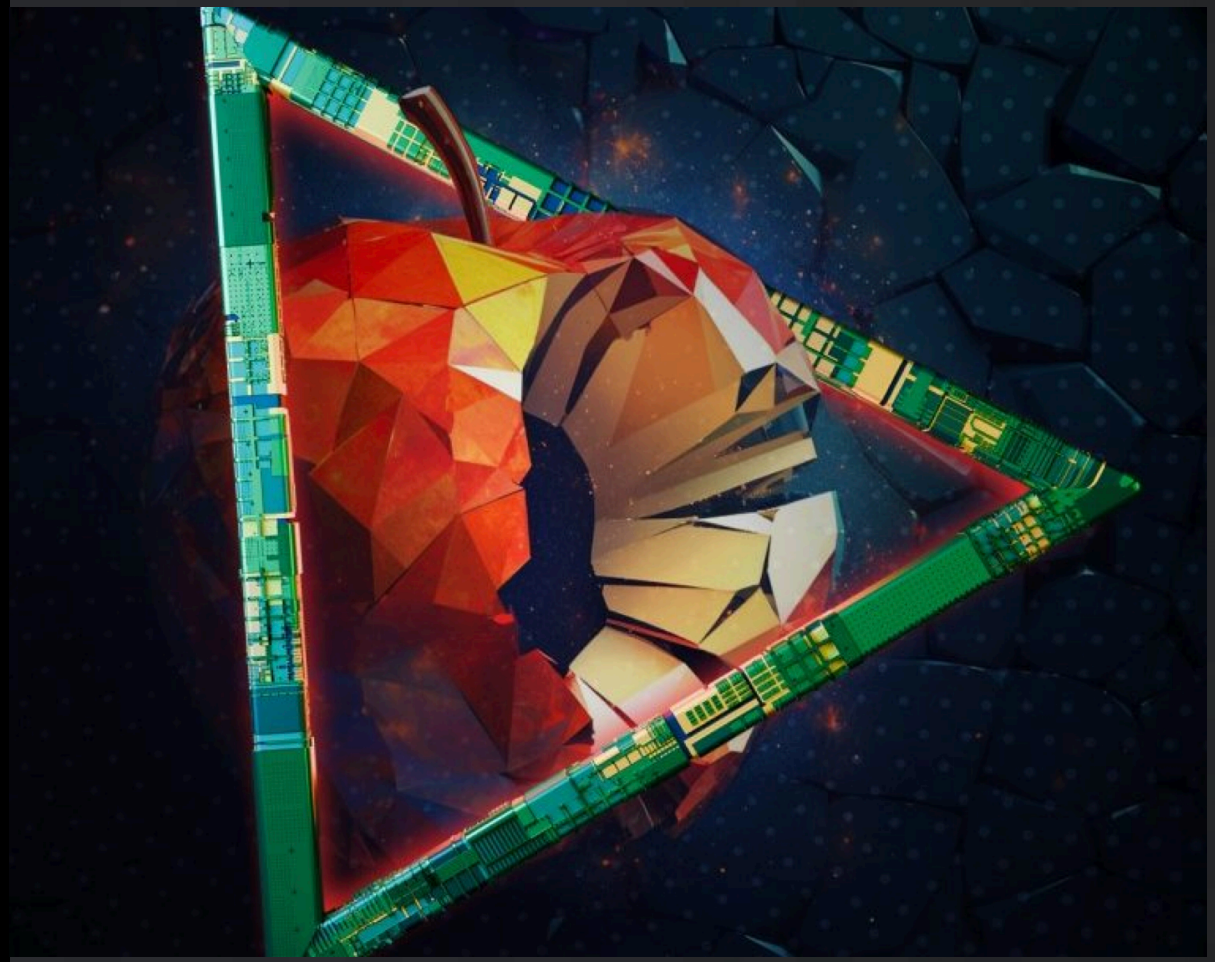
Triangulating TrueType Fonts On macOS

Reconstructing CVE-2023-41990

Aleksandar Nikolić,
Vulnerability Research

cisco | TALOS

Operation Triangulation



Operation Triangulation

The Target

Kaspersky Researchers

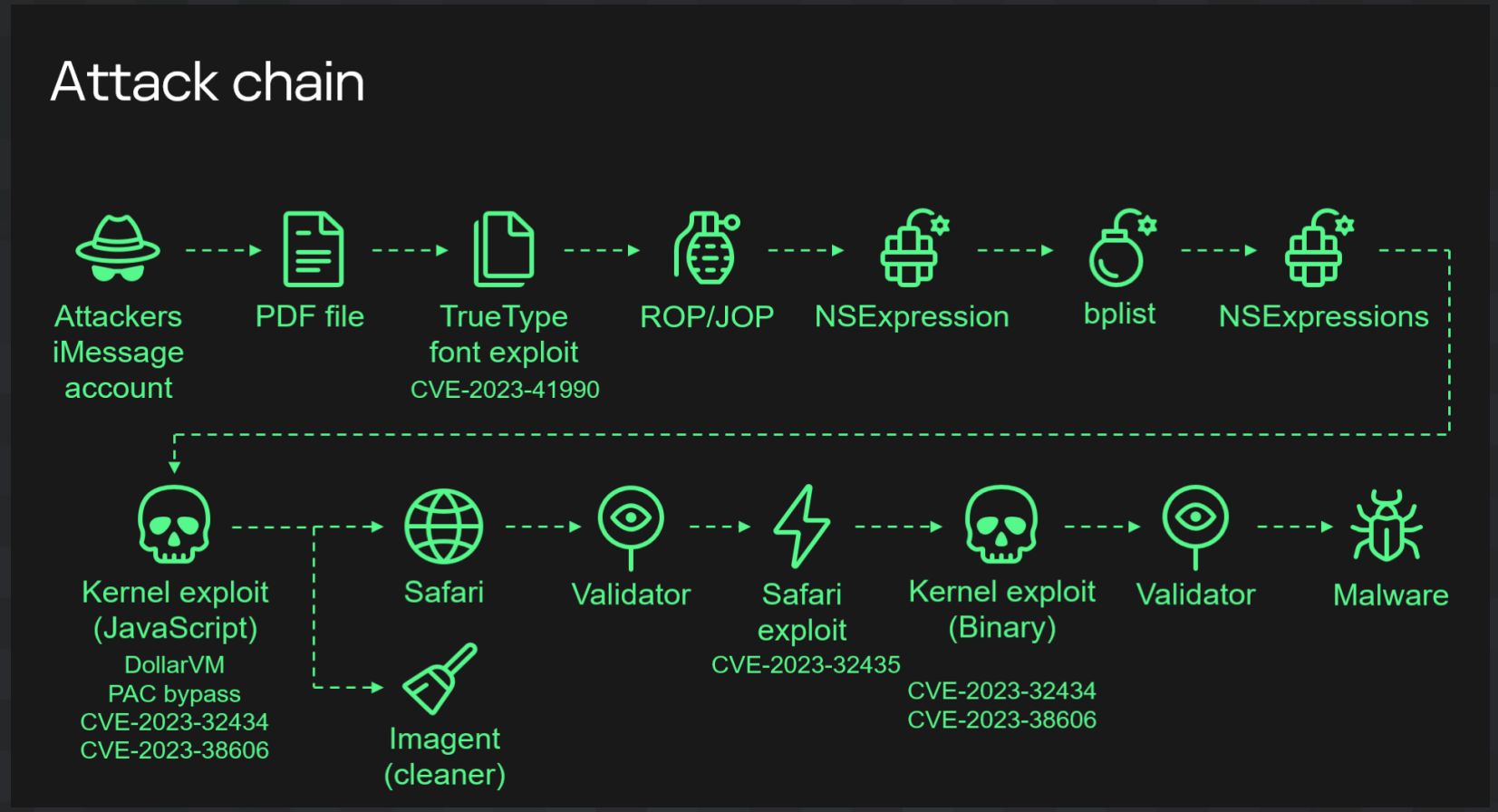
The Discovery

Suspicious egress connections, multiple reinfections, step by step forensics

The Bugs

CVE-2023-32434,
CVE-2023-32435,
CVE-2023-38606,
CVE-2023-41990…

A complex beast

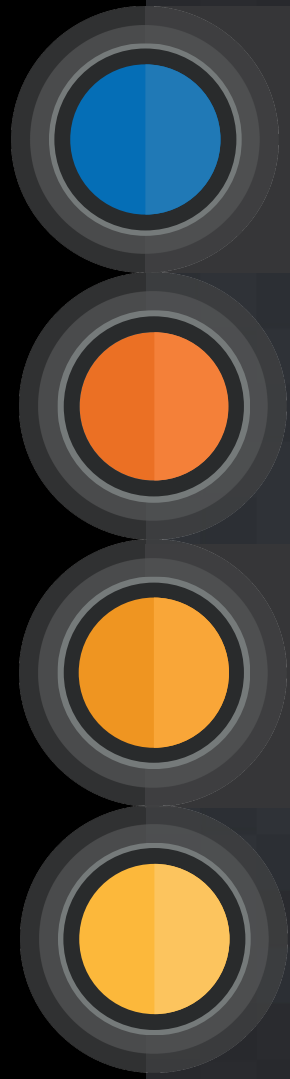


”Operation Triangulation: What You Get When Attack iPhones of Researchers”

https://www.youtube.com/watch?v=1f6YyH62jFE

Zooming in

Who We Are



Talos Vulnerability Discovery and Research

3rd Party Vulnerability Research

Focus on widely deployed software, embedded devices, IOT, ICS

Public vulnerability reports https://www.talosintelligence.com/vulnerability_reports

TALOS-2024-1946	Adobe Acrobat Reader Font gvar GlyphVariationData out-of-bounds read vulnerability	2024-05-15	CVE-2024-30311	6.5
TALOS-2024-1952	Adobe Acrobat Reader Font CPAL numColorRecords out-of-bounds read vulnerability	2024-05-15	CVE-2024-30312	6.5
TALOS-2023-1890	Adobe Acrobat Reader Annot3D object zoom event use-after-free vulnerability	2024-02-15	CVE-2024-20729	8.8
TALOS-2023-1901	Adobe Acrobat Reader FileAttachment PDAnnot destroy use-after-free vulnerability	2024-02-15	CVE-2024-20731	8.8
TALOS-2023-1905	Adobe Acrobat Reader Font CPAL numColorRecords out-of-bounds read vulnerability	2024-02-15	CVE-2024-20735	6.5
TALOS-2023-1906	Adobe Acrobat Reader Font CPAL integer overflow vulnerability	2024-02-15	CVE-2024-20730	8.8
TALOS-2023-1909	Adobe Acrobat Reader Font avar SegmentMaps out-of-bounds read vulnerability	2024-02-15	CVE-2024-20748	6.5
TALOS-2023-1910	Adobe Acrobat Reader Font CharStrings CharStringsOffset out-of-bounds read vulnerability	2024-02-15	CVE-2024-20749	6.5
TALOS-2023-1908	Adobe Acrobat Reader Font CharStrings INDEX out-of-bounds read vulnerability	2024-02-15	CVE-2024-20747	6.5
TALOS-2023-1794	Adobe Acrobat Reader Thermometer use-after-free vulnerability	2023-11-15	CVE-2023-44336	8.8
TALOS-2023-1842	Adobe Acrobat Reader U3D page event use-after-free vulnerability	2023-11-15	CVE-2023-44372	8.8
TALOS-2023-1750	Accusoft ImageGear tiff_planar_adobe out-of-bounds write vulnerability	2023-09-25	CVE-2023-32284	8.1
TALOS-2022-1516	Adobe Acrobat Reader DC overlapping annotations type confusion vulnerability	2022-07-13	CVE-2022-34221	8.8
TALOS-2022-1525	Adobe Acrobat Reader DC event value use-after-free	2022-07-13	CVE-2022-34230	8.8
TALOS-2021-1387	Adobe Acrobat Reader Javascript event.richValue use-after-free vulnerability	2022-01-11	CVE-2021-44710	8.8
TALOS-2021-1410	Adobe Acrobat Reader DC annotation gestures integer overflow vulnerability	2022-01-11	CVE-2021-44711	8.8
TALOS-2021-1233	Adobe Acrobat Reader DC JavaScript search query code execution vulnerability	2021-05-11	CVE-2021-28562	8.0
TALOS-2020-1156	Adobe Acrobat Reader DC form field format use after free	2020-11-05	CVE-2020-24437	8.8
TALOS-2020-1157	Adobe Acrobat Reader DC JavaScript submitForm heap buffer overflow redux	2020-11-05	CVE-2020-24435	8.8
TALOS-2020-1028	Adobe Acrobat Reader DC Annotation Destroy Remote Code Execution	2020-05-12	CVE-2020-9607	8.8
TALOS-2020-1031	Adobe Acrobat Reader DC Javascript submitForm Remote Code Execution Vulnerability	2020-05-12	CVE-2020-9609	8.8

Vulnerability Spotlight: Vulnerabilities in popular Japanese word processing software could lead to arbitrary code execution, other issues

Why do we care?

Spelunking through ancient code in search of vulnerabilities

Similar vulnerabilities

FORCEDENTRY

Complex weird machine built around JBIG2 vulnerability

CVE-2021-30860

BLASTPASS

An intricate heap overflow vulnerability in WebP

CVE-2023-4863

Unnamed SBX Escape

Ian Beer's last year presentation, ridiculous NSExpression based exploitation framework

CVE-2023-32409

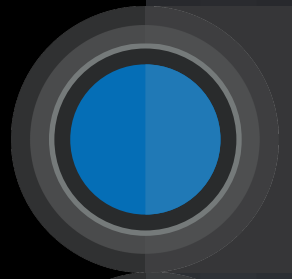
Detection / Scanning

- Matt Suiche's ELEGANT BOUNCER
- https://github.com/msuiche/elegant-bouncer

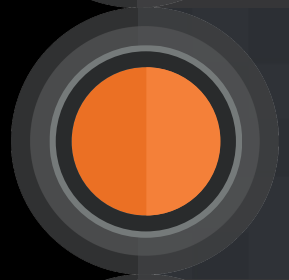
Support Table

Threat Name	CVEs	Supported
FORCEDENTRY	CVE-2021-30860	✅
BLASTDOOR	CVE-2023-4863, CVE-2023-41064	✅
TRIANGULATION	CVE-2023-41990	✅

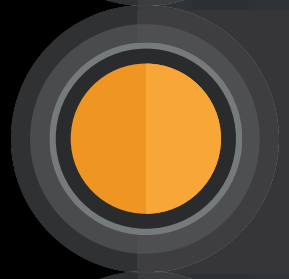
High impact vulnerabilities



Important attack surface



Variant analysis



Catching samples

Understanding exploitation

What we know?

Nobody would share an actual sample for some reason

CVE-2023-41990

1. Malicious TrueType Font
2. Delivered inside a PDF
3. Undocumented instruction
4. Vulnerability in bytecode interpreter

Bytecode interpreter?

A bit of history first

Origins of modern fonts

- Bitmap fonts
  - Not scalable
- PostScript Fonts
  - Adobe in 1980s
  - Outline fonts, verbose, multiple files
- TrueType Fonts
  - Apple in early 1990s
  - Outline fonts, hinted
- OpenType / CFF
  - Adobe & Microsoft in 1997
  - Outline fonts, binary, compact

A history of vulnerabilities

- Complex, but performance crucial
- Windows used to do it in the kernel!
- Charstrings/bytecode beautiful for exploitation
- Reachable from everywhere

One font vulnerability to rule them all

A story of cross-software ownage, shared codebases and advanced exploitation.

Mateusz “j00ru” Jurczyk

REcon 2015, Montreal

News and updates from the Project Zero team at Google

Friday, July 1, 2016

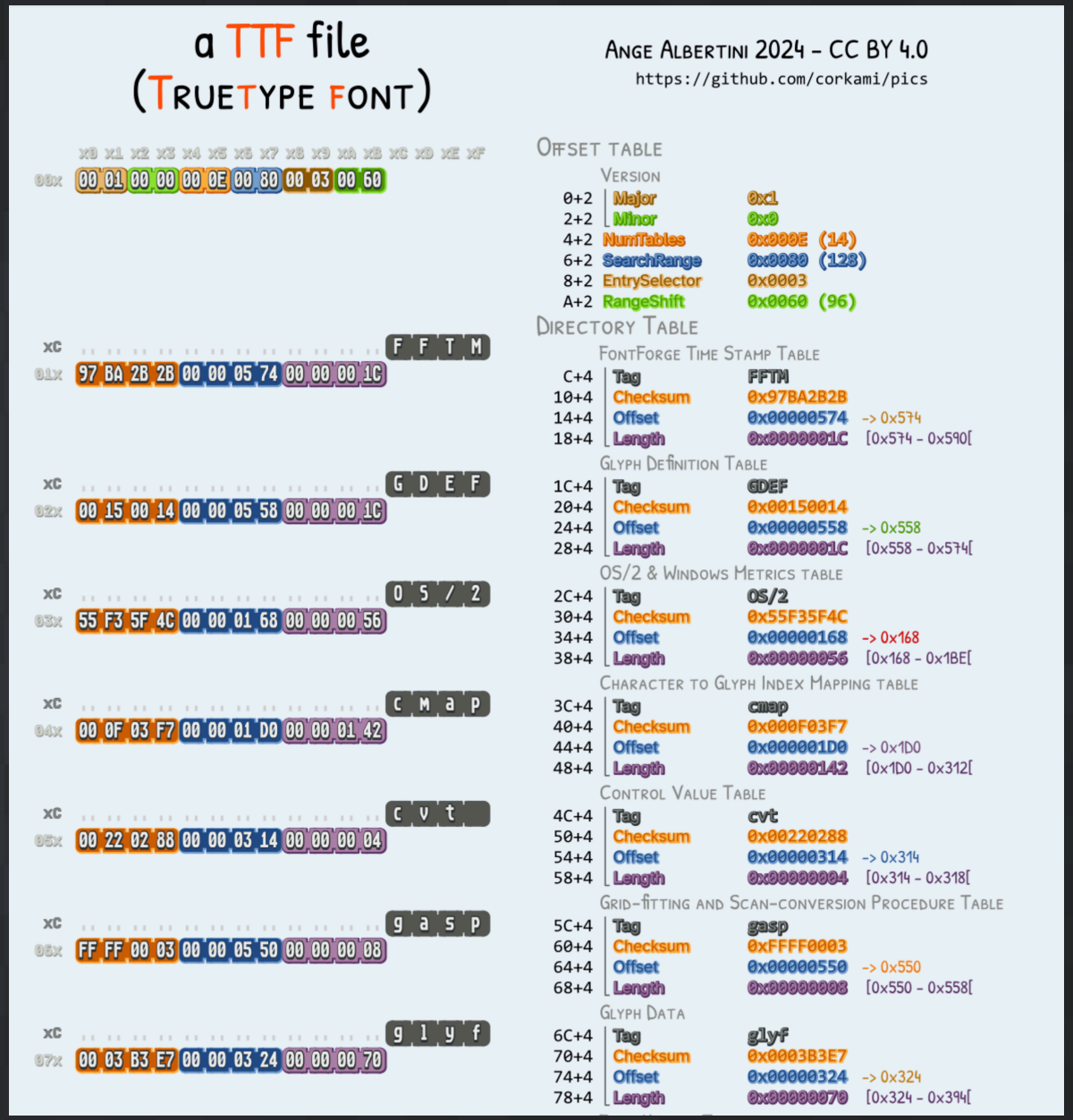
A year of Windows kernel font fuzzing #2: the techniques

Posted by Mateusz Jurczyk of Google Project Zero

Font File Format Families

- TrueTypeFonts
- OpenTypeFonts
- Ancient bitmap fonts
- WOFF/WOFF2

- It’s all SFNT format – different rendering



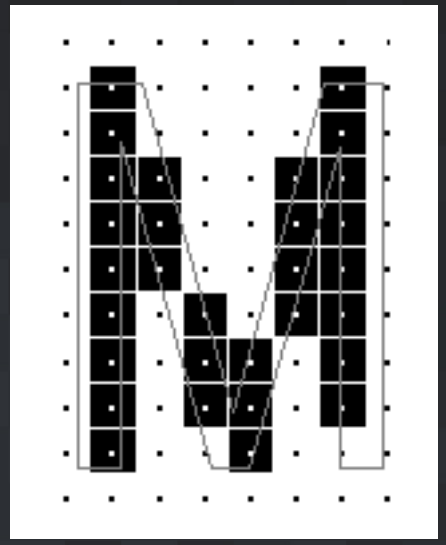
Sidenote

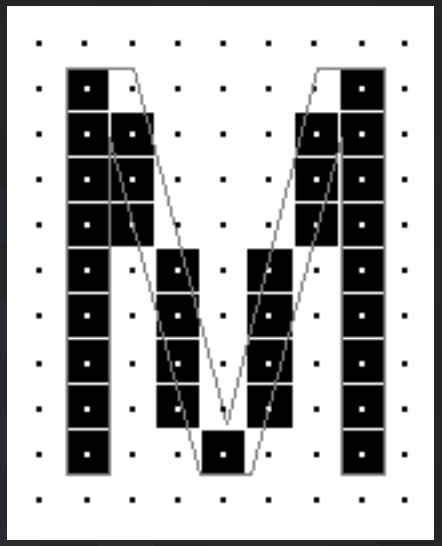
Fuzzing SFNT

- A table for everything
- Cross references with indices and file offsets
- Single offset change makes whole file unusable
- Dumb fuzzing extremely ineffective

What was that about an interpreter?

Hinted Fonts / Font Instructing





PACK MY BOX W
THE LAZY DOG V
WYVERN FOXY S
pack my box with
dog effect coffin
wyvern foxy syry
Of the greatest ar
look with too muc
the human mind a

PACK MY BOX W
THE LAZY DOG V
WYVERN FOXY S
pack my box wit
dog effect coffin
wyvern foxy syry
Of the greatest a
look with too mu
the human mind

TrueTypeFont Interpreter

- A simple stack machine
  - Push a few bytes
  - Call function that consumes them
- Dozens of instructions
- Manipulates glyph data, points, curves, lines
- Manipulates pointers, indices and arrays

Apple Developed

```
fnt_RoundUpToGrid(int32_t, fnt_LocalGraphicStateType*)
fnt_RoundToHalfGrid(int32_t, fnt_LocalGraphicStateType*)
fnt_RoundToGrid(int32_t, fnt_LocalGraphicStateType*)
fnt_RoundToDoubleGrid(int32_t, fnt_LocalGraphicStateType*)
fnt_RoundOff(int32_t, fnt_LocalGraphicStateType*)
fnt_RoundDownToGrid(int32_t, fnt_LocalGraphicStateType*)
fnt_RS(fnt_LocalGraphicStateType*)
fnt_ROUND(fnt_LocalGraphicStateType*)
fnt_ROLL(fnt_LocalGraphicStateType*)
fnt_RCVT(fnt_LocalGraphicStateType*)
fnt_RAW(fnt_LocalGraphicStateType*)
fnt_PushSomeWords(fnt_LocalGraphicStateType*, int32_t)
fnt_PushSomeBytes(fnt_LocalGraphicStateType*, int32_t)
fnt_ProjectIntegerPPEM(fnt_LocalGraphicStateType*)
fnt_Project(fnt_LocalGraphicStateType*, int32_t, int32_t)
fnt_PUSHW0(fnt_LocalGraphicStateType*)
fnt_PUSHW(fnt_LocalGraphicStateType*)
fnt_PUSHB0(fnt_LocalGraphicStateType*)
fnt_PUSHB(fnt_LocalGraphicStateType*)
fnt_POP(fnt_LocalGraphicStateType*)
fnt_OldProject(fnt_LocalGraphicStateType*, int32_t, int32_t)
fnt_Normalize(fnt_LocalGraphicStateType*, int32_t, int32_t, shortVector*)
fnt_NilFunction(fnt_LocalGraphicStateType*, int32_t)
fnt_NROUND(fnt_LocalGraphicStateType*)
fnt_NPUSHW(fnt_LocalGraphicStateType*)
fnt_NPUSHB(fnt_LocalGraphicStateType*)
fnt_MovePoint(fnt_LocalGraphicStateType*, fnt_ElementType*, int32_t, int32
fnt_MSIRP(fnt_LocalGraphicStateType*)
fnt_MPS(fnt_LocalGraphicStateType*)
fnt_MPPEM(fnt_LocalGraphicStateType*)
fnt_MIRP(fnt_LocalGraphicStateType*)
fnt_MINDEX(fnt_LocalGraphicStateType*)
fnt_MIAP(fnt_LocalGraphicStateType*)
fnt_MDRP(fnt_LocalGraphicStateType*)
fnt_MDAP(fnt_LocalGraphicStateType*)
fnt_MD(fnt_LocalGraphicStateType*)
fnt_LOOPCALL(fnt_LocalGraphicStateType*)
fnt_JROT(fnt_LocalGraphicStateType*)
fnt_JROF(fnt_LocalGraphicStateType*)
fnt_JMPR(fnt_LocalGraphicStateType*)
fnt_InnerExecute(fnt_LocalGraphicStateType*, uint8_t const*, uint8_t const
fnt_IllegalInstruction
fnt_IUP(fnt_LocalGraphicStateType*)
fnt_ISECT(fnt_LocalGraphicStateType*)
fnt_IP(fnt_LocalGraphicStateType*)
fnt_INSTCTRL(fnt_LocalGraphicStateType*)
```

```
fnt_IDEF(fnt_LocalGraphicStateType*)
fnt_GetSingleWidthSlow(fnt_LocalGraphicStateType*)
fnt_GetSingleWidthFast(fnt_LocalGraphicStateType*)
fnt_GetCVTScale(fnt_LocalGraphicStateType*)
fnt_GetCVTEntrySlow(fnt_LocalGraphicStateType*, int32_t)
fnt_GetCVTEntryFast(fnt_LocalGraphicStateType*, int32_t)
fnt_GPV(fnt_LocalGraphicStateType*)
fnt_GFV(fnt_LocalGraphicStateType*)
fnt_GETVARIATION(fnt_LocalGraphicStateType*)
fnt_GETINFO(fnt_LocalGraphicStateType*)
fnt_GETDATA(fnt_LocalGraphicStateType*)
fnt_GC(fnt_LocalGraphicStateType*)
fnt_FindENDF(fnt_LocalGraphicStateType*, uint8_t const*)
fnt_FLIPRGON(fnt_LocalGraphicStateType*)
fnt_FLIPRGOFF(fnt_LocalGraphicStateType*)
fnt_FLIPPT(fnt_LocalGraphicStateType*)
fnt_FLIPON(fnt_LocalGraphicStateType*)
fnt_FLIPOFF(fnt_LocalGraphicStateType*)
fnt_FDEF(fnt_LocalGraphicStateType*)
fnt_Execute(fnt_ElementType**, fnt_GlobalGraphicStateType*, ui
fnt_ELSE(fnt_LocalGraphicStateType*)
fnt_EIF(fnt_LocalGraphicStateType*)
fnt_DeltaEngine(fnt_LocalGraphicStateType*, void (*)(fnt_Local
fnt_DUP(fnt_LocalGraphicStateType*)
fnt_DEPTH(fnt_LocalGraphicStateType*)
fnt_DELTAP3(fnt_LocalGraphicStateType*)
fnt_DELTAP2(fnt_LocalGraphicStateType*)
fnt_DELTAP1(fnt_LocalGraphicStateType*)
fnt_DELTAC3(fnt_LocalGraphicStateType*)
fnt_DELTAC2(fnt_LocalGraphicStateType*)
fnt_DELTAC1(fnt_LocalGraphicStateType*)
fnt_DEBUG(fnt_LocalGraphicStateType*)
fnt_ComputeAndCheck_PF_Proj(fnt_LocalGraphicStateType*)
fnt_Check_PF_Proj(fnt_LocalGraphicStateType*)
fnt_CheckSingleWidth(int32_t, fnt_LocalGraphicStateType*)
fnt_ChangeCvt(fnt_LocalGraphicStateType*, fnt_ElementType*, ir
fnt_CLEAR(fnt_LocalGraphicStateType*)
fnt_CINDEX(fnt_LocalGraphicStateType*)
fnt_CALL_Common(fnt_LocalGraphicStateType*, fnt_funcDef*, int3
fnt_CALL(fnt_LocalGraphicStateType*)
fnt_BinaryOperand(fnt_LocalGraphicStateType*)
```

```
fnt_IDEF(fnt_LocalGraphicStateType*)
fnt_GetSingleWidthSlow(fnt_LocalGraphicStateType*)
fnt_GetSingleWidthFast(fnt_LocalGraphicStateType*)
fnt_GetCVTScale(fnt_LocalGraphicStateType*)
fnt_GetCVTEntrySlow(fnt_LocalGraphicStateType*, int
fnt_GetCVTEntryFast(fnt_LocalGraphicStateType*, int
fnt_GPV(fnt_LocalGraphicStateType*)
fnt_GFV(fnt_LocalGraphicStateType*)
fnt_GETVARIATION(fnt_LocalGraphicStateType*)
fnt_GETINFO(fnt_LocalGraphicStateType*)
fnt_GETDATA(fnt_LocalGraphicStateType*)
fnt_GC(fnt_LocalGraphicStateType*)
fnt_FindENDF(fnt_LocalGraphicStateType*, uint8_t co
fnt_FLIPRGON(fnt_LocalGraphicStateType*)
fnt_FLIPRGOFF(fnt_LocalGraphicStateType*)
fnt_FLIPPT(fnt_LocalGraphicStateType*)
fnt_FLIPON(fnt_LocalGraphicStateType*)
fnt_FLIPOFF(fnt_LocalGraphicStateType*)
fnt_FDEF(fnt_LocalGraphicStateType*)
fnt_Execute(fnt_ElementType**, fnt_GlobalGraphicSta
fnt_ELSE(fnt_LocalGraphicStateType*)
fnt_EIF(fnt_LocalGraphicStateType*)
fnt_DeltaEngine(fnt_LocalGraphicStateType*, void (*
fnt_DUP(fnt_LocalGraphicStateType*)
fnt_DEPTH(fnt_LocalGraphicStateType*)
fnt_DELTAP3(fnt_LocalGraphicStateType*)
fnt_DELTAP2(fnt_LocalGraphicStateType*)
fnt_DELTAP1(fnt_LocalGraphicStateType*)
fnt_DELTAC3(fnt_LocalGraphicStateType*)
fnt_DELTAC2(fnt_LocalGraphicStateType*)
fnt_DELTAC1(fnt_LocalGraphicStateType*)
fnt_DEBUG(fnt_LocalGraphicStateType*)
fnt_ComputeAndCheck_PF_Proj(fnt_LocalGraphicStateTy
fnt_Check_PF_Proj(fnt_LocalGraphicStateType*)
fnt_CheckSingleWidth(int32_t, fnt_LocalGraphicState
fnt_ChangeCvt(fnt_LocalGraphicStateType*, fnt_Eleme
fnt_CLEAR(fnt_LocalGraphicStateType*)
fnt_CINDEX(fnt_LocalGraphicStateType*)
```

Microsoft borrowed

WinNT4 / private / ntos / w32 / ntgdi / fondrv / tt / scaler / **interp.c**

Code | Blame | Executable File · 5502 lines (4648 loc) · 136

```
175  FS_PRIVATE F26Dot6 itrp_CheckSingleWidth (GSP F26Dot6
176  FS_PRIVATE fnt_instrDef*itrp_FindIDef (GSP uint8 opCo
177  FS_PRIVATE void itrp_DeltaEngine (GSP FntMoveFunc doI
178
179  /* Actual instructions for the jump table */
180  FS_PRIVATE uint8* itrp_SVTCA_0 (IPARAM);
181  FS_PRIVATE uint8* itrp_SVTCA_1 (IPARAM);
182  FS_PRIVATE uint8* itrp_SPVTCA_0 (IPARAM);
183  FS_PRIVATE uint8* itrp_SPVTCA_1 (IPARAM);
184  FS_PRIVATE uint8* itrp_SFVTCA_0 (IPARAM);
185  FS_PRIVATE uint8* itrp_SFVTCA_1 (IPARAM);
186  FS_PRIVATE uint8* itrp_SPVTL (IPARAM);
187  FS_PRIVATE uint8* itrp_SDPVTL (IPARAM);
188  FS_PRIVATE uint8* itrp_SFVTL (IPARAM);
189  FS_PRIVATE uint8* itrp_WPV (IPARAM);
190  FS_PRIVATE uint8* itrp_WFV (IPARAM);
191  FS_PRIVATE uint8* itrp_RPV (IPARAM);
192  FS_PRIVATE uint8* itrp_RFV (IPARAM);
193  FS_PRIVATE uint8* itrp_SFVTPV (IPARAM);
```

Nintendo borrowed

supermario / base / SuperMarioProj.1994-02-09 / Toolbox / FontMgr / fnt.c

Code | Blame | 4038 lines (3572 loc) · 109 KB | Code 55% faster

```
534  void fnt_DefaultJumpTable( voidFunc* function );
535
536  /* Actual instructions for the jump table */
537  void fnt_SVTCA_0(fnt_LocalGraphicStateType *gs);
538  void fnt_SVTCA_1(fnt_LocalGraphicStateType *gs);
539  void fnt_SPVTCA(fnt_LocalGraphicStateType *gs);
540  void fnt_SFVTCA(fnt_LocalGraphicStateType *gs);
541  void fnt_SPVTL(fnt_LocalGraphicStateType *gs);
542  void fnt_SDPVTL(fnt_LocalGraphicStateType *gs);
543  void fnt_SFVTL(fnt_LocalGraphicStateType *gs);
544  void fnt_SPVFS(fnt_LocalGraphicStateType *gs);
545  void fnt_SFVFS(fnt_LocalGraphicStateType *gs);
546  void fnt_GPV(fnt_LocalGraphicStateType *gs);
547  void fnt_GFV(fnt_LocalGraphicStateType *gs);
548  void fnt_SFVTPV(fnt_LocalGraphicStateType *gs);
```

Back to the vulnerability

An undocumented instruction

```
ADJUST is documented-ish - it is both 8F and 0x90 .

/* This code adjusts strokes so that their width is closer to the target width.
 * Only 1 bit is added or subtracted.
 * If the boolean part of the opCode is set, the edge that is moved is specified
 * the first argument; otherwise the edge that needs to move the least is moved.
 */
```

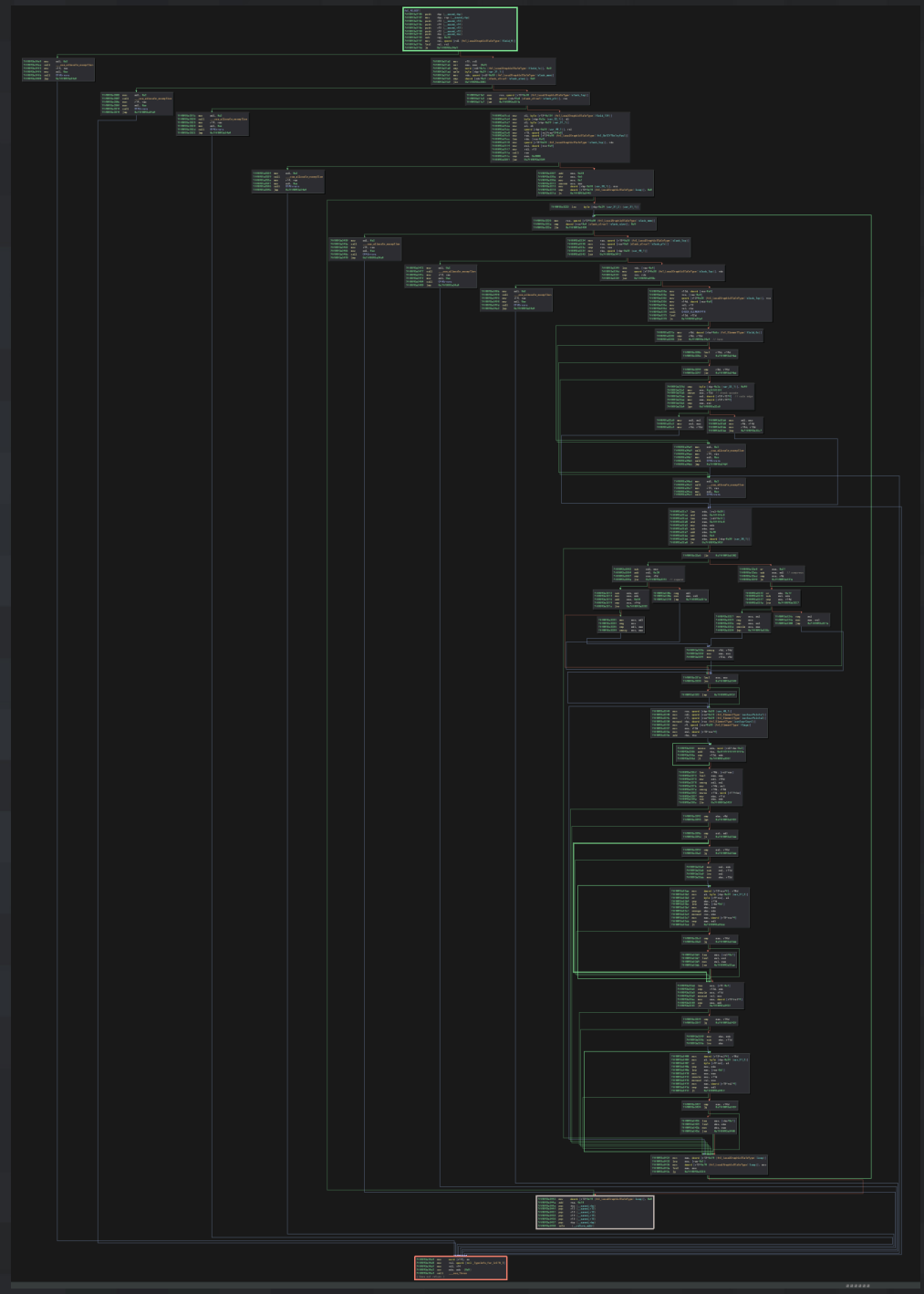
Rare mention online, from FreeType-Dev mailing list

Not really hidden

```
uint32_t fnt_ADJUST(struct fnt_LocalGraphicStateType* arg1_1)

    struct fnt_ElementType* field_0 = arg1_1->field_0
    int16_t rax_12
    int16_t* thrown_exception

    if (field_0 == 0)
        thrown_exception = ___cxa_allocate_exception(2, field_0)
        rax_12 = OFAErrors(0xe)
    else
        bool var_31_1 = arg1_1->field_1c == 0
        struct stack_struct* stack_mem_1 = arg1_1->stack_mem

        if (stack_mem_1->stack_size s<= 0)
            thrown_exception = ___cxa_allocate_exception(2, field_0, sta
            rax_12 = OFAErrors(0xe)
        else
            void* stack_top_1 = arg1_1->stack_top

            if (stack_mem_1->stack_ptr u>= stack_top_1)
                thrown_exception = ___cxa_allocate_exception(2)
                rax_12 = OFAErrors(0xe)
            else
```

Looks complex

Mind the loops



Sidenote

Patch diffing

- N-day root cause analysis
- No such luck this time
- Functionality removed entirely

Finding a testcase

Starting point

- An actual font that uses this instruction?
- Gather ALL the fonts
- Disassembly via Python FontTools
  - Unknown instructions decoded as INST<OpCode>
- No suitable sample :/

Manual reconstruction

Starting with a basic font

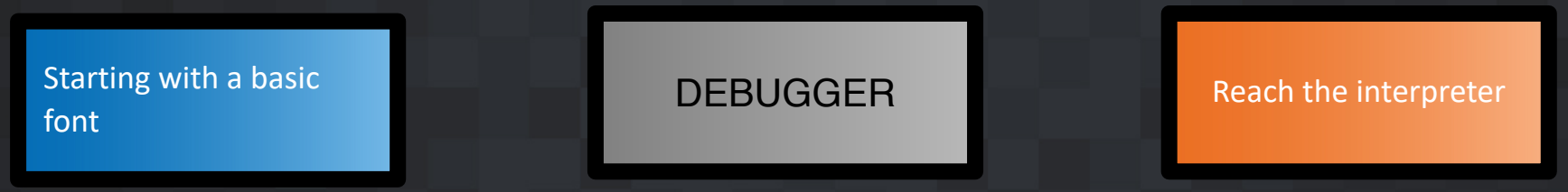
DEBUGGER

Reach the interpreter

Add ADJUST instructions
Set breakpoints

```
int64_t fnt_Execute(struct fnt_ElementType** arg1, struct stack_struct* arg2, char* arg3,
  int64_t arg4, int64_t arg5, int64_t* arg6, char arg7)
```

Manual reconstruction




```
(lldb) target create "./pdfbarf.exe"
Current executable set to './pdfbarf.exe' (x86_64).
(lldb) settings set -- target.run-args  "-f" "testing.ttf..pdf"
(lldb) breakpoint set -n fnt_Execute
Breakpoint 1: where = libFontParser.dylib`fnt_Execute, address = 0x00007ff808f8a7a7
(lldb) r
Process 26941 launched: './pdfbarf.exe' (x86_64)
Process 26941 exited with status = 0 (0x00000000)
(lldb)
```

Manual reconstruction

Add ADJUST instructions
Set breakpoints

```
int64_t fnt_Execute(struct fnt_ElementType** arg1, struct stack_struct* arg2, char* arg3,
  int64_t arg4, int64_t arg5, int64_t* arg6, char arg7)
```

Sidenote

AutoHinting

- LowDPI screens no more
- Retina very much HighDPI
- MS ClearType
- Avoid interpreter completely

- Online sources: “macOS ignores hinting”

Not entirely ignored

Reverse engineering required

```
7ff809417d43    int64_t RunGlyphProgram(int64_t** arg1, struct fnt_ElementType* arg2, void* arg3,
7ff809417d43      int32_t arg4, char* arg5, char arg6)

7ff809417d5b        int64_t rax = *___stack_chk_guard
7ff809417d5b
7ff809417d64        if (arg4 != 0)
7ff809417d70            void* r14_1 = arg1[3]
7ff809417d7f            *(r14_1 + 0x150) = 3
7ff809417da8            SetGlobalGSMapping(r14_1 + 0xd0, arg3 + 0x24, sx.d(*(arg1[1] + 0x82)))
7ff809417db4            *(r14_1 + 0x1c0) = *(r14_1 + 0x188)
7ff809417dbb            int128_t zmm0_1 = *(r14_1 + 0x158)
7ff809417dc3            int128_t zmm1_1 = *(r14_1 + 0x168)
7ff809417dd3            *(r14_1 + 0x1b0) = *(r14_1 + 0x178)
7ff809417ddb            *(r14_1 + 0x1a0) = zmm1_1
7ff809417de3            *(r14_1 + 0x190) = zmm0_1
7ff809417df7            struct fnt_ElementType* var_48 = &arg1[5][7]
7ff809417dfa            struct fnt_ElementType* var_40_1 = arg2
7ff809417e2e            uint64_t var_68_1 = zx.q(arg6)
7ff809417e2f            uint64_t var_70_1 = zx.q(*(arg1[1] + 0x9d))
7ff809417e32            fnt_Execute(&var_48, r14_1 + 0xd0, arg5, &arg5[zx.q(arg4.w)], 0, *arg1, arg
7ff809417e32
```

```
uint64_t StretchGlyph(int64_t** arg1, struct fnt_ElementType* arg2, int16_t* arg3, uint64

7ff8093bc14b                zmm4_2[0] = zmm4_2[0] f* 64f
7ff8093bc152                zmm3_2 = _mm_or_ps(_mm_and_ps(zmm3_2, zmm4_2), 0x3effffff)
7ff8093bc155                zmm3_2[0] = zmm3_2[0] f+ zmm4_2[0]
7ff8093bc166                RoundPhantomPoints(arg2, rsi_9, int.d(_mm_round_ss(0, zmm3_2
7ff8093bc166
7ff8093bc173                if (s != 0)
7ff8093bc179                    CopyHintedOutlineToUnhintedOutline(arg2)
7ff8093bc19a                    RunGlyphProgram(rbx, arg2, arg3, zx.d(s), var_170, 0)
7ff8093bc19a
```

```
▾ RunFontProgram
    |← 7ff809417bdd fnt_Execute(&v
▾ RunPreProgram
    |← 7ff809417ce3 fnt_Execute(&v
▾ RunGlyphProgram
    |← 7ff809417e32 fnt_Execute(&v
```

Special requirements

struct thead head	
TT_Fixed version	65536
TT_Fixed fontRevision	66847
ULONG checkSumAdjustment	2883570723
ULONG magicNumber	1594834165
USHORT flags	9
USHORT unitsPerEm	1024
LONGDATETIME created	
LONGDATETIME modified	
SHORT xMin	-175
SHORT yMin	-309
SHORT xMax	9938
SHORT yMax	977
USHORT macStyle	0
USHORT lowestRecPPEM	19
SHORT fontDirectionHint	2
SHORT indexToLocFormat	0
SHORT glyphDataFormat	0

High Units Per Em (>1024)

High Lowest Recommended Pixels Per Em (>19)

And we are good to go

```
* frame #0: 0x000000019afee494 libFontParser.dylib`fnt_FDEF(fnt_LocalGraphicStateType*)
  frame #1: 0x000000019afc2b30 libFontParser.dylib`fnt_InnerExecute(fnt_LocalGraphicStateType*, unsigned char const*, unsigned char
  frame #2: 0x000000019afc2e2c libFontParser.dylib`fnt_Execute(fnt_ElementType**, fnt_GlobalGraphicStateType*, unsigned char const*,
  frame #3: 0x000000019aff3f34 libFontParser.dylib`RunFontProgram(fsg_SplineKey*, void (*)()) + 108
  frame #4: 0x000000019af79130 libFontParser.dylib`CreateGlyphElement(fsg_SplineKey*, int, unsigned char, unsigned char) + 156
  frame #5: 0x000000019af7936c libFontParser.dylib`CreateScalerGlyphBlock(fsg_SplineKey*, memoryContext*, scalerGlyph const*) + 180
  frame #6: 0x000000019af7703c libFontParser.dylib`AssureGlyphBlock(fsg_SplineKey*, memoryContext*, scalerGlyph*) + 172
  frame #7: 0x000000019af6bfdc libFontParser.dylib`TTRenderGlyphs + 536
  frame #8: 0x000000019af68034 libFontParser.dylib`TConcreteFontScaler::CopyGlyphPath(unsigned short, CGAffineTransform const*) cons
  frame #9: 0x000000019afdf388 libFontParser.dylib`TFPFont::CopyGlyphPath(unsigned int) const + 328
  frame #10: 0x000000019af49b74 libFontParser.dylib`FPFontCopyGlyphPath + 40
  frame #11: 0x000000019690e9d4 CoreGraphics`CGFontCreateGlyphPath + 48
  frame #12: 0x000000019690e7cc CoreGraphics`CGFontCreateGlyphBitmapWithDilation + 584
  frame #13: 0x000000019690e4f0 CoreGraphics`CGGlyphBuilder::create_missing_bitmaps(CGGlyphIdentifier const*, unsigned long, CGGlyph
  frame #14: 0x000000019690e074 CoreGraphics`CGGlyphBuilder::lock_glyph_bitmaps(CGGlyphIdentifier const*, unsigned long, CGGlyphBitm
  frame #15: 0x000000019690de60 CoreGraphics`render_glyphs + 216
  frame #16: 0x000000019690d6c8 CoreGraphics`draw_glyph_bitmaps + 1320
  frame #17: 0x000000019690d0f4 CoreGraphics`ripc_DrawGlyphs + 1268
  frame #18: 0x000000019690ca70 CoreGraphics`CGContextDelegateDrawGlyphs + 368
  frame #19: 0x0000000196c50a68 CoreGraphics`draw_glyphs + 576
  frame #20: 0x0000000196cf03dc CoreGraphics`draw_glyphs + 844
  frame #21: 0x00000001969a3504 CoreGraphics`cid_draw + 316
  frame #22: 0x000000019699a4c0 CoreGraphics`CGPDFTextLayoutDrawGlyphs + 100
  frame #23: 0x0000000196a7cc04 CoreGraphics`op_Tj + 80
  frame #24: 0x000000019691d990 CoreGraphics`pdf_scanner_handle_xname + 128
  frame #25: 0x000000019691cbf4 CoreGraphics`CGPDFScannerScan + 456
  frame #26: 0x000000019691acc0 CoreGraphics`CGPDFDrawingContextDrawPage + 720
  frame #27: 0x000000019691a950 CoreGraphics`pdf_page_draw_in_context + 184
  frame #28: 0x000000019691a7d8 CoreGraphics`CGContextDrawPDFPageWithDrawingCallbacks + 272
  frame #29: 0x000000019691a618 CoreGraphics`CGContextDrawPDFPageWithAnnotations + 76
  frame #30: 0x0000000194caddbc AppKit`-[NSPDFImageRep draw] + 228
```

PDFs & Fonts

- PDFKit & CoreText don’t really care!
- Font renderer decides
- Supports extra:
- *.dfont (Macintosh)*
- *WOFF/WOFF2*
- *Bitmap fonts*

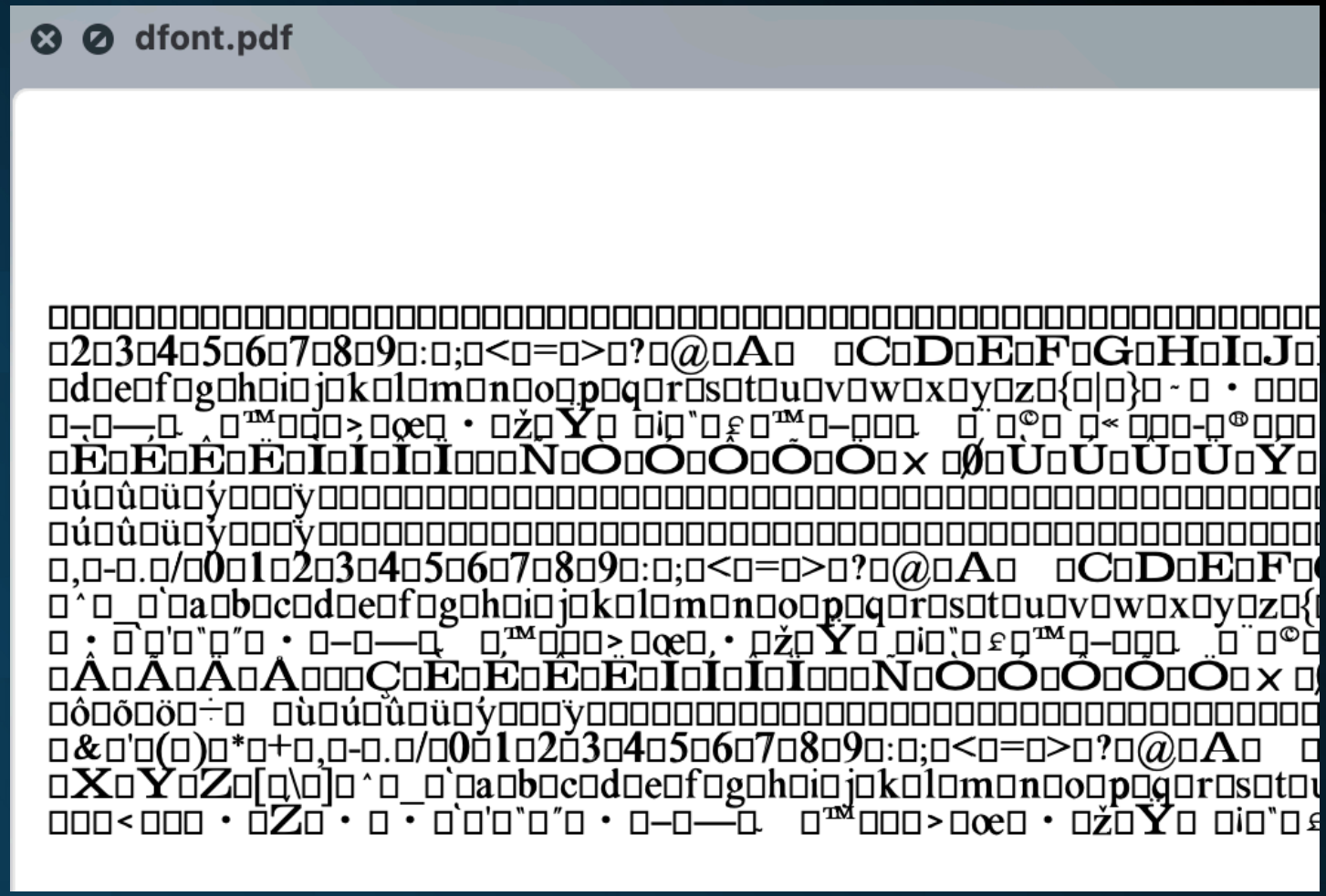
PDF standard supports limited font types (PS, TTF, OTF)

```
case 1, 2
    *(arg1 + 0x120) = _CGPDFFontType1Load(arg1)
case 3
    *(arg1 + 0x120) = _CGPDFFontTrueTypeLoad(arg1)
case 5
    *(arg1 + 0x120) = _CGPDFFontCIDType0Load(arg1)
case 6
    *(arg1 + 0x120) = _CGPDFFontCIDType2Load(arg1)
```

An interesting discovery

- System won't preview them or load them
- Get rendered in PDFs

.dfonts & others



Back to the vulnerability

```
PUSHB[ ]
2 11
FLIPRG
INSTR144[ ]
RTG[ ]
PUSHB[ ]
7
SRP1[ ]
PUSHB[ ]
6
SRP2[ ]
PUSHB[ ]
21
IP[ ]
SVTCA[1]
SFVTL[1]
```

We can reach the vulnerable Instruction

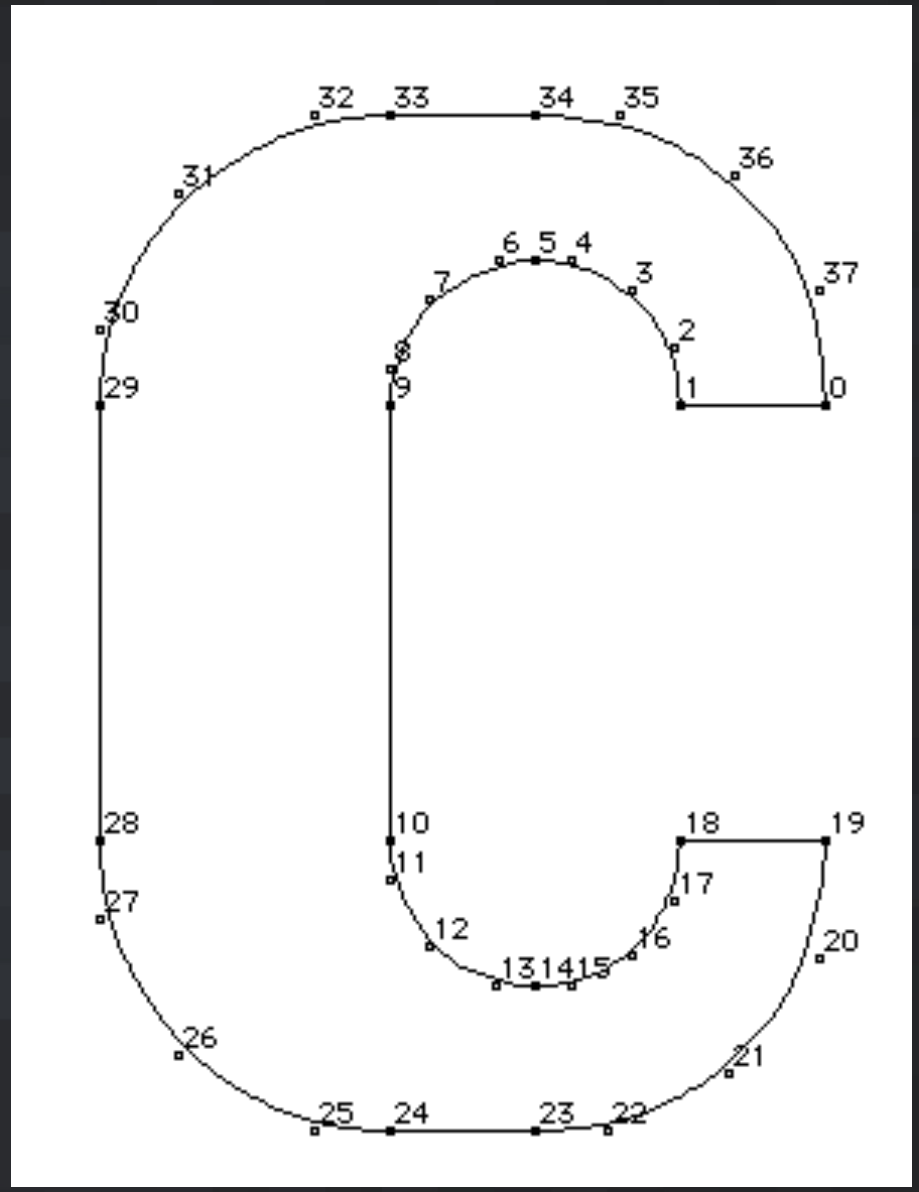
```
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
  * frame #0: 0x00007ff823787186 libFontParser.dylib`fnt_ADJUST(fnt_LocalGraphicStateType*)
    frame #1: 0x00007ff82379a651 libFontParser.dylib`fnt_InnerExecute
    frame #2: 0x00007ff82379a97f libFontParser.dylib`fnt_Execute
    frame #3: 0x00007ff8237bbe37 libFontParser.dylib`RunGlyphProgram
    frame #4: 0x00007ff82376019f libFontParser.dylib`StretchGlyph
    frame #5: 0x00007ff823761799 libFontParser.dylib`CreateGlyphOutline
    frame #6: 0x00007ff823768856 libFontParser.dylib`CreateScalerGlyphBlock
    frame #7: 0x00007ff8237663e9 libFontParser.dylib`AssureGlyphBlock
    frame #8: 0x00007ff82375d34e libFontParser.dylib`TTRenderGlyphs
    frame #9: 0x00007ff823758df2 libFontParser.dylib`TConcreteFontScaler::CopyGlyphPath
    frame #10: 0x00007ff8237acbf4 libFontParser.dylib`TFPFont::CopyGlyphPath
    frame #11: 0x00007ff81fee8a39 CoreGraphics`CGFontCreateGlyphPath + 42
```

The Unknowns

1. What the instruction does
2. How many parameters it takes
3. What are the preconditions for the vulnerability

Outline fonts

- Points and Contours
- Bezier curves



Points representation

Logical representation

```
<contour>
<pt x="269" y="-14" on="1"/>
<pt x="161" y="-14" on="0"/>
<pt x="43" y="174" on="0"/>
<pt x="43" y="344" on="1"/>
<pt x="43" y="528" on="0"/>
<pt x="170" y="729" on="0"/>
<pt x="285" y="729" on="1"/>
<pt x="285" y="728" on="1"/>
<pt x="392" y="728" on="0"/>
<pt x="511" y="536" on="0"/>
<pt x="511" y="361" on="1"/>
<pt x="511" y="182" on="0"/>
<pt x="384" y="-14" on="0"/>
</contour>
```

Binary representation

struct tSimpleGlyph SimpleGlyph[0]	3 cor
SHORT numberOfContours	3
SHORT xMin	12
SHORT yMin	0
SHORT xMax	650
SHORT yMax	717
⌄ USHORT endPtsOfContours[3]	
USHORT endPtsOfContours[0]	13
USHORT endPtsOfContours[1]	17
USHORT endPtsOfContours[2]	75

Code representation

```
struct fnt_ElementType
{
    int32_t contourCount;
    int32_t pointCount;
    uint32_t* HintedOutline_x;
    uint32_t* HintedOutline_y;
    uint16_t* contourStartPoints;
    uint16_t* contourEndPoints;
    char* onCurve;
    uint8_t* flags;
    uint32_t* UnhintedOutline_x;
    uint32_t* UnhintedOutline_y;
```

What the function does

It doesn’t matter

Find a specified curve

Iterate over its points

“Adjust” one dimension

```
int32_t* hintedOutline_target = (&field_0->HintedOutline_x)[rax]
int64_t fnt_GetCVTEntryFast = arg1_1->fnt_GetCVTEntryFast
arg1_1->stack_top = stack_top_1 - 4
int32_t ctrlValTableEntry = fnt_GetCVTEntryFast(arg1_1, zx.q(*(stack_top_1 - 4)))

if (ctrlValTableEntry u< 0x8000)
    uint32_t cvtEntry = (ctrlValTableEntry + 0x20) u>> 6
    uint32_t cvtEntry_1 = 1

    if (cvtEntry != 0)
        cvtEntry_1 = cvtEntry

    if (arg1_1->loop s>= 0)
        while (true)
            struct stack_struct* stack_mem = arg1_1->stack_mem

            if (stack_mem->stack_size s<= 0)
                thrown_exception = ___cxa_allocate_exception(2)
                rax_12 = OFAErrors(0xe)
                goto label_7ff8093e34d4

            void* stack_top = arg1_1->stack_top
            void* stack_ptr = stack_mem->stack_ptr

            if (stack_ptr u>= stack_top)
                thrown_exception = ___cxa_allocate_exception(2)
                rax_12 = OFAErrors(0xe)
                goto label_7ff8093e34d4

            arg1_1->stack_top = stack_top - 4

            if (stack_ptr u>= stack_top - 4)
                thrown_exception = ___cxa_allocate_exception(2)
                rax_12 = OFAErrors(0xe)
                goto label_7ff8093e34d4

            uint64_t param1 = zx.q(*(stack_top - 4))
            arg1_1->stack_top = stack_top - 8
            uint64_t param2 = zx.q(*(stack_top - 8))
            CHECK_ELEMENTPTR(arg1_1, field_0)
```

How many parameters?

1. Control Value Table index
2. Contour point index
3. Contour point index

What it really does

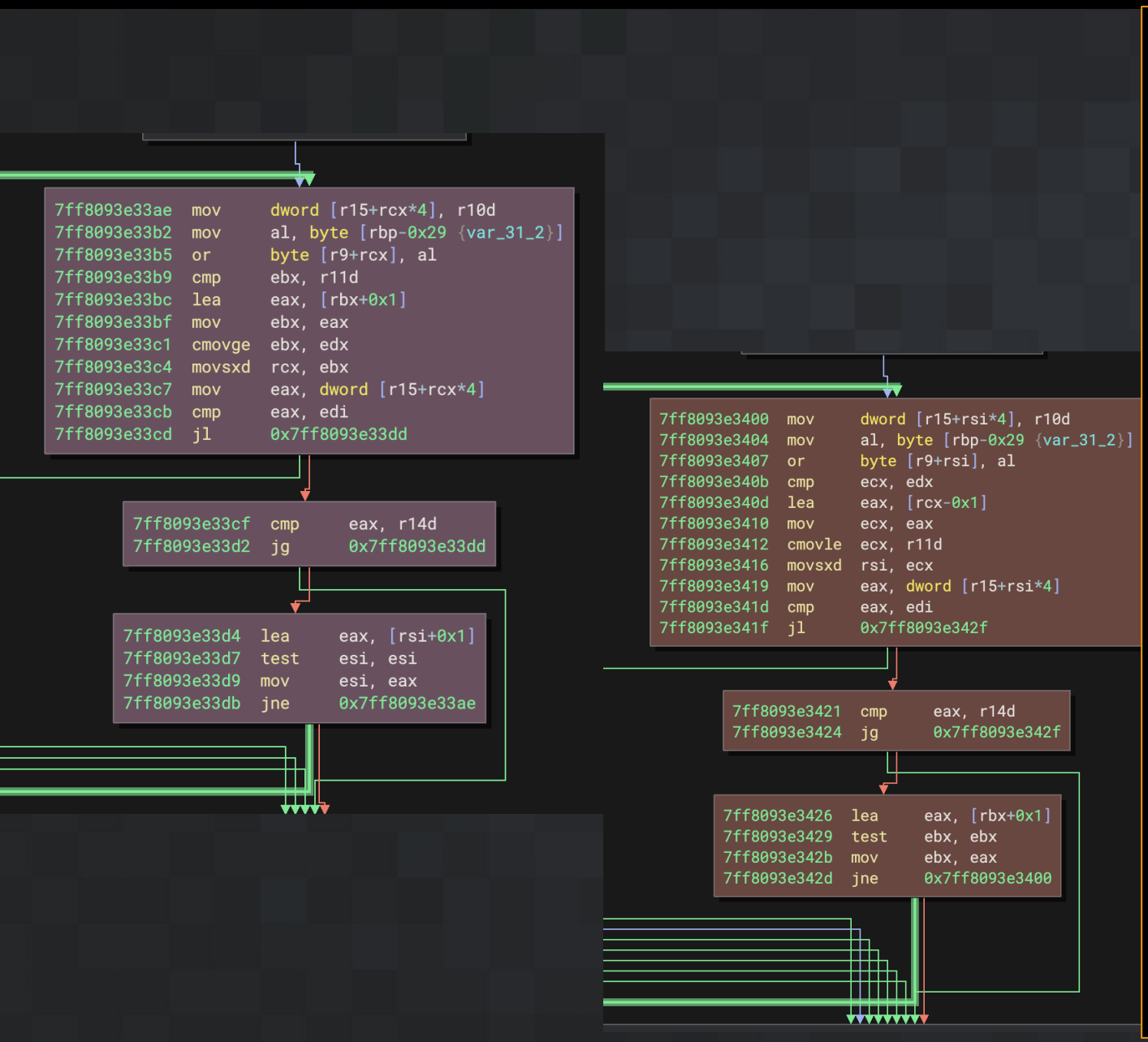
The effects important for vulnerability

Index into a memory buffer

Iterate over the elements

Write new value to each element

What it really does




```
        points = [...]
        pointsEnd = points[-1]
        pointsStart =

def adjust(target_point)
        start, finish = find_contour(target_point)
        num_points = finish-start
        idx = target_point
        while True:
                points[idx] = new_value
                idx+=1
                if idx > finish:
                        idx = start
                num_points-=1
                if numpoints = 0
                        break
        num_points = finish-start
        idx = target_point - 1

        while True:
                points[idx] = new_value
                idx-=1
                if idx < start:
                        idx = finish
                num_points-=1
                if numpoints = 0
                        break
```

Path to memory corruption

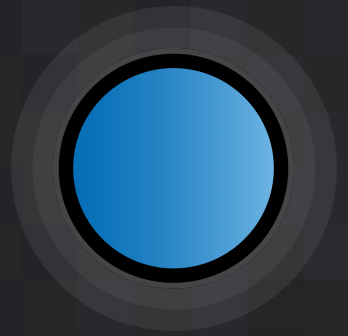
- Get index to point out of bounds of the buffer

```
do
    hintedOutline_target[idx] = coordNew
    rax_8.b = var_31_1 + 1
    flags[idx] |= rax_8.b
    bool cond:10_1 = rcx_5 s<= start
    rcx_5 -= 1

    if (cond:10_1)
        rcx_5 = finish

    idx = sx.q(rcx_5)
    int32_t rax_10 = hintedOutline_target[idx]

    if (rax_10 s< rdi_7)
        break

    if (rax_10 s> r14_1)
        break

    cond:8_1 = rbx_11 != 0
    rbx_11 += 1
while (cond:8_1)
```

What do we control

At the time of the instruction call

Number of points



Total number of points per contour in a glyph

Number of contours

Total number of contours per glyph

Target contour

We can specify which contour we invoke ADJUST on (first, last , etc.)

Target point in the contour

Target point inside the contour (the one on which the operation starts)

Size of the buffer

Indirectly controlled by total number of points and maxPointsCount

An important bit

Finding the contour

```
int64_t idx_1 = zx.q(param1.d)
int32_t coordEdge = hintedOutline_target[idx_1]
int64_t contourOffset = sx.q(field_0->contourCount) * 2
int32_t start

do
    start = sx.d(*(field_0->contourStartPoints + contourOffset - 2))
    contourOffset -= 2
while (param1.d s< start)
```

Is it that simple?

Well, no

- Constraints — Limits to index values
- Memory — What comes after the array
- Control — Controlling the value written

Exploitation speculation

Until I see the original sample

FORCEDENTRY

Complex weird machine built around JBIG2 vulnerability

CVE-2021-30860

BLASTPASS

An intricate heap overflow vulnerability in WebP

CVE-2023-4863

Unnamed SBX Escape

Ian Beer's last year presentation, ridiculous NSExpression based exploitation framework

CVE-2023-32409

Conclusions

1. Variant analysis?
2. “Signature” for an exploit?
3. Fuzzing takeaways & new attack surface

Main highlights

“Why didn’t fuzzing caught this?” – A LOT of preconditions + unfriendly file format

Curious difference between logical, binary and memory font representation – common culprit for vulnerabilites

Circumstances make things difficult: patch diffing without a patch, reconstructing without a sample , bottom up analysis

REFERENCES

Kaspersky reports

https://web.archive.org/web/20240208170009/https://securelist.com/trng-2023/

https://media.ccc.de/v/37c3-11859-operation_triangulation_what_you_get_when_attack_iphones_of_researchers

Mateusz “j00ru” Jurczyk

https://j00ru.vexillium.org/slides/2015/44con.pdf

https://j00ru.vexillium.org/slides/2015/recon.pdf

https://googleprojectzero.blogspot.com/2015/07/one-font-vulnerability-to-rule-them-all.html

https://github.com/googleprojectzero/BrokenType

https://googleprojectzero.blogspot.com/2016/06/a-year-of-windows-kernel-font-fuzzing-1_27.html

https://googleprojectzero.blogspot.com/2016/07/a-year-of-windows-kernel-font-fuzzing-2.html

Tools

https://github.com/fonttools/fonttools

https://github.com/fontforge/fontforge

Misc

https://github.com/ZoloZiak/WinNT4/blob/master/private/ntos/w32/ntgdi/fondrv/tt/scaler/interp.c#L3529

https://github.com/elliotnunn/supermario/blob/master/base/SuperMarioProj.1994-02-09/Toolbox/FontMgr/fnt.c#L385

https://learn.microsoft.com/en-us/typography/opentype/spec/ttch01

Aleksandar Nikolic

Your title

@FuzzyAleks (infosec.exchange\| bsky.social)

Thank you!

TALOSINTELLIGENCE.COM

blog.talosintelligence.com

@talossecurity