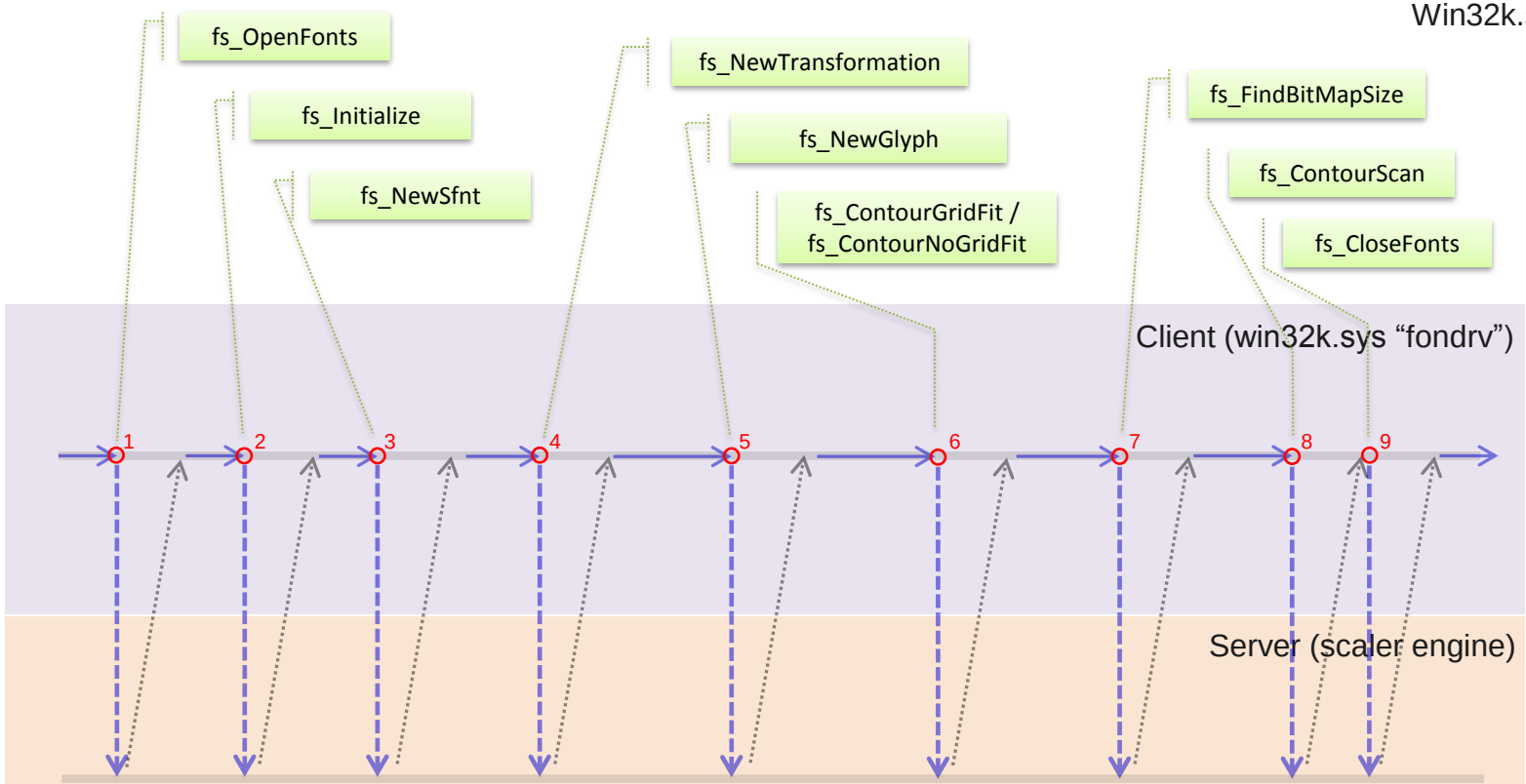
```
0:000:x86> dt fs_GlyphInputType -r
client!fs_GlyphInputType
   +0x000 version                : Int4B
   +0x004 memoryBases            : [9] Ptr32 Char
   +0x028 sfntDirectory          : Ptr32 Int4B
   +0x02c GetSfntFragmentPtr     : Ptr32 void*
   +0x030 ReleaseSfntFrag        : Ptr32 void
   +0x034 clientID               : Int4B
   +0x038 param                  : __unnamed
      +0x000 newsfnt             : __unnamed
         +0x000 platformID       : Uint2B
         +0x002 specificID       : Uint2B
      +0x000 newtrans            : __unnamed
         +0x000 pointSize        : Int4B
         +0x004 xResolution      : Int2B
         +0x006 yResolution      : Int2B
         +0x008 pixelDiameter    : Int4B
         +0x00c transformMatrix  : Ptr32 transMatrix
         +0x010 traceFunc        : Ptr32 Void
      +0x000 newglyph            : __unnamed
         +0x000 characterCode    : Uint2B
         +0x002 glyphIndex       : Uint2B
      +0x000 gridfit             : __unnamed
         +0x000 styleFunc        : Ptr32 void
         +0x004 traceFunc        : Ptr32 Void
      +0x000 scan                : __unnamed
         +0x000 bottomClip       : Int2B
         +0x002 topClip          : Int2B
         +0x004 outlineCache     : Ptr32 Int4B
```

```
0:000:x86> dt fs_GlyphInfoType -r
client!fs_GlyphInfoType
   +0x000 memorySizes                : [9] Int4B
   +0x024 glyphIndex                 : Uint2B
   +0x026 numberOfBytesTaken         : Uint2B
   +0x028 metricInfo                 : metricsType
      +0x000 advanceWidth            : vectorType
         +0x000 x                    : Int4B
         +0x004 y                    : Int4B
      +0x008 leftSideBearing         : vectorType
         +0x000 x                    : Int4B
         +0x004 y                    : Int4B
      +0x010 leftSideBearingLine     : vectorType
         +0x000 x                    : Int4B
         +0x004 y                    : Int4B
      +0x018 devLeftSideBearingLine : vectorType
         +0x000 x                    : Int4B
         +0x004 y                    : Int4B
      +0x020 devAdvanceWidth         : vectorType
         +0x000 x                    : Int4B
         +0x004 y                    : Int4B
      +0x028 devLeftSideBearing      : vectorType
         +0x000 x                    : Int4B
         +0x004 y                    : Int4B
   +0x058 bitMapInfo                 : BitMap
      +0x000 baseAddr                : Ptr32 Char
      +0x004 rowBytes                : Int2B
      +0x006 bounds                  : Rect
         +0x000 top                  : Int2B
         +0x002 left                 : Int2B
         +0x004 bottom               : Int2B
         +0x006 right                : Int2B
   +0x068 outlineCacheSize           : Int4B
   +0x06c outlinesExist              : Uint2B
   +0x06e numberOfContours           : Uint2B
   +0x070 xPtr                       : Ptr32 Int4B
   +0x074 yPtr                       : Ptr32 Int4B
   +0x078 startPtr                   : Ptr32 Int2B
   +0x07c endPtr                     : Ptr32 Int2B
    .....
```

Base Data Structures of the Engine



The State Machine of the Engine



State Transition

---

fs_Initialize : INITIALIZED

fs_NewSfnt : INITIALIZED | NEWSFNT

fs_NewTransformation : INITIALIZED | NEWSFNT | NEWTRANS

fs_NewGlyph : INITIALIZED | NEWSFNT | NEWTRANS | GOTINDEX

fs_ContourGridFit / fs_ContourNoGridFit : INITIALIZED | NEWSFNT | NEWTRANS | GOTINDEX | GOTGLYPH

fs_FindBitMapSize : INITIALIZED | NEWSFNT | NEWTRANS | GOTINDEX | GOTGLYPH | SIZEKNOWN

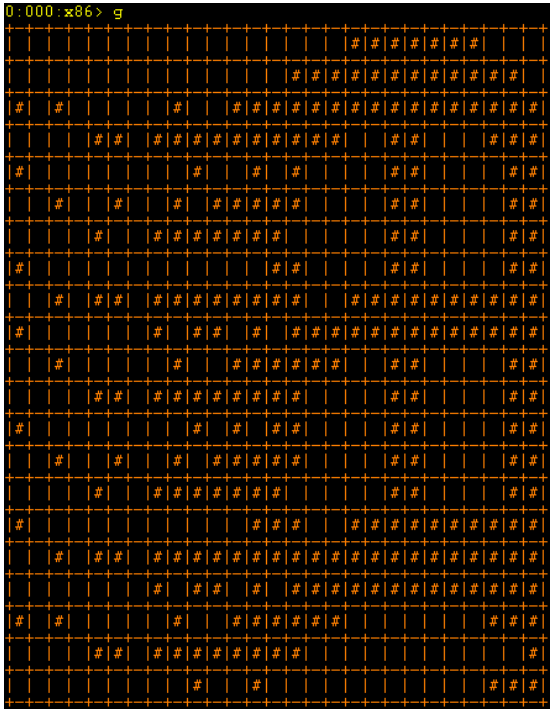
“Sounds Great~ But... So What”

Okay, Let's make it more interesting and see more "clearly"~

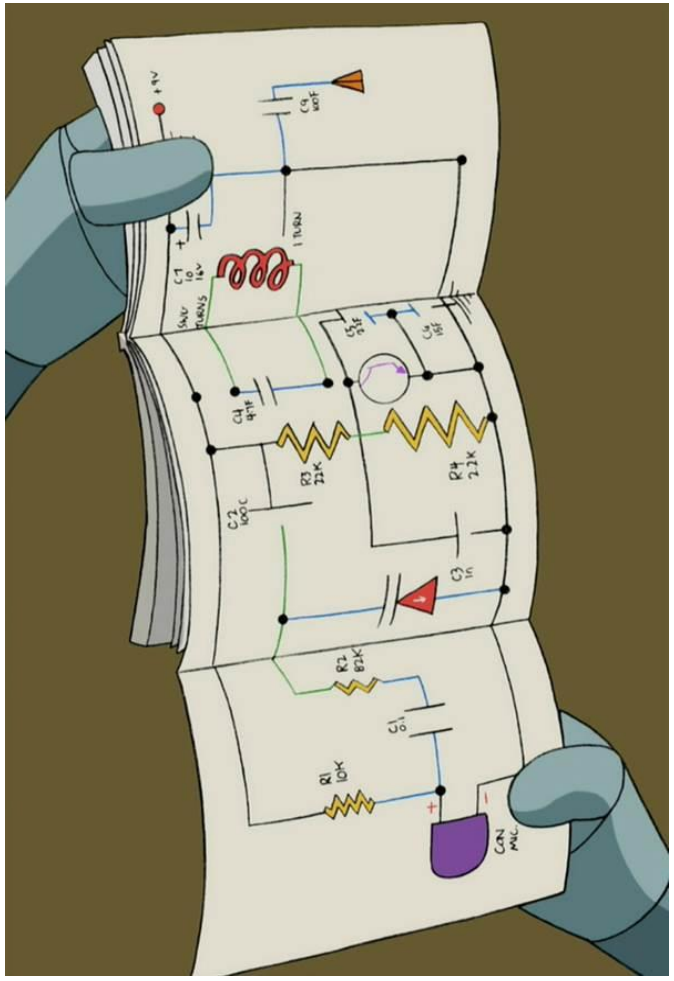
Project : fs-engine

https://github.com/keenjoy95/fs-engine

Windows Logo

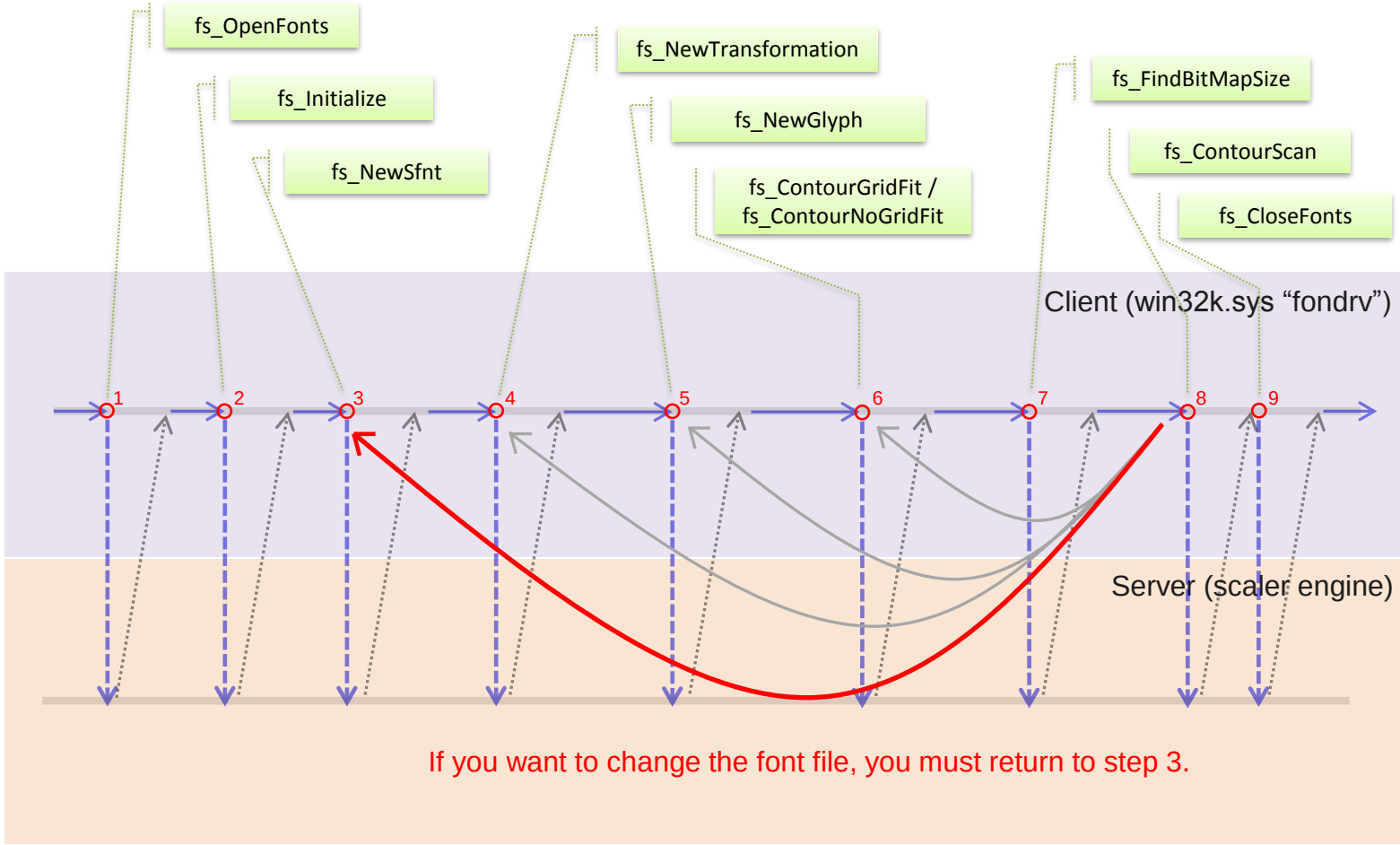


Wingdings TrueType Font

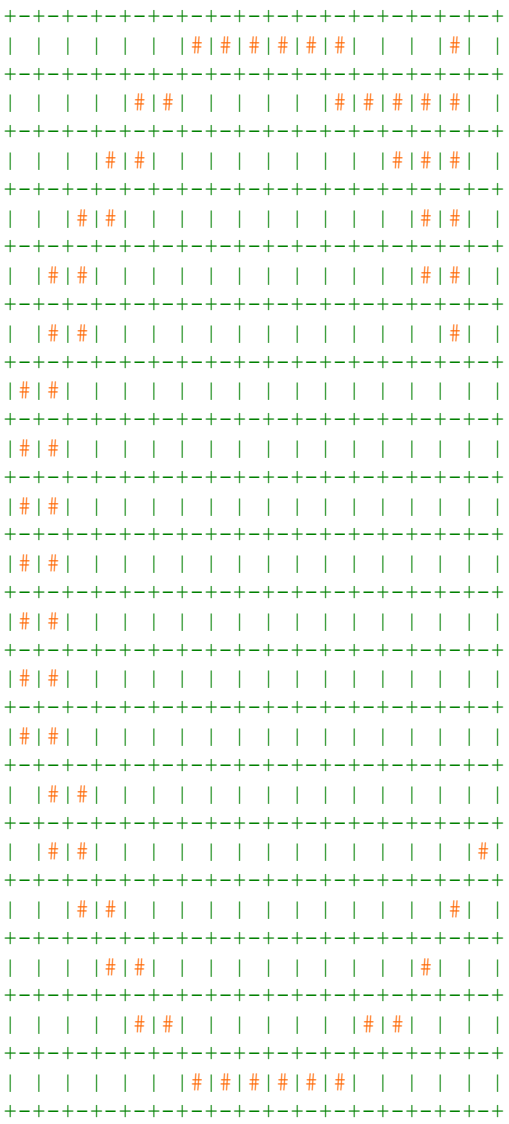


Bender Bending Rodríguez

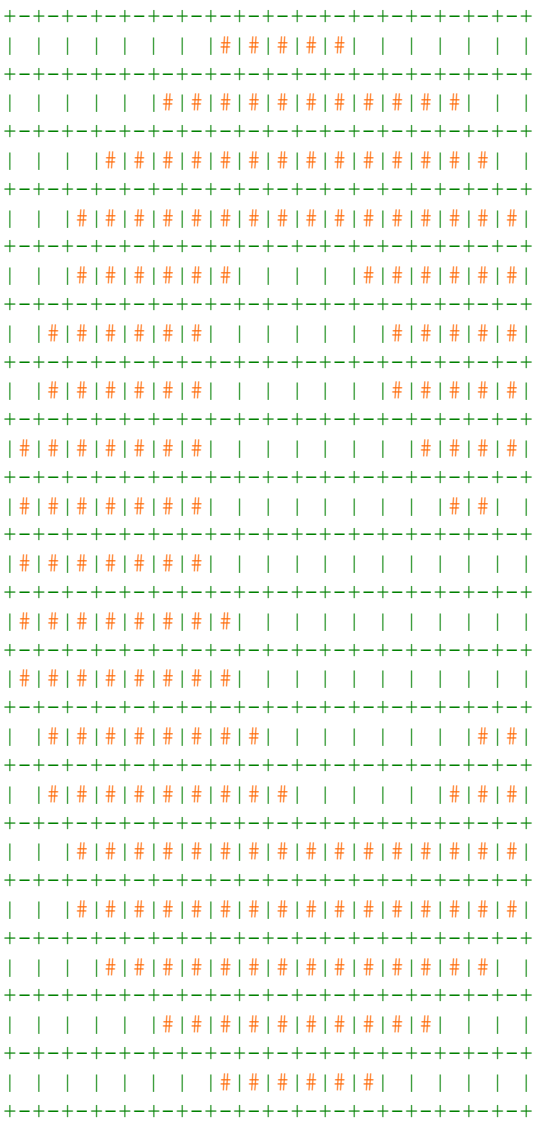
DEMO 1 : Change the Font File



DEMO 1 : Change the Font File

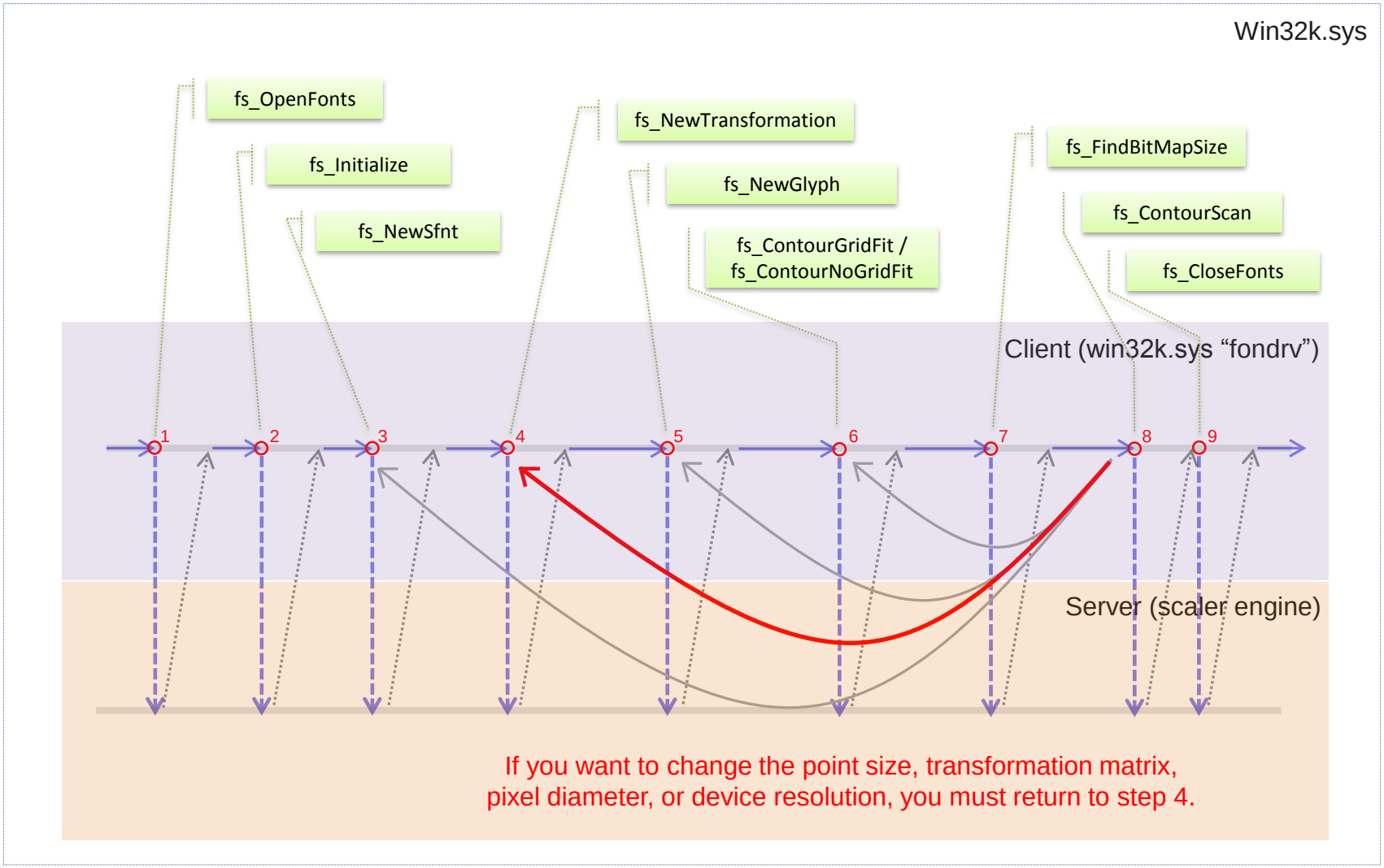


using “times.ttf” to display

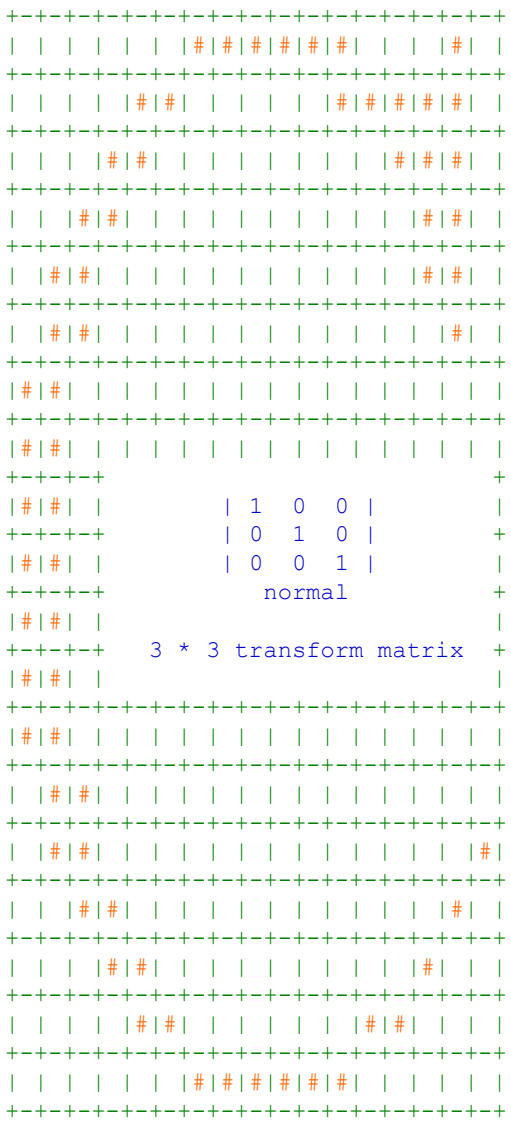


using “coopbl.ttf” to display

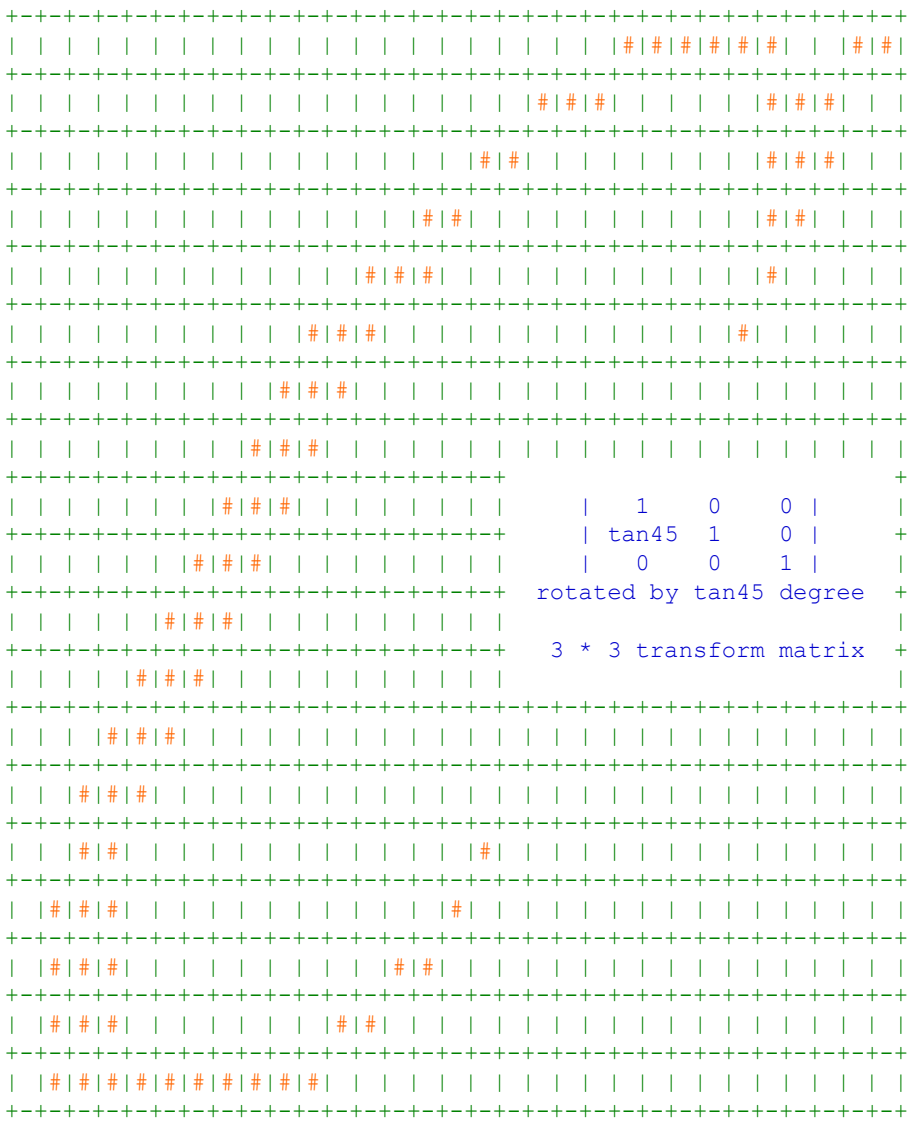
DEMO 2 : Change the Transformation Matrix



DEMO 2 : Change the Transformation Matrix

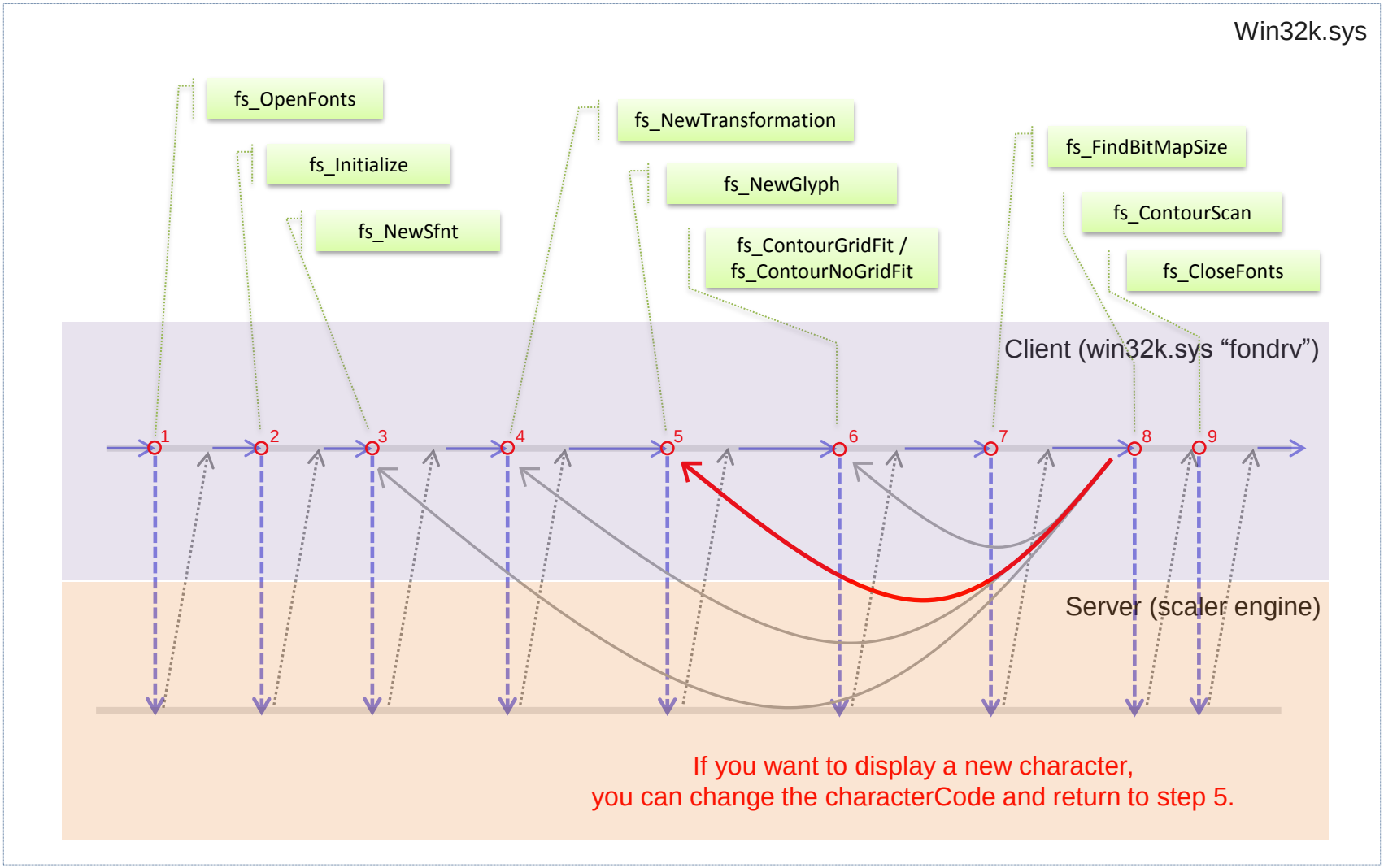


normal

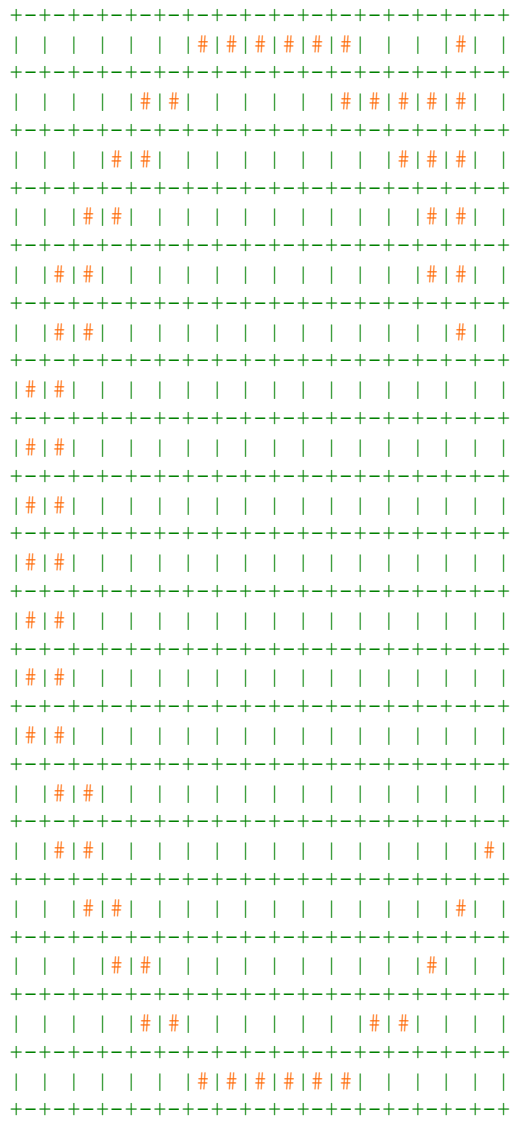


rotated by tan45 degree

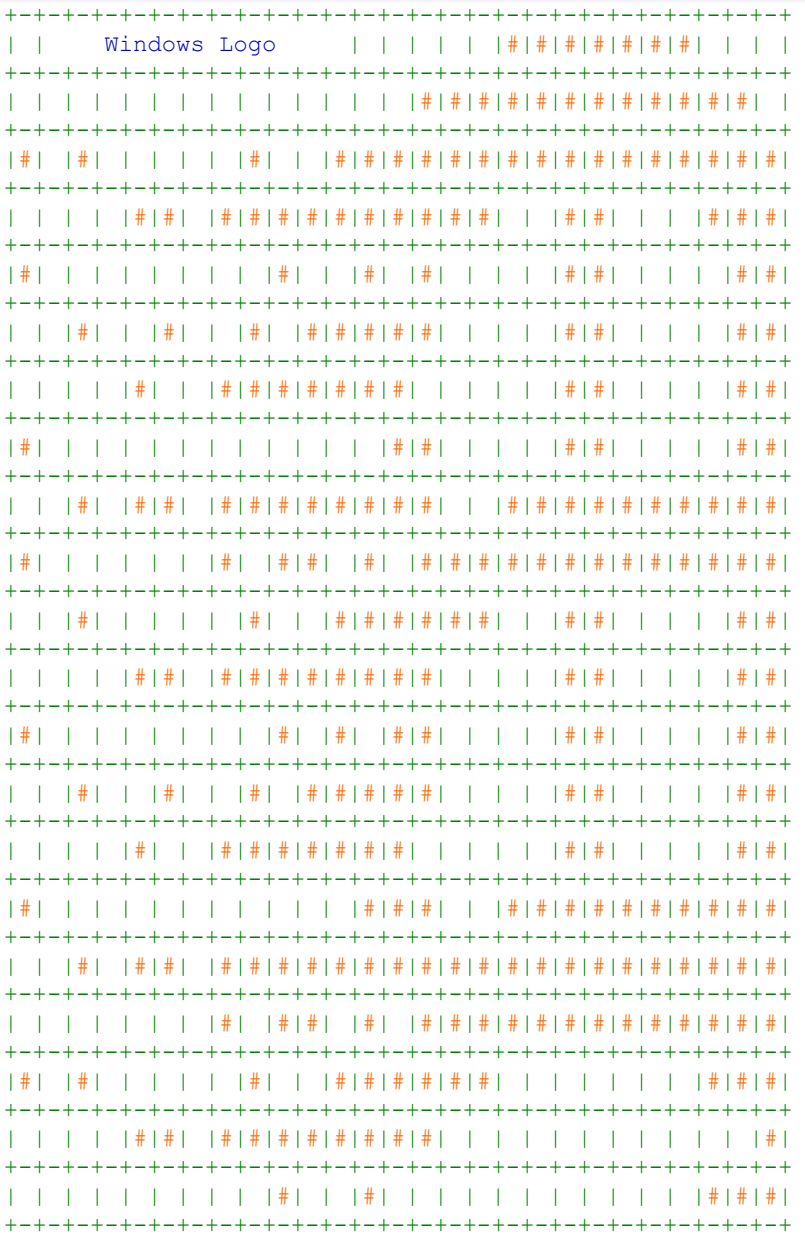
DEMO 3 : Change to Display a New Character



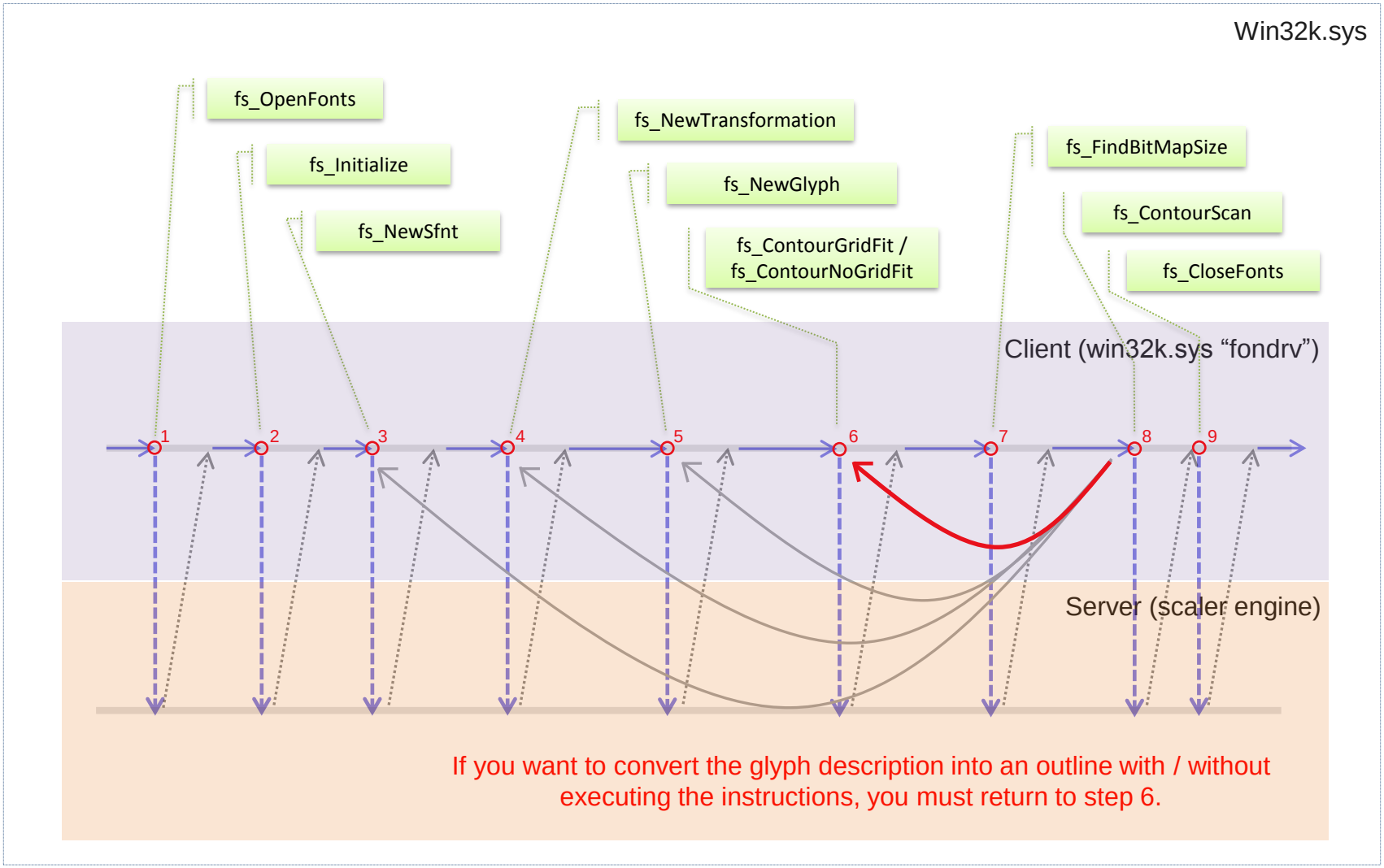
DEMO 3 : Change to Display a New Character



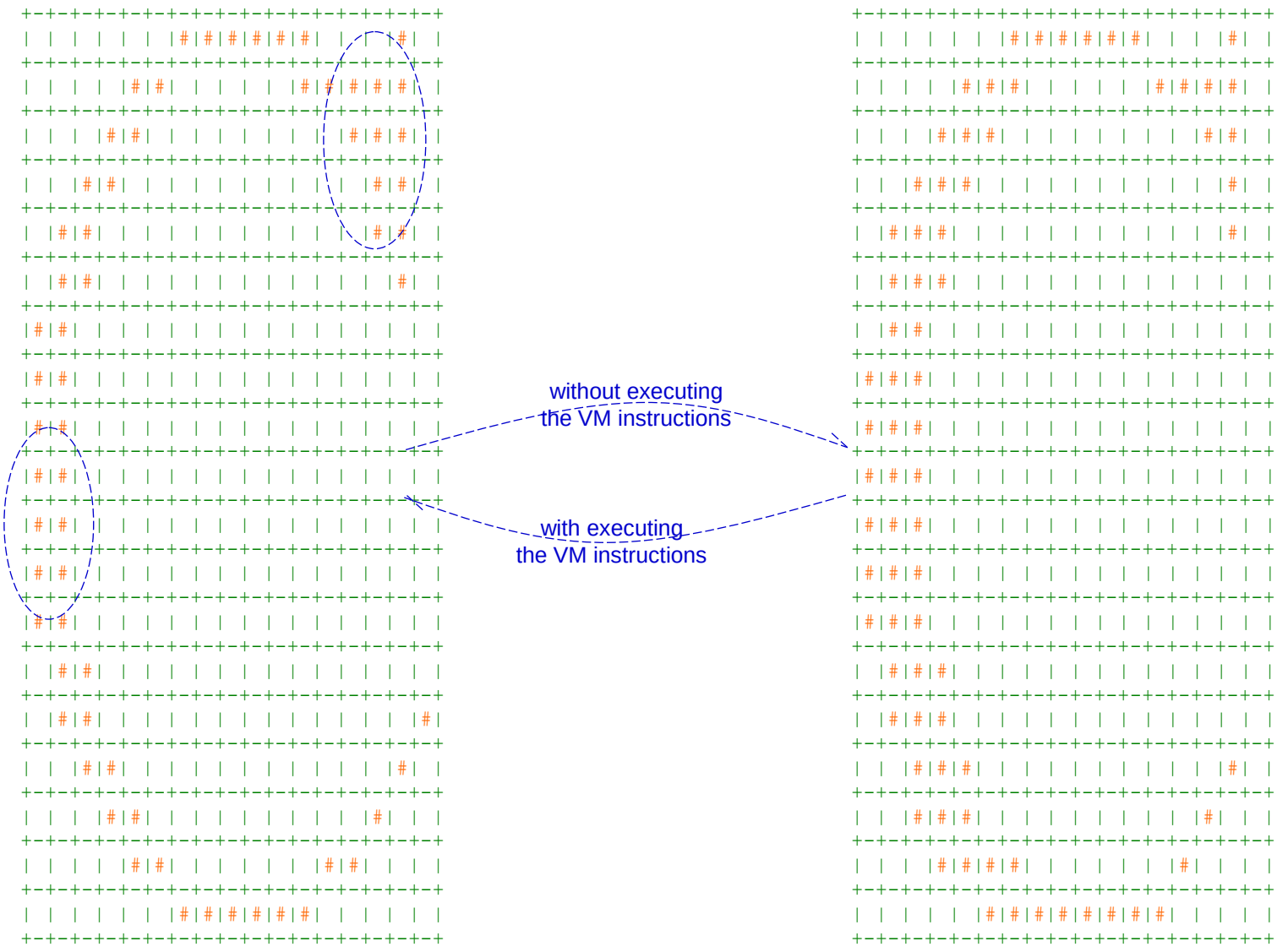
characterCode is 'C'



DEMO 4 : With or Without Executing the VM Instructions



DEMO 4 : With or Without Executing the VM Instructions



Summary

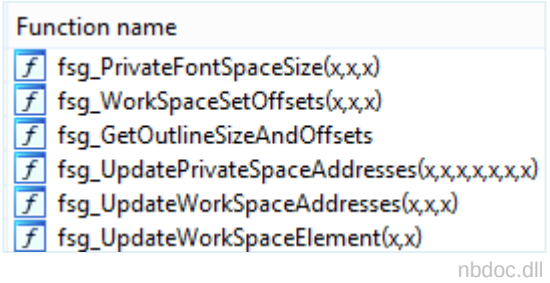
- Memory Allocation Policy of Font Engine

The engine did not allocate any memories, the caller of the font engine is responsible for the allocation and deallocation.

In project "fs-engine," memories are allocated independently, for Win32k.sys, it also has its own assumption about memory allocation, which is quite different from my approach.

- The value of project "fs-engine"

- CVE-2014-1824 and "...\Windows Journal\NBDoc.DLL"



Part Three

The Old New Thing :
CVE-2011-3402, The Vulnerability and Exploitation

Please Tell Me Why

There are some analysis of CVE-2011-3402 in the internet, We all knew that there're some overflows somewhere which cause code execution and EOP (Elevation of Privilege).

But, many problems will come up if we think it over :

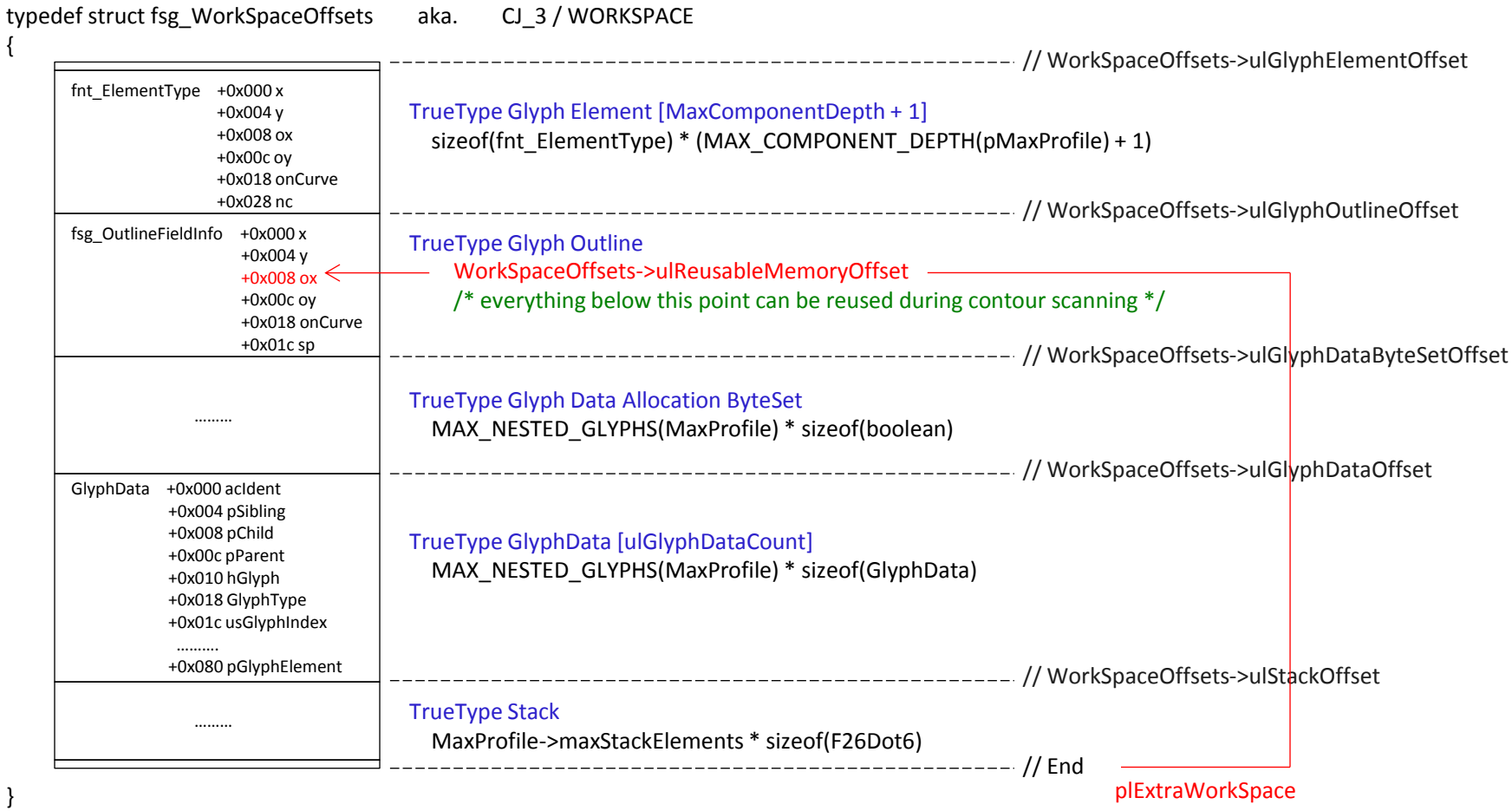
1) What data structure is overflowed, and why is this data structure triggering the vulnerability, not others?

2) How heap overwrite could be used to precisely manipulate the field "cvtCount"? Why there is no need to control the memory layout of the heap?

3) What the whole exploitation looks like?

.....

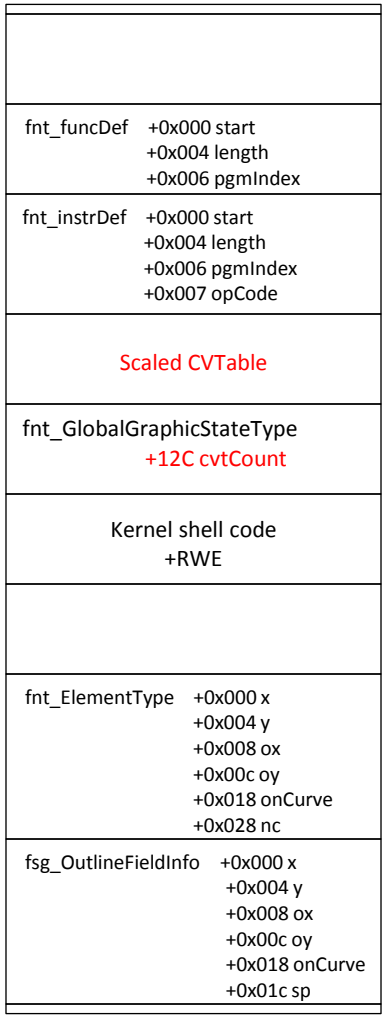
CJ_3 – "Work Space" Memory Layout



CJ_4 – "Private Space" Memory Layout

typedef struct fsg_PrivateSpaceOffsets aka. CJ_4 / PRIVATE SPACE

{



Overread / Overwrite

// PrivateSpaceOffsets->offset_storage

TrueType Storage
sizeof(F26Dot6) * MaxProfile->maxStorage

// PrivateSpaceOffsets->offset_functions

TrueType Function Defs
sizeof(fnt_funcDef) * MaxProfile->maxFunctionDefs

// PrivateSpaceOffsets->offset_instrDefs

TrueType Instruction Defs
sizeof(fnt_instrDef) * MaxProfile->maxInstructionDefs

// PrivateSpaceOffsets->offset_controlValues

TrueType Scaled CVT
sizeof(F26Dot6) * (SFAC_LENGTH(ClientInfo, sfnt_controlValue) / sizeof(sfnt_ControlValue))

// PrivateSpaceOffsets->offset_globalGS

TrueType Global GS
sizeof(fnt_GlobalGraphicStateType)

// PrivateSpaceOffsets->offset_FontProgram

TrueType Font Program
SFAC_LENGTH(ClientInfo, sfnt_fontProgram)

// PrivateSpaceOffsets->offset_PreProgram

TrueType Pre Program
SFAC_LENGTH(ClientInfo, sfnt_preProgram)

// PrivateSpaceOffsets->offset_TwilightZone

TrueType Twilight Element
sizeof(fnt_ElementType)

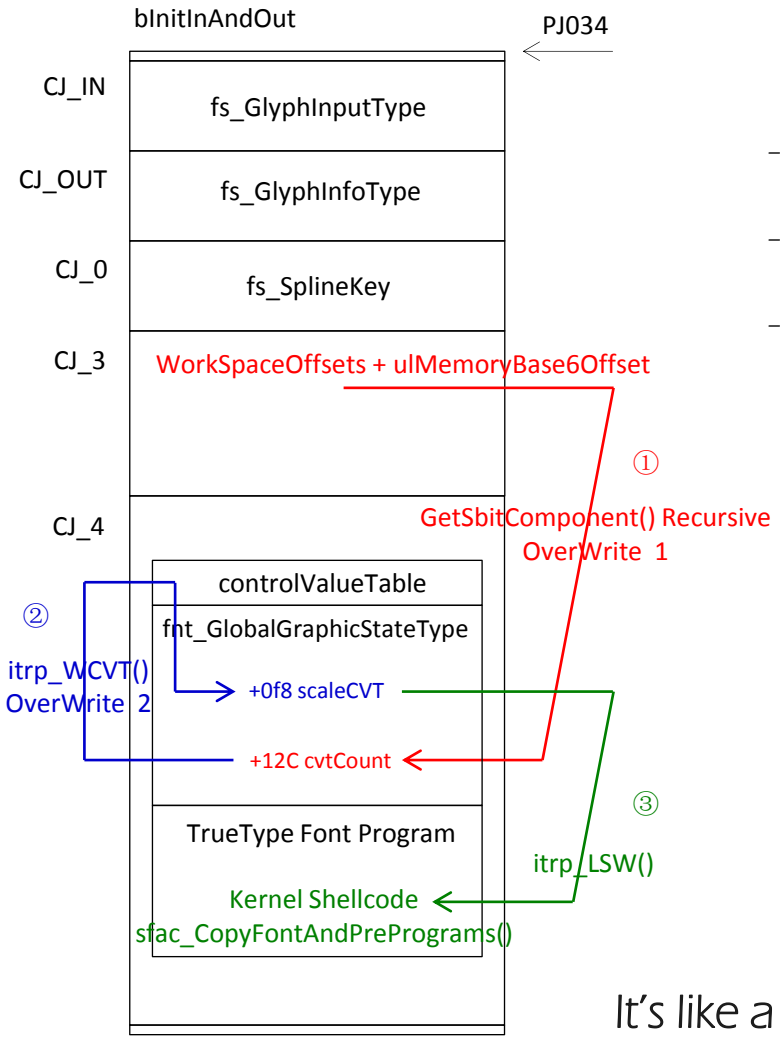
// PrivateSpaceOffsets->offset_TwilightOutline

TrueType Twilight Outline
sizeof(fsg_OutlineFieldInfo)

// End

}

The Key Points of CVE-2011-3402



The Key Points of CVE-2011-3402

- Assumption of memory allocation policy
- CJ_3 could be reused
- Routine GetSbitComponent who reuse CJ_3 does not enforce bound checking in recursion

The corruption of "cvtCount" leads to inconsistency between actual number of "cvtTable" and it claimed

The instructions "itrp_RCVT" and "itrp_WCVT" which are dependent on "cvtCount" and "cvtTable" could also generate Overread and Overwrite again

It's like a Domino Effect with a chain of Overread / write !

Advanced Exploitation Based on the TT Instructions

If you have ever analyzed CVE-2011-3402, whether you have wondered what does the thousands of "TT Interpreter" instructions actually do?
Or, what is the logic of that thousands of instructions?

Exercise : If the memory has the following layout, we try to find the value 0xbf8e8656 in the address 0xe1225030 and replace it,
Could you give me a description of the searching and matching algorithm?

```
          1: kd> dd e1224f80
begin --> e1224f80  00000000 e1224afc e1224f00 e1224f80
          e1224f90  00030004 00040000 00000000 00000000
          e1224fa0  00000000 00000000 00000044 00000000
          e1224fb0  00000000 00000000 00000000 00000040
          e1224fc0  bf8e89e0 00000003 00000000 00000000
          e1224fd0  00000000 00030009 00000080 00000001
          e1224fe0  00000044 00000000 00000000 00000000
          e1224ff0  00000000 00000040 bf8e89e0 00000003
          e1225000  00000000 00000000 00000000 00030009
          e1225010  00000080 00000001 e1224f80 e1224f80
          e1225020  00000000 00000000 bf8e8656 bf8e8656
  end --> e1225030  bf8e8656
```

The three consecutive 0xbf8e8656 in memory are easy-to-spot, and all the matching could be started with this feature.

```
        controlValueTable                                   00000000
-------------------------------------------------------------------
   +0x000 stackBase          : Ptr32 Int4B                  e1224afc
   +0x004 store              : Ptr32 Int4B                  e1224f00
   +0x008 controlValueTable  : Ptr32 Int4B                  e1224f80
   +0x00c pixelsPerEm        : Uint2B                       0004
   +0x00e pointSize          : Uint2B                       0003
   +0x010 fpem               : Int4B                        00040000
   +0x014 engine             : [4] Int4B                    00000000
                                                            00000000
                                                            00000000
                                                            00000000
   +0x024 defaultParBlock    : fnt_ParameterBlock
        +0x000 wTCI             : Int4B                     00000044
        +0x004 sWCI             : Int4B                     00000000
        +0x008 scaledSW         : Int4B                     00000000
        +0x00c scanControl      : Int4B                     00000000
        +0x010 instructControl  : Int4B                     00000000
        +0x014 minimumDistance  : Int4B                     00000040
        +0x018 RoundValue       : Ptr32 long                bf8e89e0
        +0x01c RoundValue2      : Int4B                     00000003
        +0x020 periodMask       : Int4B                     00000000
        +0x024 period45         : Int2B                     0000
        +0x026 period           : Int2B                     0000
        +0x028 phase            : Int2B                     0000
        +0x02a threshold        : Int2B                     0000
        +0x02c deltaBase        : Int2B                     0009
        +0x02e deltaShift       : Int2B                     0003
        +0x030 angleWeight      : Int2B                     0080
        +0x032 sW               : Int2B                     0000
        +0x034 autoFlip         : Char                      01
        +0x035 pad              : Char                      00
        +0x036 pad2             : Int2B                     0000
   +0x058 localParBlock      : fnt_ParameterBlock
        +0x000 wTCI             : Int4B                     00000044
        +0x004 sWCI             : Int4B                     00000000
        +0x008 scaledSW         : Int4B                     00000000
        +0x00c scanControl      : Int4B                     00000000
        +0x010 instructControl  : Int4B                     00000000
        +0x014 minimumDistance  : Int4B                     00000040
        +0x018 RoundValue       : Ptr32 long                bf8e89e0
        +0x01c RoundValue2      : Int4B                     00000003
        +0x020 periodMask       : Int4B                     00000000
        +0x024 period45         : Int2B                     0000
        +0x026 period           : Int2B                     0000
        +0x028 phase            : Int2B                     0000
        +0x02a threshold        : Int2B                     0000
        +0x02c deltaBase        : Int2B                     0009
        +0x02e deltaShift       : Int2B                     0003
        +0x030 angleWeight      : Int2B                     0080
        +0x032 sW               : Int2B                     0000
        +0x034 autoFlip         : Char                      01
        +0x035 pad              : Char                      00
        +0x036 pad2             : Int2B                     0000
   +0x08c funcDef            : Ptr32 fnt_funcDef           e1224f80
   +0x090 instrDef           : Ptr32 fnt_instrDef          e1224f80
   +0x094 ScaleFuncXBase     : Ptr32 long                  00000000
   +0x098 ScaleFuncYBase     : Ptr32 long                  00000000
   +0x09c ScaleFuncX         : Ptr32 long                  bf8e8656
   +0x0a0 ScaleFuncY         : Ptr32 long                  bf8e8656
   +0x0a4 ScaleFuncCVT       : Ptr32 long                  bf8e8656  <------- object hook ---------+
   +0x0a8 pgmList            : [2] fnt_pgmList                                                      |
        +0x000 Instruction      : Ptr32 UChar              e1260bb3                                  |
        +0x004 Length           : Uint4B                   0000000d                                  |
        +0x000 Instruction      : Ptr32 UChar              e1225318  -- shellcode at offset +50 -->+
        +0x004 Length           : Uint4B                   0003b89b
```

The Answer

Based on project “fs-engine”,
let me give some meaning to these data

And then, let me put some RESTRICTIONS to this exercise :

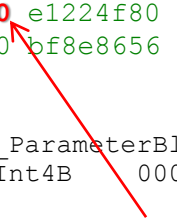
Please make use of “TT Rasterizer Interpreter” instructions to implement your search algorithm ...

Duqu Team, Three years ago !

Little Trick : itrp_FLIPON and itrp_FLIPOFF

```
1: kd> dd e1224f80
e1224f80  00000000 e1224afc e1224f00 e1224f80
e1224f90  00030004 00040000 00000000 00000000
e1224fa0  00000000 00000000 00000044 00000000
e1224fb0  00000000 00000000 00000000 00000040
e1224fc0  bf8e89e0 00000003 00000000 00000000
e1224fd0  00000000 00030009 00000080 00000001
e1224fe0  00000044 00000000 00000000 00000000
e1224ff0  00000000 00000040 bf8e89e0 00000003
e1225000  00000000 00000000 00000000 00030009
e1225010  00000080 00000000 e1224f80 e1224f80
e1225020  00000000 00000000 bf8e8656 bf8e8656
e1225030  bf8e8656

+0x058 localParBlock   : fnt_ParameterBlock
     +0x000 wTCI          : Int4B    00000044
      .....
     +0x034 autoFlip      : Char     00
     +0x035 pad           : Char     00
```



while Loop, Round 1

1) FLIP OFF, Set localParBlock.autoFlip to 0

2) The first overread fetch the value of A

A == 0x00000000

3) FLIP ON, Set localParBlock.autoFlip to 1

4) Second overread fetch the value of B

B == 0x00000000

A is equal to B

Little Trick : itrp_FLIPON and itrp_FLIPOFF

```
1: kd> dd e1224f80
e1224f80  00000000 e1224afc e1224f00 e1224f80
e1224f90  00030004 00040000 00000000 00000000
e1224fa0  00000000 00000000 00000044 00000000
e1224fb0  00000000 00000000 00000000 00000040
e1224fc0  bf8e89e0 00000003 00000000 00000000
e1224fd0  00000000 00030009 00000080 00000001
e1224fe0  00000044 00000000 00000000 00000000
e1224ff0  00000000 00000040 bf8e89e0 00000003
e1225000  00000000 00000000 00000000 00030009
e1225010  00000080 00000000 e1224f80 e1224f80
e1225020  00000000 00000000 bf8e8656 bf8e8656
e1225030  bf8e8656

+0x058 localParBlock  : fnt_ParameterBlock
     +0x000 wTCI          : Int4B    00000044
      .....
     +0x034 autoFlip      : Char     00
     +0x035 pad           : Char     00
```

while Loop, Round 1

1) FLIP OFF, Set localParBlock.autoFlip to 0

2) The first overread fetch the value of A

A == 0x00000000

3) FLIP ON, Set localParBlock.autoFlip to 1

4) Second overread fetch the value of B

B == 0x00000000

A is equal to B

Little Trick : itrp_FLIPON and itrp_FLIPOFF

```
1: kd> dd e1224f80
e1224f80  00000000 e1224afc e1224f00 e1224f80
e1224f90  00030004 00040000 00000000 00000000
e1224fa0  00000000 00000000 00000044 00000000
e1224fb0  00000000 00000000 00000000 00000040
e1224fc0  bf8e89e0 00000003 00000000 00000000
e1224fd0  00000000 00030009 00000080 00000001
e1224fe0  00000044 00000000 00000000 00000000
e1224ff0  00000000 00000040 bf8e89e0 00000003
e1225000  00000000 00000000 00000000 00030009
e1225010  00000080 00000001 e1224f80 e1224f80
e1225020  00000000 00000000 bf8e8656 bf8e8656
e1225030  bf8e8656

+0x058 localParBlock  : fnt_ParameterBlock
     +0x000 wTCI          : Int4B    00000044
      .....
     +0x034 autoFlip      : Char     01
     +0x035 pad           : Char     00
```

while Loop, Round 1

1) FLIP OFF, Set localParBlock.autoFlip to 0

2) The first overread fetch the value of A

A == 0x00000000

3) FLIP ON, Set localParBlock.autoFlip to 1

4) Second overread fetch the value of B

B == 0x00000000

A is equal to B

Little Trick : itrp_FLIPON and itrp_FLIPOFF

```
1: kd> dd e1224f80
e1224f80  00000000 e1224afc e1224f00 e1224f80
e1224f90  00030004 00040000 00000000 00000000
e1224fa0  00000000 00000000 00000044 00000000
e1224fb0  00000000 00000000 00000000 00000040
e1224fc0  bf8e89e0 00000003 00000000 00000000
e1224fd0  00000000 00030009 00000080 00000001
e1224fe0  00000044 00000000 00000000 00000000
e1224ff0  00000000 00000040 bf8e89e0 00000003
e1225000  00000000 00000000 00000000 00030009
e1225010  00000080 00000001 e1224f80 e1224f80
e1225020  00000000 00000000 bf8e8656 bf8e8656
e1225030  bf8e8656

+0x058 localParBlock  : fnt_ParameterBlock
     +0x000 wTCI          : Int4B    00000044
      .....
     +0x034 autoFlip      : Char     01
     +0x035 pad           : Char     00
```

while Loop, Round 1

1) FLIP OFF, Set localParBlock.autoFlip to 0

2) The first overread fetch the value of A

A == 0x00000000

3) FLIP ON, Set localParBlock.autoFlip to 1

4) Second overread fetch the value of B

B == 0x00000000

A is equal to B

Little Trick : itrp_FLIPON and itrp_FLIPOFF

```
1: kd> dd e1224f80
e1224f80  00000000 e1224afc e1224f00 e1224f80
e1224f90  00030004 00040000 00000000 00000000
e1224fa0  00000000 00000000 00000044 00000000
e1224fb0  00000000 00000000 00000000 00000040
e1224fc0  bf8e89e0 00000003 00000000 00000000
e1224fd0  00000000 00030009 00000080 00000001
e1224fe0  00000044 00000000 00000000 00000000
e1224ff0  00000000 00000040 bf8e89e0 00000003
e1225000  00000000 00000000 00000000 00030009
e1225010  00000080 00000000 e1224f80 e1224f80
e1225020  00000000 00000000 bf8e8656 bf8e8656
e1225030  bf8e8656

+0x058 localParBlock   : fnt_ParameterBlock
     +0x000 wTCI          : Int4B    00000044
      .....
     +0x034 autoFlip      : Char     00
     +0x035 pad           : Char     00
```

while Loop, Round 2

1) FLIP OFF, Set localParBlock.autoFlip to 0

2) The first overread fetch the value of A

A == 0xe1224afc

3) FLIP ON, Set localParBlock.autoFlip to 1

4) Second overread fetch the value of B

B == 0xe1224afc

A is equal to B

Little Trick : itrp_FLIPON and itrp_FLIPOFF

```
1: kd> dd e1224f80
e1224f80  00000000 e1224afc e1224f00 e1224f80
e1224f90  00030004 00040000 00000000 00000000
e1224fa0  00000000 00000000 00000044 00000000
e1224fb0  00000000 00000000 00000000 00000040
e1224fc0  bf8e89e0 00000003 00000000 00000000
e1224fd0  00000000 00030009 00000080 00000001
e1224fe0  00000044 00000000 00000000 00000000
e1224ff0  00000000 00000040 bf8e89e0 00000003
e1225000  00000000 00000000 00000000 00030009
e1225010  00000080 00000000 e1224f80 e1224f80
e1225020  00000000 00000000 bf8e8656 bf8e8656
e1225030  bf8e8656

+0x058 localParBlock  : fnt_ParameterBlock
     +0x000 wTCI          : Int4B    00000044
      .....
     +0x034 autoFlip      : Char     00
     +0x035 pad           : Char     00
```

while Loop, Round 2

1) FLIP OFF, Set localParBlock.autoFlip to 0

2) The first overread fetch the value of A

A == 0xe1224afc

3) FLIP ON, Set localParBlock.autoFlip to 1

4) Second overread fetch the value of B

B == 0xe1224afc

A is equal to B

Little Trick : itrp_FLIPON and itrp_FLIPOFF

```
1: kd> dd e1224f80
e1224f80  00000000 e1224afc e1224f00 e1224f80
e1224f90  00030004 00040000 00000000 00000000
e1224fa0  00000000 00000000 00000044 00000000
e1224fb0  00000000 00000000 00000000 00000040
e1224fc0  bf8e89e0 00000003 00000000 00000000
e1224fd0  00000000 00030009 00000080 00000001
e1224fe0  00000044 00000000 00000000 00000000
e1224ff0  00000000 00000040 bf8e89e0 00000003
e1225000  00000000 00000000 00000000 00030009
e1225010  00000080 00000001 e1224f80 e1224f80
e1225020  00000000 00000000 bf8e8656 bf8e8656
e1225030  bf8e8656

+0x058 localParBlock   : fnt_ParameterBlock
     +0x000 wTCI          : Int4B    00000044
      .....
     +0x034 autoFlip      : Char     01
     +0x035 pad           : Char     00
```

while Loop, Round 2

1) FLIP OFF, Set localParBlock.autoFlip to 0

2) The first overread fetch the value of A

A == 0xe1224afc

3) FLIP ON, Set localParBlock.autoFlip to 1

4) Second overread fetch the value of B

B == 0xe1224afc

A is equal to B

Little Trick : itrp_FLIPON and itrp_FLIPOFF

```
1: kd> dd e1224f80
e1224f80  00000000 e1224afc e1224f00 e1224f80
e1224f90  00030004 00040000 00000000 00000000
e1224fa0  00000000 00000000 00000044 00000000
e1224fb0  00000000 00000000 00000000 00000040
e1224fc0  bf8e89e0 00000003 00000000 00000000
e1224fd0  00000000 00030009 00000080 00000001
e1224fe0  00000044 00000000 00000000 00000000
e1224ff0  00000000 00000040 bf8e89e0 00000003
e1225000  00000000 00000000 00000000 00030009
e1225010  00000080 00000001 e1224f80 e1224f80
e1225020  00000000 00000000 bf8e8656 bf8e8656
e1225030  bf8e8656

+0x058 localParBlock  : fnt_ParameterBlock
     +0x000 wTCI          : Int4B    00000044
      .....
     +0x034 autoFlip      : Char     01
     +0x035 pad           : Char     00
```

while Loop, Round 2

1) FLIP OFF, Set localParBlock.autoFlip to 0

2) The first overread fetch the value of A

A == 0xe1224afc

3) FLIP ON, Set localParBlock.autoFlip to 1

4) Second overread fetch the value of B

B == 0xe1224afc

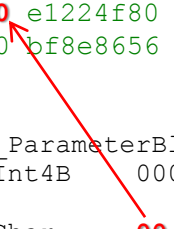
A is equal to B

Round 3 / 4 / 5 .....

Little Trick : itrp_FLIPON and itrp_FLIPOFF

```
1: kd> dd e1224f80
e1224f80  00000000 e1224afc e1224f00 e1224f80
e1224f90  00030004 00040000 00000000 00000000
e1224fa0  00000000 00000000 00000044 00000000
e1224fb0  00000000 00000000 00000000 00000040
e1224fc0  bf8e89e0 00000003 00000000 00000000
e1224fd0  00000000 00030009 00000080 00000001
e1224fe0  00000044 00000000 00000000 00000000
e1224ff0  00000000 00000040 bf8e89e0 00000003
e1225000  00000000 00000000 00000000 00030009
e1225010  00000080 00000000 e1224f80 e1224f80
e1225020  00000000 00000000 bf8e8656 bf8e8656
e1225030  bf8e8656

+0x058 localParBlock  : fnt_ParameterBlock
     +0x000 wTCI          : Int4B    00000044
      .....
     +0x034 autoFlip      : Char     00
     +0x035 pad           : Char     00
```



while Loop, Round 38

1) FLIP OFF, Set localParBlock.autoFlip to 0

2) The first overread fetch the value of A

A == 0x00000000

3) FLIP ON, Set localParBlock.autoFlip to 1

4) Second overread fetch the value of B

B == 0x00000001

A is NOT equal to B

Little Trick : itrp_FLIPON and itrp_FLIPOFF

```
1: kd> dd e1224f80
e1224f80  00000000 e1224afc e1224f00 e1224f80
e1224f90  00030004 00040000 00000000 00000000
e1224fa0  00000000 00000000 00000044 00000000
e1224fb0  00000000 00000000 00000000 00000040
e1224fc0  bf8e89e0 00000003 00000000 00000000
e1224fd0  00000000 00030009 00000080 00000001
e1224fe0  00000044 00000000 00000000 00000000
e1224ff0  00000000 00000040 bf8e89e0 00000003
e1225000  00000000 00000000 00000000 00030009
e1225010  00000080 00000000 e1224f80 e1224f80
e1225020  00000000 00000000 bf8e8656 bf8e8656
e1225030  bf8e8656
```

```
+0x058 localParBlock  : fnt_ParameterBlock
     +0x000 wTCI          : Int4B    00000044
      .....
     +0x034 autoFlip      : Char     00
     +0x035 pad           : Char     00
```

while Loop, Round 38

1) FLIP OFF, Set localParBlock.autoFlip to 0

2) The first overread fetch the value of A

A == 0x00000000

3) FLIP ON, Set localParBlock.autoFlip to 1

4) Second overread fetch the value of B

B == 0x00000001

A is NOT equal to B

Little Trick : itrp_FLIPON and itrp_FLIPOFF

```
1: kd> dd e1224f80
e1224f80  00000000 e1224afc e1224f00 e1224f80
e1224f90  00030004 00040000 00000000 00000000
e1224fa0  00000000 00000000 00000044 00000000
e1224fb0  00000000 00000000 00000000 00000040
e1224fc0  bf8e89e0 00000003 00000000 00000000
e1224fd0  00000000 00030009 00000080 00000001
e1224fe0  00000044 00000000 00000000 00000000
e1224ff0  00000000 00000040 bf8e89e0 00000003
e1225000  00000000 00000000 00000000 00030009
e1225010  00000080 00000001 e1224f80 e1224f80
e1225020  00000000 00000000 bf8e8656 bf8e8656
e1225030  bf8e8656

+0x058 localParBlock  : fnt_ParameterBlock
     +0x000 wTCI          : Int4B    00000044
      .....
     +0x034 autoFlip      : Char     01
     +0x035 pad           : Char     00
```

while Loop, Round 38

1) FLIP OFF, Set localParBlock.autoFlip to 0

2) The first overread fetch the value of A

A == 0x00000000

3) FLIP ON, Set localParBlock.autoFlip to 1

4) Second overread fetch the value of B

B == 0x00000001

A is NOT equal to B

Little Trick : itrp_FLIPON and itrp_FLIPOFF

```
1: kd> dd e1224f80
e1224f80  00000000 e1224afc e1224f00 e1224f80
e1224f90  00030004 00040000 00000000 00000000
e1224fa0  00000000 00000000 00000044 00000000
e1224fb0  00000000 00000000 00000000 00000040
e1224fc0  bf8e89e0 00000003 00000000 00000000
e1224fd0  00000000 00030009 00000080 00000001
e1224fe0  00000044 00000000 00000000 00000000
e1224ff0  00000000 00000040 bf8e89e0 00000003
e1225000  00000000 00000000 00000000 00030009
e1225010  00000080 00000001 e1224f80 e1224f80
e1225020  00000000 00000000 bf8e8656 bf8e8656
e1225030  bf8e8656
```

```
+0x058 localParBlock  : fnt_ParameterBlock
     +0x000 wTCI          : Int4B    00000044
      .....
     +0x034 autoFlip      : Char     01
     +0x035 pad           : Char     00
```

while Loop, Round 38

1) FLIP OFF, Set localParBlock.autoFlip to 0

2) The first overread fetch the value of A

A == 0x00000000

3) FLIP ON, Set localParBlock.autoFlip to 1

4) Second overread fetch the value of B

B == 0x00000001

A is NOT equal to B

Little Trick : itrp_FLIPON and itrp_FLIPOFF

```
1: kd> dd e1224f80
e1224f80  00000000 e1224afc e1224f00 e1224f80
e1224f90  00030004 00040000 00000000 00000000
e1224fa0  00000000 00000000 00000044 00000000
e1224fb0  00000000 00000000 00000000 00000040
e1224fc0  bf8e89e0 00000003 00000000 00000000
e1224fd0  00000000 00030009 00000080 00000001
e1224fe0  00000044 00000000 00000000 00000000
e1224ff0  00000000 00000040 bf8e89e0 00000003
e1225000  00000000 00000000 00000000 00030009
e1225010  00000080 00000001 e1224f80 e1224f80
e1225020  00000000 00000000 bf8e8656 bf8e8656
e1225030  bf8e8656
```

```
+0x058 localParBlock   : fnt_ParameterBlock
     +0x000 wTCI          : Int4B    00000044
      .....
     +0x034 autoFlip      : Char     01
     +0x035 pad           : Char     00
```

while Loop, Round 38

1) FLIP OFF, Set localParBlock.autoFlip to 0

2) The first overread fetch the value of A

A == 0x00000000

3) FLIP ON, Set localParBlock.autoFlip to 1

4) Second overread fetch the value of B

B == 0x00000001

A is NOT equal to B

This "trick" enable code to known exactly when it has reached the location "0xe1225014",

Then we could check whether there're "three consecutive 0xbf8e8656", this method turns out to be more stable than other linear methods.

Advanced Exploitation Based on the TT Instructions

1) put the overread "index" into storage area

2) flip-off, set localParBlock.autoFlip to 0

3) first overread to retrieve the value A with CVTable[index]

4) flip-on, set localParBlock.autoFlip to 1

5) second overread to retrieve the value B with CVTable[index]

6) if B and A are equal, then index++ and return to step 1,
if they're not equal then start to search for three consecutive constant features

7) orderly read the values, check if they're what we are looking for

8) find the target, object hook / hijack, privilege elevation

That's the whole story behind that thousands of instructions !

Summary of CVE-2011-3402

the contiguous memory allocation policy of CJ_3 and CJ_4

memory reuse of CJ_3

absence of bounds-checking

→ the first overwrite

→ cvtCount > the actual size of cvtTable

→ the second overread and overwrite

object hijack

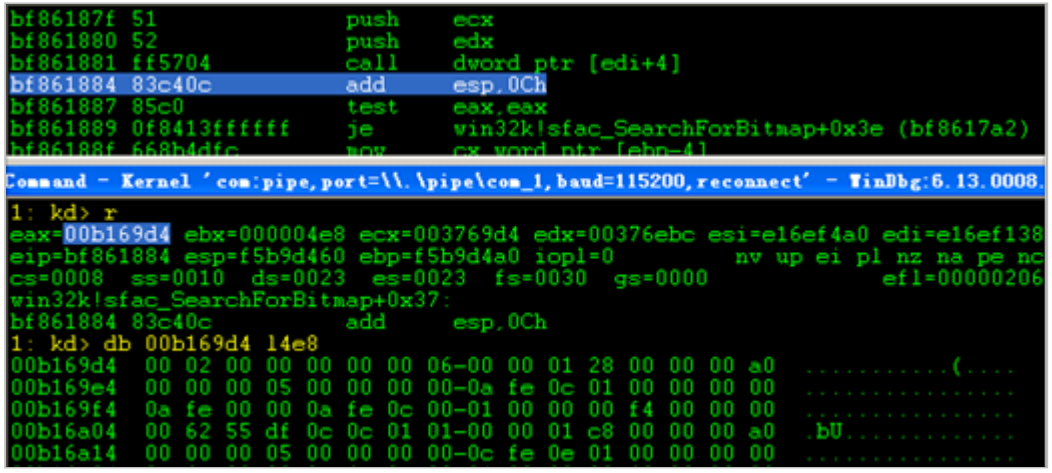
elevation of privilege

making use of the TT instructions to conduct signature-based search

CJ_4 . Font Program is +RWX

What Does It Really Mean by Moving Engine into Kernel ?

Really ? read Ring-3 data directly ?
Be careful the TOCTTOU problems !



Part Four

TOCTTOU Problems in the Font Scaler Engine

The TOCTTOU Problem

Time-of-check to time-of-use

"Inconsistency between the checking of a condition and the use of the results of that check."

en.wikipedia.org/wiki/Time_of_check_to_time_of_use

Double or multiple fetch is a specific case of TOCTTOU.

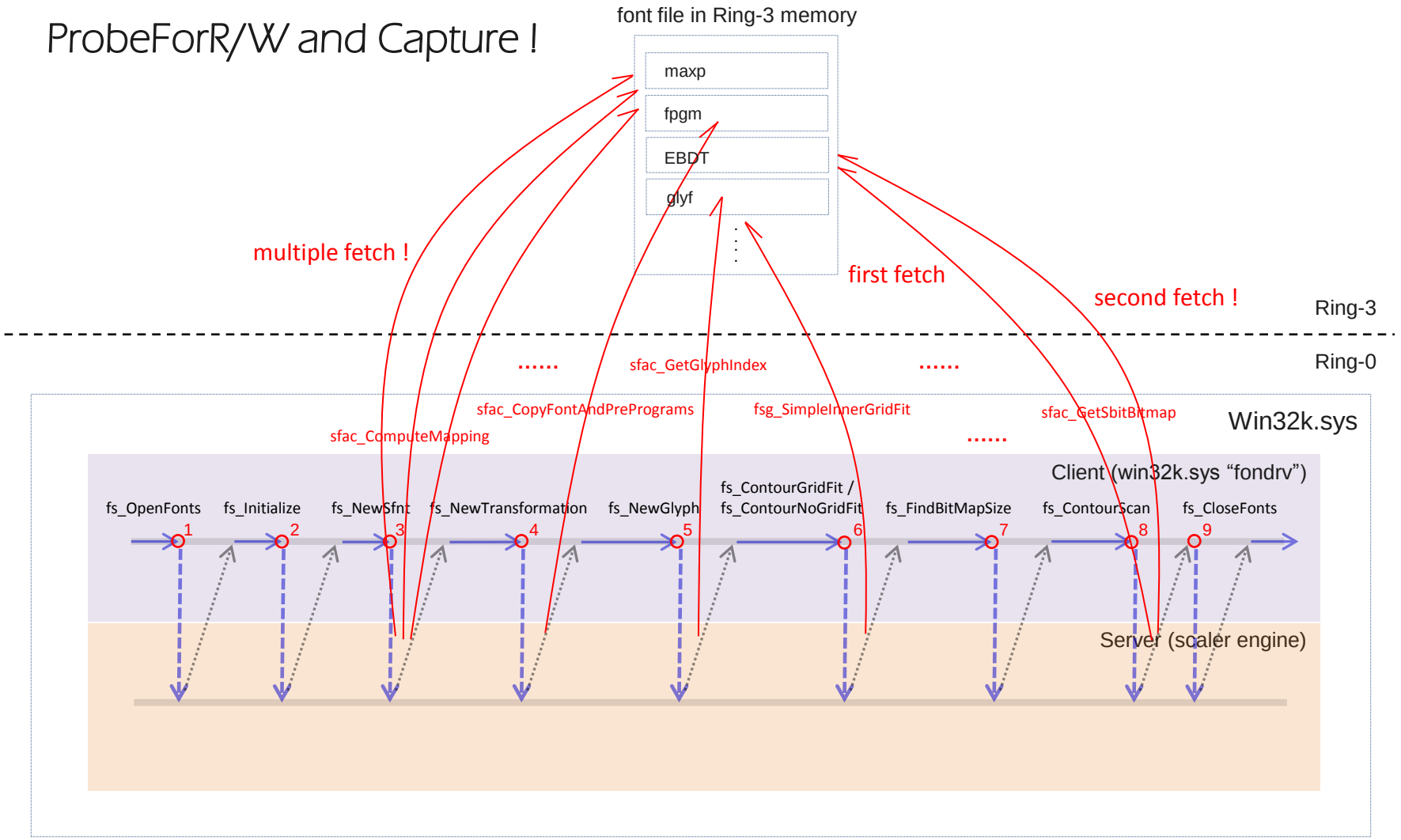
The right way : ProbeForR/W and Capture

"When the kernel captures the user mode data, it works with the local copy and is not affected by any changes that happen to the user mode copy.

This capture avoids race conditions involving different values of the data being returned from subsequent user mode fetches."

http://blogs.technet.com/b/srd/archive/2008/10/14/ms08-061-the-case-of-the-kernel-mode-double-fetch.aspx

TOCTTOU Problems in the Font Scaler Engine



Identifying 0-days via Bochspwn

Bochspwn, SyScan 2013 and BlackHat USA 2013

Bochspwn is an instrumentation module for Bochs for memory access pattern analysis.



j00ru and Gynvael Coldwind

```
==========================================================================
= #001 == 27 instances of double fetches in win32k.sys functions performing user-mode callbacks. =
==========================================================================
```

https://docs.google.com/document/d/1eQamOx1Z4bwm7J-FMHgNOw8WJ0IJFdNgQdK9vlLRHLo/edit

- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1254)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1255)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1256)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1257)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1258)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1259)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1260)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1261)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1262)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1263)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1264)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1265)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1266)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1267)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1268)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1269)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1270)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1271)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1272)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1273)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1274)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1275)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1276)
- Mateusz "j00ru" Jurczyk of Google Inc for reporting the Win32k Race Condition Vulnerability (CVE-2013-1277)

MS13-036

Case Study : CVE-2013-1341 / MS13-076

Before

```
while ( !found )
{
  LOBYTE(v10) = *(_WORD *)table >> 8;
  HIBYTE(v10) = *(_WORD *)table;
```

Out-Of-Bounds

After

```
while ( !found )
{
  if ( table > v8 + v13 - 8 )
    goto LABEL_36;

  LOBYTE(v14) = *(_WORD *)table >> 8;
  HIBYTE(v14) = *(_WORD *)table;
```

Before

```
v14 = *(_DWORD *)(v4 + 16);
*(_DWORD *)(v4 + 16) = v14 + 6;
LOBYTE(v15) = *(_WORD *)(v14 + v7) >> 8;
HIBYTE(v15) = *(_WORD *)(v14 + v7);
*(_WORD *)(v4 + 204) = v15;
```

Integer Overflow

After

```
v15 = *(_DWORD *)(v5 + 16);
if ( ULongAdd(v8, *(_DWORD *)(v5 + 16), (unsigned __int32)&v35, v25, v28) >= 0
  && ULongAdd(v15, 6, (unsigned __int32)&v32, v26, v29) >= 0 )
```

https://technet.microsoft.com/library/security/ms13-076

Identifying 0-days via Bochspwn

Great Job ! Bochspwn !

But, What is Left for Us / for the 99% ?

Any New Ideal ? Am I Gonna Lose My Job ?

Further research

- Kernel instrumentation potential is far from being exhausted.
  - in fact, there are hundreds* of low-hanging fruit waiting to be found.
  - so far it seems most are in Windows.
- Hack on kfetch-toolkit
  - port to other platforms (more exotic?).
  - find novel patterns, models or whole bug classes.
  - improve coverage.
  - test other presented approaches.

* personal estimate.

black hat USA 2013

What is Your Answer ?

My answer is :

1) Find the difficult-to-cover code paths for Bochspwn

2) Find the new pattern of vulnerabilities that unable discovered by Bochspwn

Code Coverage Analysis

Bochspwn is triggered by input data, this could provide us the chance, which is the code coverage.

I find something interesting, which its logic is as follows:

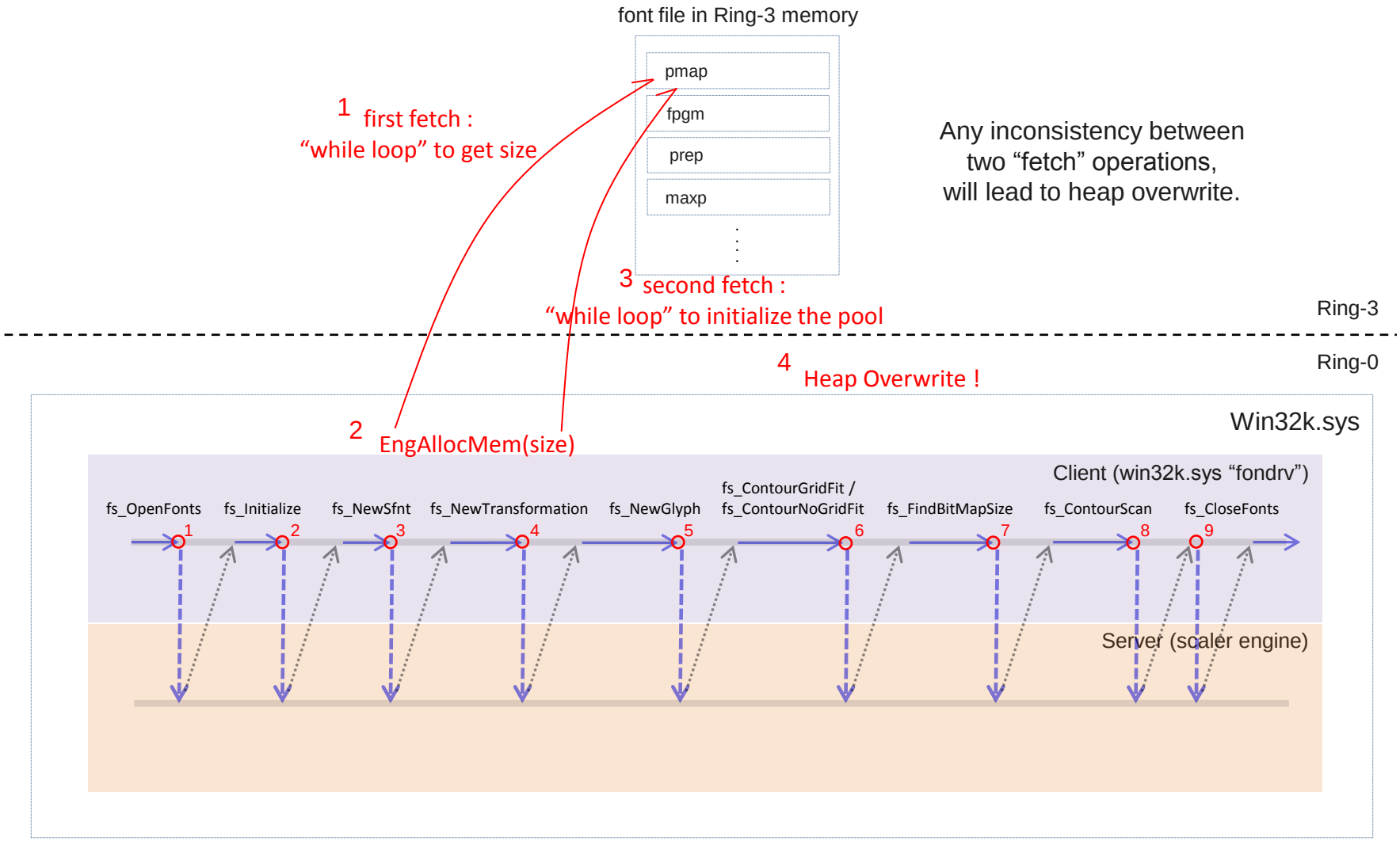
1) engine fetch some data from user-mode, and computes the "Size" of that data

2) allocate memory with "Size" from heap

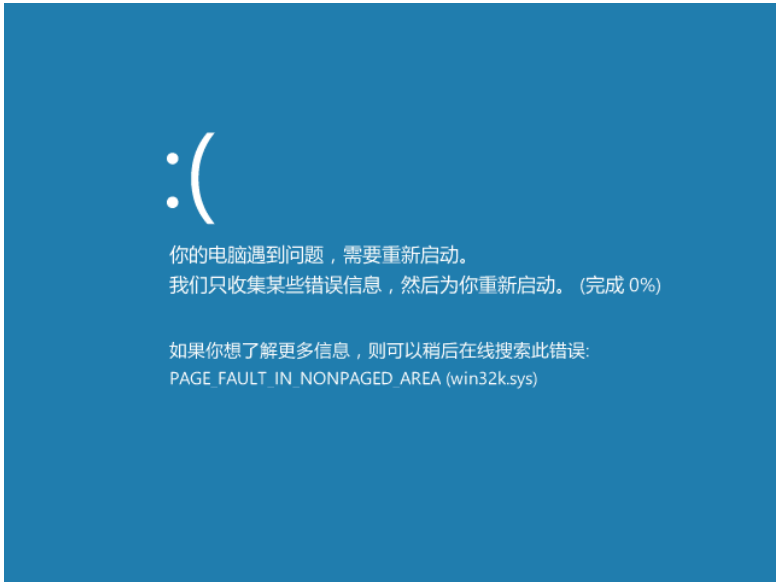
3) engine double fetch and re-computes the "Size" to initialize the kernel heap

Will be fixed on August 12, Patch Tuesday.

Code Coverage Analysis



DEMO 1 : The Double Fetch Problem




```
Sorry, no more details before Patch Tuesday
--------------------------------------------

   esi=000031fe edi=80872010
   ...
   0008:8fcf40dc 8324f700        and     dword ptr [edi+esi*8],0 ds:0023:8088b000=????????
```

The Kernel Font Cache Problem

The binary audit afterwards leads me to another interesting problem, which its logic is as follows:

1) engine fetch data from kernel-mode font cache, and computes the "Size" of that data

2) allocate memory with "Size" from heap

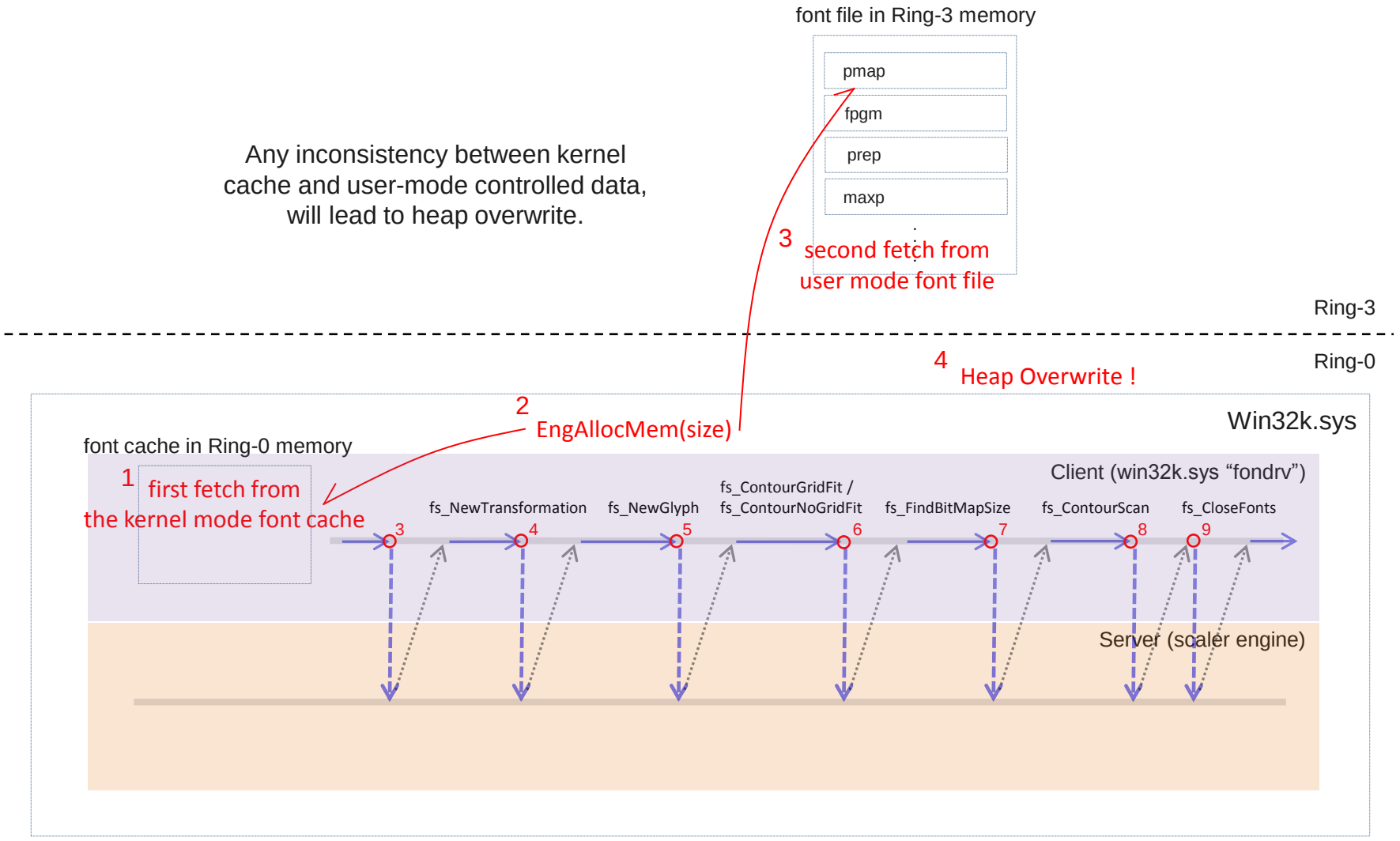
3) engine fetch data from user-mode, and re-computes the "Size" to initialize the kernel heap

This vulnerability is variant of "double fetch". The pattern is: Firstly, fetch from kernel, Secondly, fetch from user-mode controlled data.

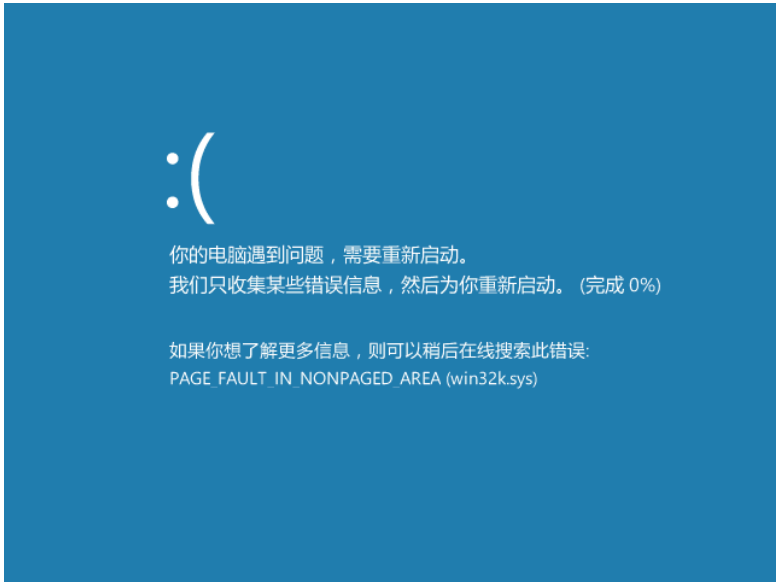
Bochspwn has not found them simply because there're no expected second-time fetch operation.

Will be fixed on August 12, Patch Tuesday.

The Kernel Font Cache Problem



DEMO 2 : The Kernel Font Cache Problem



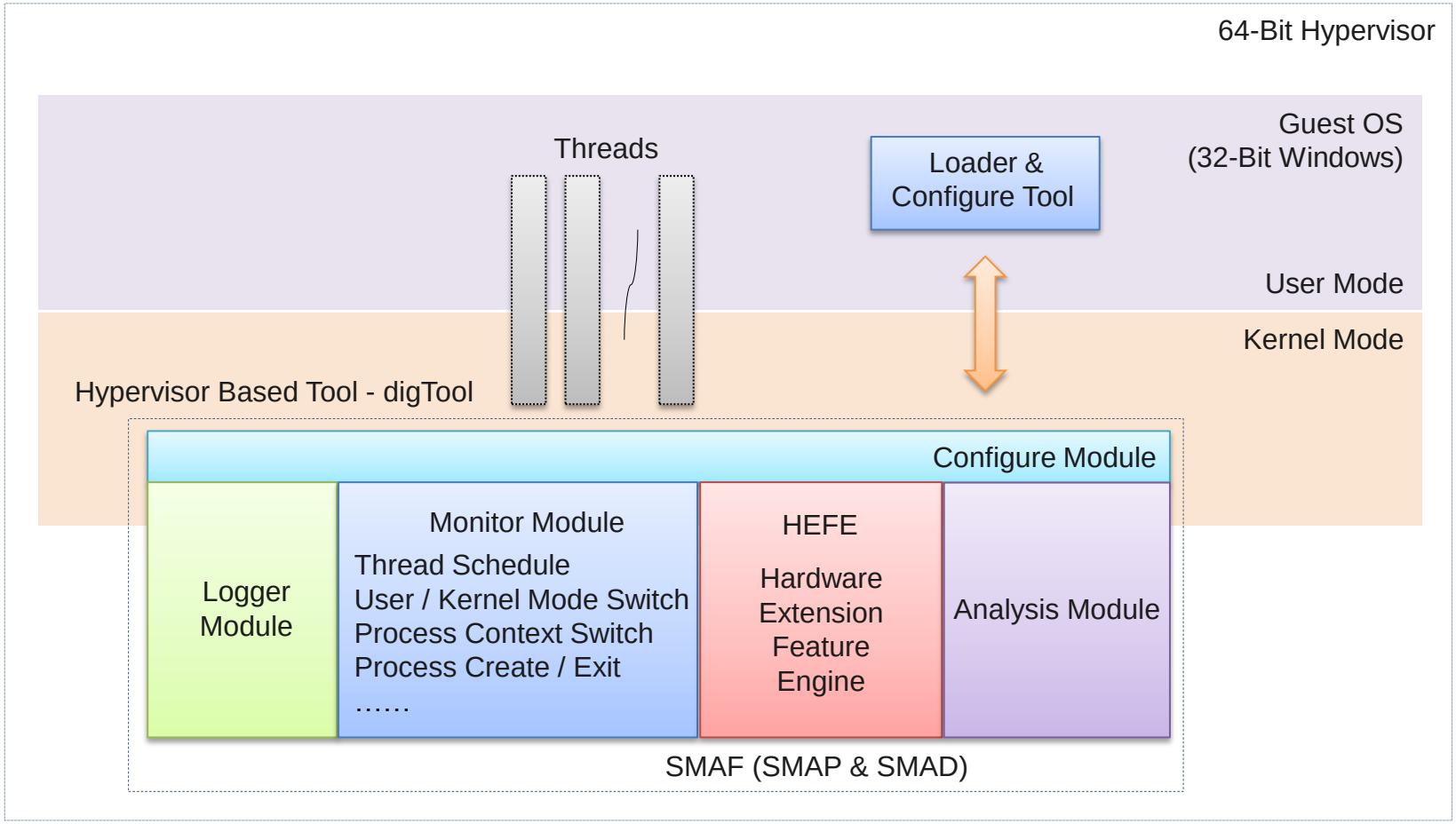

```
Sorry, no more details before Patch Tuesday
--------------------------------------------

     eax=000048e0 esi=8088f000
     ...
     0008:8fa7207c 8906            mov     dword ptr [esi],eax ds:0023:8088f000=????????
```

The Architecture of our Hypervisor Based Tool – digTool

Introduce the Architecture of digTool, PJF, 2014

IceSword, PJF, 2003



Original Log Info

nt!NtCreateFile

```
kd> u 8053E6F4
nt!KiFastCallEntry+0xf4:
8053e6f4 f3a5            rep movs dword ptr es:[edi],dword ptr [esi]
8053e6f6 ffd3            call     ebx
```

1

```
kd> u 8056C09B
nt!IopCreateFile+0x18d:
8056c09b 8911            mov      dword ptr [ecx],edx
```

2

```
Thread: 1152, Eip: 8053E6F4, Address: 0012FEDC, rw: R, TimeStamp:10128810bc0
Thread: 1152, Eip: 8056C09B, Address: 0012FF4C, rw: W, TimeStamp:1012881d340
Thread: 1152, Eip: 8056C0AC, Address: 0012FF24, rw: R, TimeStamp:101288291e0
Thread: 1152, Eip: 8056C0AE, Address: 0012FF24, rw: W, TimeStamp:10128834f80
Thread: 1152, Eip: 8056C0B0, Address: 0012FF28, rw: R, TimeStamp:10128840d30
Thread: 1152, Eip: 8056C0B3, Address: 0012FF28, rw: W, TimeStamp:1012884c950
Thread: 1152, Eip: 805B7D41, Address: 0012FEEC, rw: R, TimeStamp:10128858e90
Thread: 1152, Eip: 805B7D4A, Address: 0012FEF8, rw: R, TimeStamp:10128864c20
Thread: 1152, Eip: 805B7D57, Address: 0012FEF0, rw: R, TimeStamp:101288709a0
Thread: 1152, Eip: 805B7D5D, Address: 0012FEF8, rw: R, TimeStamp:1012887c770
Thread: 1152, Eip: 805B7DFF, Address: 0012FF18, rw: R, TimeStamp:101288c3720
Thread: 1152, Eip: 805B7C19, Address: 0012FF2C, rw: R, TimeStamp:101288cf780
Thread: 1152, Eip: 805B7C1E, Address: 0012FF30, rw: R, TimeStamp:101288db4e0
Thread: 1152, Eip: 805B7CA0, Address: 001520A0, rw: R, TimeStamp:101288e7530
Thread: 1152, Eip: 805B7CA0, Address: 001520A4, rw: R, TimeStamp:101288f3540
```

3

4

```
kd> u 805B7D41
nt!ObpCaptureObjectCreateInformation+0x49:
805b7d41 833818           cmp      dword ptr [eax],18h
```

```
kd> u 805B7C19
nt!ObpCaptureObjectName+0x3f:
805b7c19 8b31             mov      esi,dword ptr [ecx]
```

```
kd> u 805B7CA0
nt!ObpCaptureObjectName+0xc6:
805b7ca0 f3a5             rep movs dword ptr es:[edi],dword ptr [esi]
```

5

Case Study : ObpCaptureObjectCreateInformation

Win-XP / WRK nt!ObpCaptureObjectCreateInformation `ObjectAttributes->Attributes` Double Fetch Detection

```
kd> u 805B7D4A
nt!ObpCaptureObjectCreateInformation+0x52:
805b7d4a f7400c0df8ffff  test    dword ptr [eax+0Ch],0FFFFF80Dh
805b7d51 0f85b0000000    jne     nt!ObpCaptureObjectCreateInformation+0x10f (805b7e07)
805b7d57 8b4804          mov     ecx,dword ptr [eax+4]
805b7d5a 894b04          mov     dword ptr [ebx+4],ecx
```

first fetch

```
================== [ double fetch detected! ] ==================
Eip 1st: 805B7D4A, Address: 0012F95C, rw: R, TimeStamp:10dfa8d5810
Eip 2nd: 805B7D5D, Address: 0012F95C, rw: R, TimeStamp:10dfa8e78f0
```

second fetch

```
kd> u 805B7D5D
nt!ObpCaptureObjectCreateInformation+0x65:
805b7d5d 8b480c          mov     ecx,dword ptr [eax+0Ch]
805b7d60 81e1f2070000    and     ecx,7F2h
805b7d66 890b            mov     dword ptr [ebx],ecx
805b7d68 807d1000        cmp     byte ptr [ebp+10h],0
```

Case Study : ObpCaptureObjectCreateInformation

```
NTSTATUS
ObpCaptureObjectCreateInformation (
    IN POBJECT_TYPE ObjectType OPTIONAL,
    IN KPROCESSOR_MODE ProbeMode,
    IN KPROCESSOR_MODE CreatorMode,
    IN POBJECT_ATTRIBUTES ObjectAttributes,
    IN OUT PUNICODE_STRING CapturedObjectName,
    IN POBJECT_CREATE_INFORMATION ObjectCreateInfo,
    IN LOGICAL UseLookaside
    )
{
    ......
```

first fetch

```
    if (ObjectAttributes->Length != sizeof(OBJECT_ATTRIBUTES) ||
        (ObjectAttributes->Attributes & ~OBJ_ALL_VALID_ATTRIBUTES)) {

        Status = STATUS_INVALID_PARAMETER;

        goto failureExit;
    }

    //
    //  Capture the object attributes.
    //

    ObjectCreateInfo->RootDirectory = ObjectAttributes->RootDirectory;
    ObjectCreateInfo->Attributes = ObjectAttributes->Attributes & OBJ_ALL_VALID_ATTRIBUTES;
```

second fetch

```
    ......
}
```

Summary

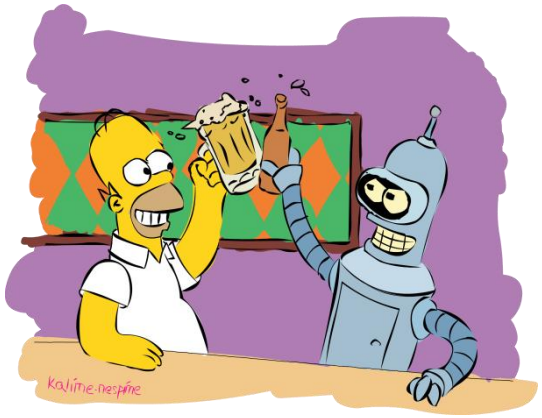
We're not going to lose our job in the short-period of time~

Think :

1) What does it really mean by putting engine into OS kernel

2) The font engine will precompute the size of data structure CJ_3 and CJ_4, is this the right way to do that?

3) .....



Part Five

The End

Think Deeply

Rashomon

"Moving ... the GDI from user mode to kernel mode has provided improved performance without any significant decrease in system stability or reliability" ...

– Is Windows Less Stable with USER and GDI in Kernel Mode?
Windows Internals, Fourth Edition

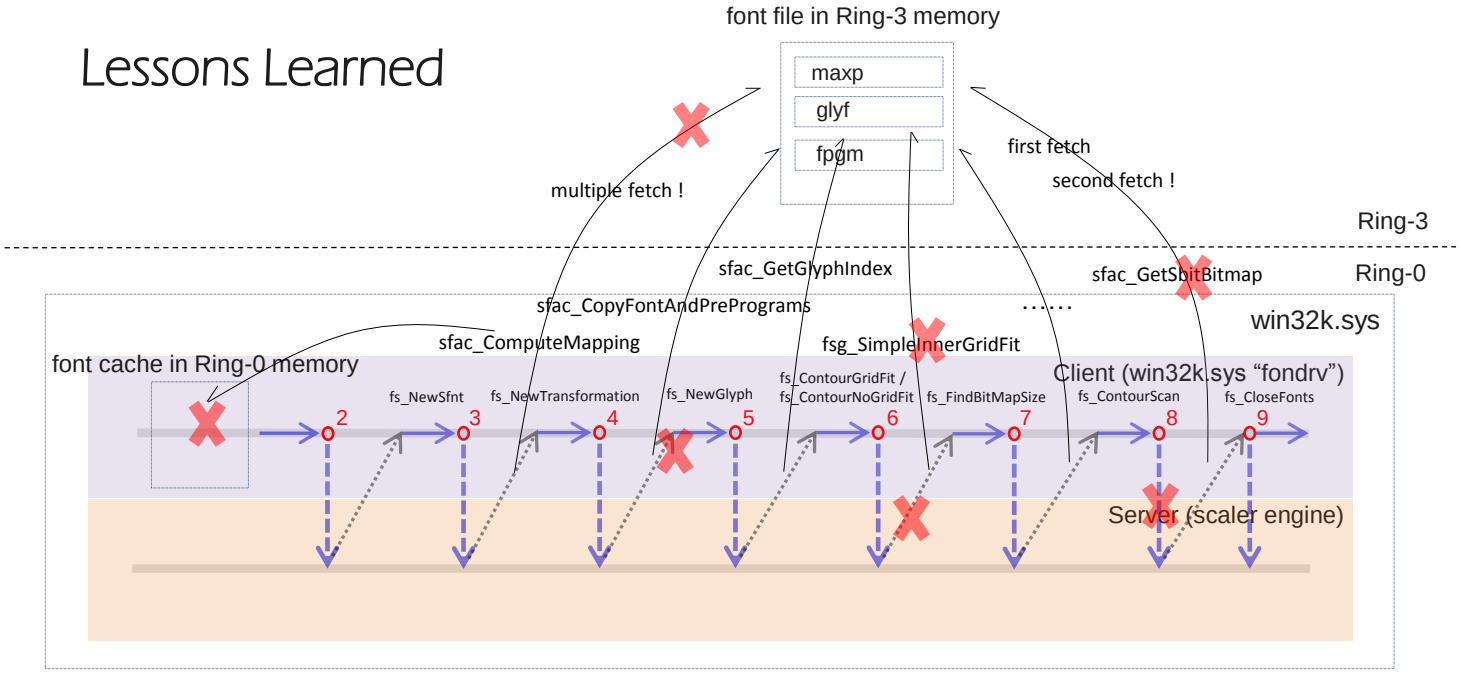
"GDI represents a significant kernel attack surface, and is perhaps the most easily accessible remotely.

This resulted in perhaps our most critical discovery, remote ring0 code execution when a user visits a hostile website (even for unprivileged or protected mode users)" ...

– There's a Party at Ring0 and You're Invited
Tavis Ormandy and Julien Tinnes, BlackHat USA, 2010

Think Deeply

Lessons Learned



All inputs are harmful

But, this is only the tip of the iceberg

Some other grey corners of this matter are not included in this presentation

Before Microsoft fix them, be mindful of your font engine please

Thank You !

Acknowledgements

P1P1Winner

Wu Shi

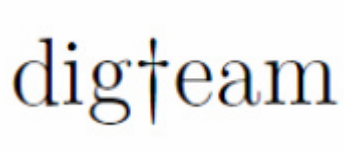
PJF

Liang Chen

Bugvuln

Royce Lu

360Safe



Q&A

wangyu@360.cn