一枚字体CRASH到FUZZING的探索

演讲者：

知道创宇404实验室-李松林
(sunglin from the Knownsec 404 team)

目录
CONTENTS

PART 01

fuzzing的介绍

介绍

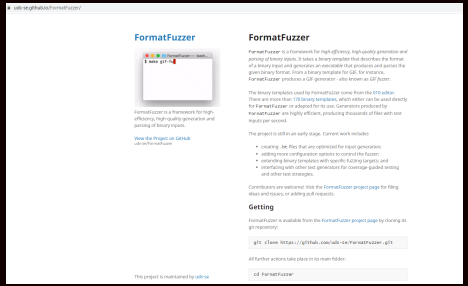
2020是个非常奇妙与难忘的一年，同样fuzzing技术在2020年技术井喷，涌现了一大批fuzzing工具和前沿的研究；
在2020年的年终时候，我在知乎上对2020年的开源fuzzing做了一些总结与些许研究，一个开源的研究与项目叫FormatFuzzer，这个项目的出现，让我想起了aflsmart的开源fuzzing项目，两者都是类似的文本结构fuzzing，和aflsmart不同的是FormatFuzzer项目还处于开发阶段；
通过我对aflsmart fuzzing项目的熟悉，我决定以Formartfuzzer为契机来玩一下。

https://zhuanlan.zhihu.com/p/344008210

Formartfuzzer

Formartfuzzer项目地址：

https://uds-se.github.io/FormatFuzzer/

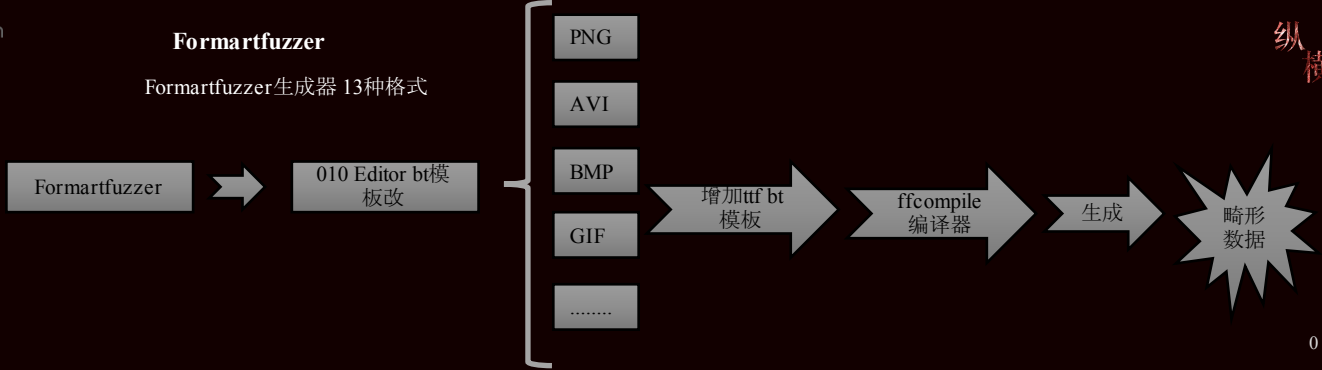


PART 02

fuzzing这么好玩，就crash了

Formartfuzzer

Formartfuzzer生成器 13种格式



ttf结构

1、9个table表必须出现在任何有效的 TrueType 字体文件中
2、这些表可以按任何顺序出现
3、30多个特殊的表

Tag	Table
'cmap'	character to glyph mapping
'glyf'	glyph data
'head'	font header
'hhea'	horizontal header
'hmtx'	horizontal metrics
'loca'	index to location
'maxp'	maximum profile
'name'	naming
'post'	PostScript

010 Editor ttf字体结构name视图


```
typedef struct  {
// http://www.microsoft.com/typography/OTSPEC/name.htm
    local quad name_table = FTell();
    USHORT format;  // Format selector (=0).
    USHORT count;   // Number of name records.
    USHORT stringOffset;    // Offset to start of string storage (from start of table).
    local quad NextNameRecord;

    struct tNameRecord {
        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).
        NextNameRecord = FTell();
        FSeek(name_table + stringOffset + offset);
        char name[length];
        FSeek(NextNameRecord);
    } NameRecord[count] <read=NameRecordRead, optimize = false>;
}tname <read=NameRecordRead>;
```

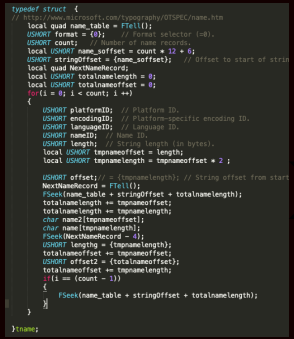
Template Results – TTF.bt

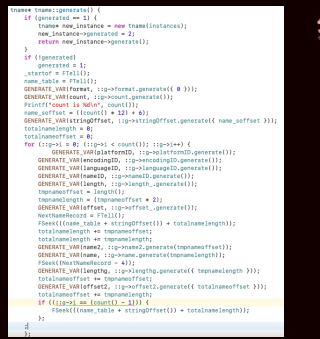
Name	Value	Start	Size
struct tname name	36 Names	78A34h	1B6h
USHORT format	0h	78A34h	2h
USHORT count	24h	78A36h	2h
USHORT stringOffset	1B6h	78A38h	2h
struct tNameRecord NameRecord[0]	p1 E0 LD Copyright(c) 2009 M+ FONTS PROJECT	78A3Ah	Ch
USHORT platformID	1h	78A3Ah	2h
USHORT encodingID	0h	78A3Ch	2h
USHORT languageID	0h	78A3Eh	2h
USHORT nameID	0h	78A40h	2h
USHORT length	22h	78A42h	2h
USHORT offset	46h	78A44h	2h
char name[34]	Copyright(c) 2009 M+ FONTS PROJECT	78C30h	22h
struct tNameRecord NameRecord[1]	p1 E0 LD M+ 1p	78A46h	Ch
struct tNameRecord NameRecord[2]	p1 E0 LD medium	78A52h	Ch
struct tNameRecord NameRecord[3]	p1 E0 LD FontForge 2.0 : M+ 1p medium : 2-2-2009	78A5Eh	Ch
struct tNameRecord NameRecord[4]	p1 E0 LD M+ 1p medium	78A6Ah	Ch
struct tNameRecord NameRecord[5]	p1 E0 LD Version 1.021	78A76h	Ch
struct tNameRecord NameRecord[6]	p1 E0 LD mplus-1p-medium	78A82h	Ch
struct tNameRecord NameRecord[7]	p1 E0 LD http://mplus-fonts.sourceforge.jp	78A8Eh	Ch
struct tNameRecord NameRecord[8]	p1 E0 LD All Typographic Features	78A9Ah	Ch
struct tNameRecord NameRecord[9]	p1 E0 LD Ligatures	78AA6h	Ch
struct tNameRecord NameRecord[10]	p1 E0 LD All Type Features	78AB2h	Ch
struct tNameRecord NameRecord[11]	p1 E0 LD Common Ligatures	78ABEh	Ch
struct tNameRecord NameRecord[12]	p1 E0 LI Fonctions typographiques	78ACAh	Ch
struct tNameRecord NameRecord[13]	p1 E0 LI Ligatures	78AD6h	Ch
struct tNameRecord NameRecord[14]	p1 E0 LI Toutes fonctions typographiques	78AE2h	Ch
struct tNameRecord NameRecord[15]	p1 E0 LI Ligatures Usuelles	78AEEh	Ch
struct tNameRecord NameRecord[16]	p1 E0 LZ Alle typografischen Möglichkeiten	78AFAh	Ch
struct tNameRecord NameRecord[17]	p1 E0 LZ Ligaturen	78B06h	Ch

TTF 010 BT模板的改造

21 table的改写

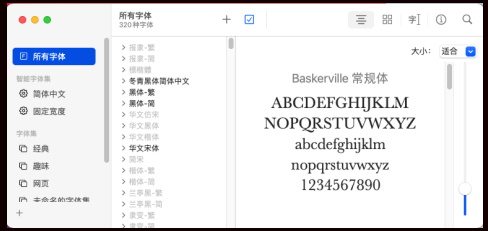
010 editor的bt模板 → 改造 → ttf模板生成器 → pfp解释器





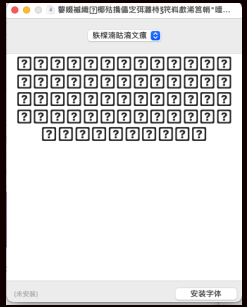
字体fuzzing目标的选定

macos ?
/System/Applications/Font Book.app/Contents/MacOS/Font Book



ttf模板生成器生成畸形字体

通过Font Book打开？

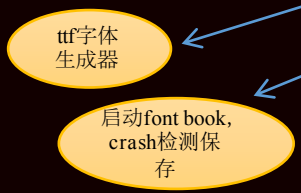


fuzzing start ?

条件：

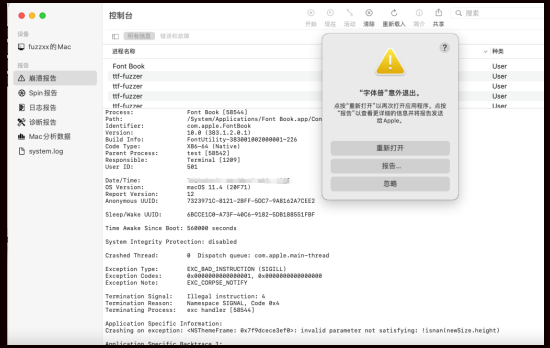
1、自动启动和关闭；
2、产生畸形数据
3、可以检测crash与保存crash

fuzzing脚本：




```
# ! /bin/bash
for I in {1..200000};do
   ./ttf-fuzzer fuzz test.ttf
   sleep 1s
   ./test test.ttf  &
   echo "sleep start\n"
   sleep 1s
   echo "kill start\n"
   ps -efww|grep -w 'Font'|grep -v grep|cut -c 7-17|xargs  kill -9
done
   echo "over\n"
   exit 0
```

经过24小时的fuzzing.....
fuzzing这么好玩，就crash了



crash 字体分析

1、head.unitsPerEm = 0
（设置为 16 到 16384 之间的值。此范围内的任何值均有效）
2、name.nameID = 1(字体系列)

以上设字体参数设置将会导致font book参数异常，而引发崩溃

fuzzing这么好玩，就crash了

加入Formartfuzzer 自带变异算法，如 smart_replace、smart_delete、smart_insert、smart_abstract、smart_swap，重新启用脚本fuzzing，

等待数天.jpg

2 x 24h fuzzing测试后.....

0 crash.....

思考

1、对于字体特殊的table组成，随机剪切、替换会引发很多无效的数据；
2、没有有效的反馈，做了无用功；
3、是不是该做点fuzzing进阶了。

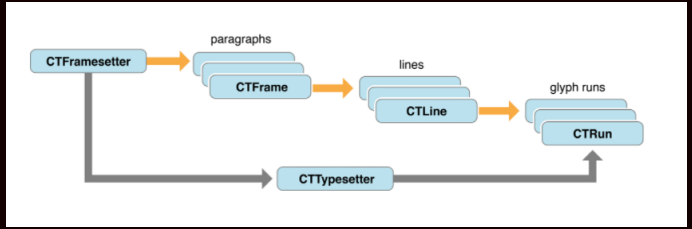
比如，apple字体框架是如何解析的？加入程序路径覆盖率反馈等

PART 03

进阶！apple 字体处理框架的fuzzing

apple 字体处理框架

Core Text 是一种用于布局文本和处理字体的高级低等级技术。Core Text 直接与 Core Graphics (CG) 一起工作，也称为 Quartz，它是一种高速图形渲染引擎，可在 OS X 和 iOS 的最低级别处理二维图像。



CoreText框架

基础框架：

CTFrameSetter：相当于一个CTFrame工厂，来生产CTFrame，一个界面上可以有多个CTFrame；

CTFrame：可以视为一个画布，范围是由CGPath(图形路径)来决定，其后的绘制是只在这个范围内绘制；

CTLine：一个CTFrame由多个CTLine组成，一行就是一个CTLine；

CTRun：一个CTLine是由一个或多个CTRun组成，可以理解为一个块，当一个CTLine中包含了多个不同属性时，比如字体、颜色、Attachment等，都会通过CTRun将CTLine分隔开；

apple 字体对象的创建

```
func CTFontCreateWithName(_ name: CFString, _ size: CGFloat,  _ matrix: UnsafePointer<CGAffineTransform>?) -> CTFont

CGFontRef CGFontCreateWithDataProvider(CGDataProviderRef provider);

func CTFontCreateWithGraphicsFont(_ graphicsFont: CGFont,
                                  _ size: CGFloat,
                                  _ matrix: UnsafePointer<CGAffineTransform>?,
                                  _ attributes: CTFontDescriptor?) -> CTFont
```

创建字体属性字符串

Core Text 布局引擎通常使用属性字符串 (CFAttributedStringRef) 和图形路径 (CGPathRef)。属性字符串对象封装了支持显示文本的字符串，并包括定义字符串中字符的风格方面的属性（或“属性”）

```
CFMutableAttributedStringRef attributeRef = CFAttributedStringCreateMutable(kCFAllocatorDefault, 0);
void CFAttributedStringSetAttribute(CFMutableAttributedStringRef attributeRef , CFRange range, CFStringRef attrName, CFTypeRef value);
```

从属性字符串创建一个不可变的 framesetter 对象

CTFramesetter是CTFrame对象的对象工厂。

framesetter 接受一个属性字符串对象和一个形状描述符对象，并调用排版器来创建填充该形状的线条对象。输出是一个包含行数组的框架对象。然后框架可以将自身直接绘制到当前图形上下文中。

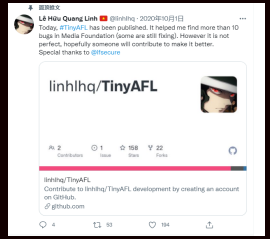
func CTFramesetterCreateWithAttributedString(_ attrString: CFAttributedString) -> CTFramesetter

其他处理属性api：

1、CTLine ：
一个CTLine对象包含一组字形运行。Line 对象由排版员在框架设置操作期间创建，并且可以将其自身直接绘制到图形上下文中。
CTLineCreateWithAttributedString、CTLineDraw
2、CTTypesetter：
一个排版机，执行线路布局。
CTTypesetterCreateWithAttributedString
3、CTGlyphInfo：
覆盖字体从 Unicode 到字形 ID 的指定映射。
CTGlyphInfoCreateWithGlyph
CTRunDraw

google 轻量级动态插装库TinyInst

TinyInst 可在 Windows（32 位和 64 位）和 macOS（64 位）上使用，很快就有人将TinyInst移植到了afl中，TinyInst主要是以inter xen为反汇编引擎为主做的指令级解析



fuzzing 进阶：

Formartfuzzer + AFL + **TinyInst？ -> apple font frame**

fuzzing start
条件：

1、自动启动和关闭；
2、产生畸形数据
3、可以检测crash与保存crash
4、可以获取程序路径覆盖率

2 x 24h fuzzing 测试后....

1 crash get .....


```
┌─ process timing ──────────────────────────────────┐ ┌─ overall results ──────────────┐
│        run time : 2 days, 1 hrs, 7 min, 8 sec     │ │  cycles done : 1130            │
│   last new path : 0 days, 0 hrs, 5 min, 30 sec    │ │  total paths : 757             │
│ last uniq crash : 0 days, 2 hrs, 19 min, 2 sec    │ │ uniq crashes : 1               │
│  last uniq hang : 0 days, 0 hrs, 20 min, 21 sec   │ │   uniq hangs : 40              │
├─ cycle progress ──────────────────┐ ┌─ map coverage ──────────────────────────────┤
│  now processing : 655 (86.53%)    │ │   map density : 9.28% / 23.96%              │
│ paths timed out : 1 (0.13%)       │ │ count coverage : 1.11 bits/tuple            │
├─ stage progress ──────────────────┤ ├─ findings in depth ─────────────────────────┤
│  now trying : havoc               │ │ favored paths : 220 (29.06%)                │
│ stage execs : 452/614 (73.62%)    │ │  new edges on : 413 (54.56%)                │
│ total execs : 6.16M               │ │ total crashes : 1 (1 unique)                │
│  exec speed : 19.88/sec (zzzz...) │ │  total tmouts : 1241 (40 unique)            │
├─ fuzzing strategy yields ─────────┴─┴─ path geometry ──────────────────────────────┤
│   bit flips : n/a, n/a, n/a                  │ │    levels : 21                     │
│  byte flips : n/a, n/a, n/a                  │ │   pending : 10                     │
│ arithmetics : n/a, n/a, n/a                  │ │  pend fav : 1                      │
│  known ints : n/a, n/a, n/a                  │ │ own finds : 591                    │
│  dictionary : n/a, n/a, n/a                  │ │  imported : n/a                    │
│       havoc : 437/1.72M, 155/3.43M           │ │ stability : 77.48%                 │
│        trim : 3.61%/1.00M, n/a               │ │                                    │
└──────────────────────────────────────────────┘ └────────────────────────────────────┘
```

core text ttf越界写：

```
GuardMalloc[tttfest-loader-94a1]:  - Some buffer overruns may not be noticed.
GuardMalloc[tttfest-loader-94a1]:  - Applications using vector instructions (e.g., SSE) should work.
GuardMalloc[tttfest-loader-94a1]: version 40x638.38.1
UndefinedBehaviorSanitizer:DEADLYSIGNAL
==94a1==ERROR: UndefinedBehaviorSanitizer: SEGV on unknown address 0x0001102abc8d (pc 0x7fff3312e2b0 bp 0x7ffee43f5000 sp 0x7ffee43e72d0 T8603)
==94a1==The signal is caused by a WRITE memory access.
    #0 0x7fff3312e2b0 in (anonymous namespace)::MorxLigatureSubtableBuilder::FromLigInputs(long, unsigned int, unsigned int, TInlineVector<std::__1::vector<unsigned short, std::__1::allocator<unsigned short> >, 30ul> const&, std::__1::__wrap_iter<(anonymous namespace)::LigInput*>, std::__1::__wrap_iter<(anonymous namespace)::LigInput*>, std::__1::vector<unsigned short, std::__1::allocator<unsigned short> > const&)+0x15a (CoreText:x86_64+0x1182b0)
    #1 0x7fff3312ab68 in (anonymous namespace)::MorxTableBuilder::FromArabicPresentationForms(TFont const&)+0x53ce (CoreText:x86_64+0x114068)
    #2 0x7fff3312ab36 in ConvertArabicPresentationFormsToMorxInternal(TFont const&)+0xe5 (CoreText:x86_64+0x10eb36)
    #3 0x7fff330a342e in invocation function for block in ConvertToMorxAsync(__CTFont const*)+0x11 (CoreText:x86_64+0x8d426)
    #4 0x7fff3617e667 in _dispatch_client_callout+0x7 (libdispatch.dylib:x86_64+0x2667)
    #5 0x7fff36b18ae6b in _dispatch_lane_barrier_sync_invoke_and_complete+0x3b (libdispatch.dylib:x86_64+0xe6eb)
    #6 0x7fff3306d88d in ConvertToMorxAsync(__CTFont const*)+0x19a (CoreText:x86_64+0x37888)
    #7 0x7fff3305623b in TASCIIEncoder::Encode()+0x2b (CoreText:x86_64+0x4023b)
    #8 0x7fff33055ca6 in TGlyphEncoder::EncodeChars(CFRange, TAttributes const&, TGlyphEncoder::Fallbacks)+0x3fe (CoreText:x86_64+0x3fca6)
    #9 0x7fff33055390 in TTypesetterAttrString::Initialize(__CFAttributedString const*)+0x122 (CoreText:x86_64+0x3f390)
    #10 0x7fff3305586ac in TTypesetterAttrString::TTypesetterAttrString(__CFAttributedString const*, __CFDictionary const*)+0xa2 (CoreText:x86_64+0x3f0ac)
    #11 0x7fff3306b0cd in TCFRefCFTframesetterAttrString> TCFBase_NEW<CTFramesetterAttrString, __CFAttributedString const*&>(__CFAttributedString const*&)+0x58 (CoreText:x86_64+0x750cd)
    #12 0x7fff3308b038 in CTFramesetterCreateWithAttributedString+0x2f (CoreText:x86_64+0x75038)
    #13 0x10b885c60 in fuzz+0x2d0 (tttfest-loader:x86_64+0x100003c60)
    #14 0x10b885d8a in main+0x10a (tttfest-loader:x86_64+0x100003d8a)
    #15 0x7fff6b1d7cc8 in start+0x0 (libdyld.dylib:x86_64+0x1acc8)

==94a1==Register values:
rax = 0x0000000000000d0d  rbx = 0x0000000118297920  rcx = 0x0000000110aaf00  rdx = 0x0000000000000002
rdi = 0x0000000000000d0b  rsi = 0x0000000118297793a  rbp = 0x00007ffee43f5000  rsp = 0x00007ffee43e72d0
 r8 = 0x0000000000000000  r9 = 0x0000000110a2c090  r10 = 0x0000000020000001  r11 = 0x0000000000000206
r12 = 0x0000000118297790c  r13 = 0x0000000000e30009  r14 = 0x0000000110a2c560  r15 = 0x0000000118297790c
UndefinedBehaviorSanitizer can not provide additional info.
SUMMARY: UndefinedBehaviorSanitizer: SEGV (CoreText:x86_64+0x1182b0) in (anonymous namespace)::MorxLigatureSubtableBuilder::FromLigInputs(long, unsigned int, unsigned int, TInlineVector<std::__1::vector<unsigned short, std::__1::allocator<unsigned short> >, 30ul> const&, std::__1::__wrap_iter<(anonymous namespace)::LigInput*>, std::__1::__wrap_iter<(anonymous namespace)::LigInput*>, std::__1::vector<unsigned short, std::__1::allocator<unsigned short> > const&)+0x15a
==94a1==ABORTING
zsh: abort      DYLD_INSERT_LIBRARIES=/usr/lib/libgmalloc.dylib ./tttfest-loader
fuzz@fuzzxxdeMac Documents %
```

阶段性总结

Formartfuzzer 优缺点

1、可以依照改造的010 bt模板生成新的ttf字体（优点）

2、生成的数据过于随机化，细化度不够，缺乏可靠性

3、数据结构过于单一等

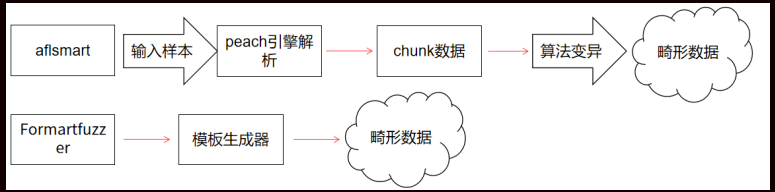
挖洞的思考：同样的fuzzing挖不同的攻击面，不同的fuzzing挖相同的攻击面，畸形数据多样性与可靠性

PART 04

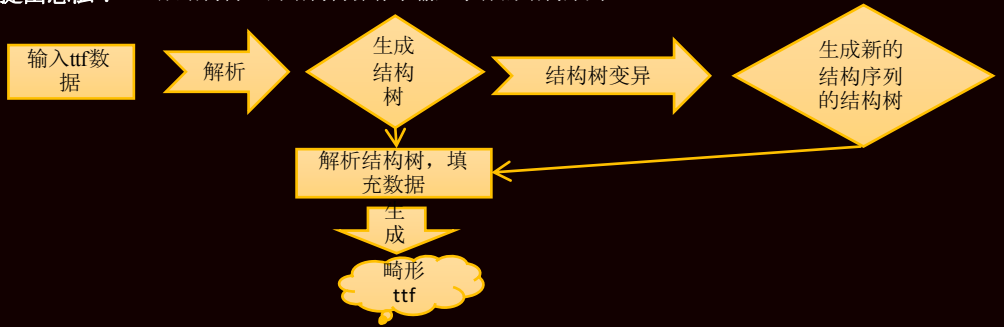
升级！ fuzzing的探索与改造

对比前：

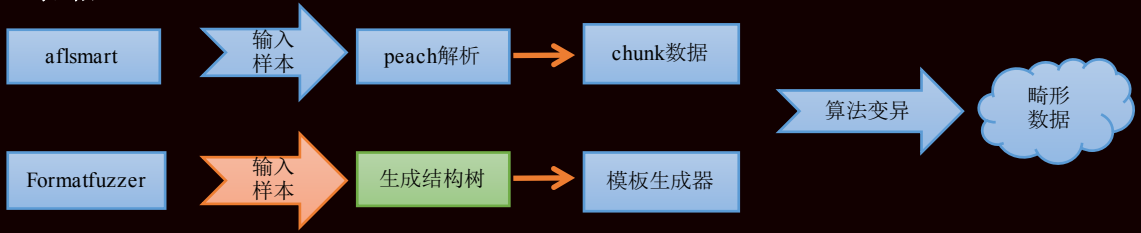
aflsmart通过调用peach对输入图片数据进行解析->生成chunk数据，对chunk数据进行算法的变异，生成新的有效畸形数据，对比Formartfuzzer通过定义010的bt版本模板，根据模板生成畸形数据，再加上算法对畸形数据再一次变异



提出想法： 生成结构树，而结构树保存了输入字体的结构序列



对比后：


aflsmart
输入
样本
peach解析
chunk数据
Formatfuzzer
输入
样本
生成结构树
模板生成器
算法变异
畸形
数据

结构树的作用？

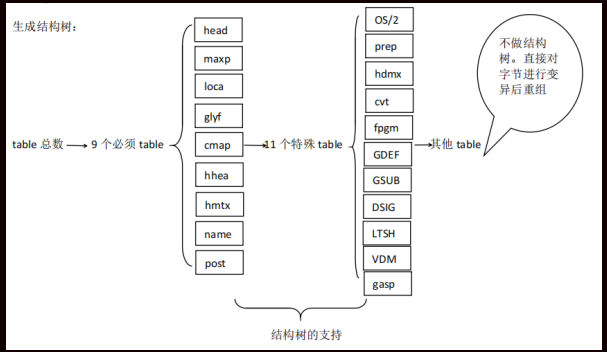
1、通过依靠可用字体结构序列解决了 Formartfuzzer依据bt模板生成数据的不可靠性，比如可以解决SHORT、LONG、LONG　LONG随机性过大造成实际数据的无效

2、通过提取了输入了有效字体的结构序列，可直接增强生成样本的多样性

3、可变异的结构序列

4、减轻bt模板对offset、length的计算等等

生成结构树：
结构树必须包含主要9个table的支持



结构树格式：

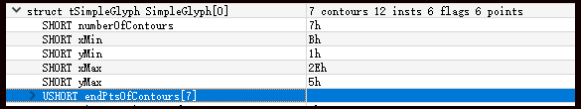
1、结构体成员： 1 -> 有此成员；0 -> 无

2、结构体子类数据: 1 : type : data

```
if(data)
{
    1:type D: data
}
else
{
    0
}
```

结构树格式：

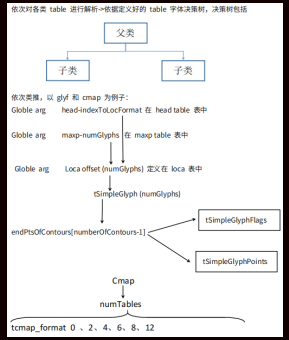
3、结构体父类到子类: 1 : type struc struc numberofContours -> endPtsOfContours[numberofContours]



4、结构体子类到固定值: 1 :xxxxtype data

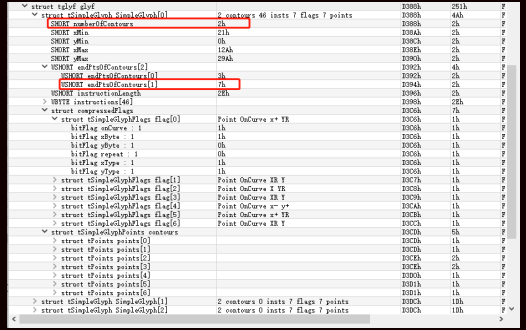
5、结构体父类到子类数组 4:type numb number

结构树视图：
Table表如下：
Head、 cmap、glyf 等等



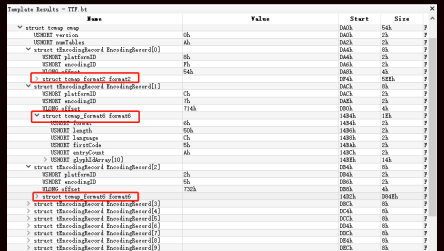
ttf字体结构中的glyf

```
glyf:
01:02S 0700 01 01 01 01 01:02S 0600 01:02S 0c00
01:02S 0f00 01 01 01 01 01:02S 0c00 01:02S 1000
01:02S 0700 01 01 01 01 01:02S 0700 01:02S 0a00
01:02S 0300 01 01 01 01 01:02S 0500 01:02S 0e00
01:02S 0200 01 01 01 01 01:02S 0100 01:02S 0a00
01:02S 0c00 01 01 01 01 01:02S 0f00 01:02S 0c00
01:02S 5500 01 01 01 01 01:02S 0f00 01:02S 0700
01:02S 0500 01 01 01 01 01:02S 0700 01:02S 0300
01:02S 0d00 01 01 01 01 01:02S 0500 01:02S 0100
01:02S 6700 01 01 01 01 01:02S 0e00 01:02S 6000
01:02S 0400 01 01 01 01 01:02S 3500 01:02S cd00
01:02S 0700 01 01 01 01 01:02S 1600 01:02S 0200
```



ttf字体结构中的cmap


```
cmap:
00 01:02S 0a00 01:02D 0800 01:02D 0f00 01:04X 54000000 01:02X 0200
decode_cmap2:
01:02X c006 01:02D 0a00 01:02D 0c00 01:02D 0700 01:04X 14070000 01:02X 0600
decode_cmap6:
01:02X 5000 01:02D 0c00 01:02D 0500 01:02S 0a00 01:02D 0200 01:02D 0500 01:04X 32070000 01:02X 0600
decode_cmap6:
01:02X 1000 01:02D 0c00 01:02D 0a00 01:02S 226c 01:02D 0900 01:02D 0500 01:04X 80df0000 01:02X 0200
decode_cmap2:
01:02X c006 01:02D 0900 01:02D ff7f 01:02D 0300 01:04X 40e60000 01:02X 0400
decode_cmap4:
01:02X 5600 01:02D 0800 01:02S 0e00 01:02D 0a00 01:02D 0f00 01:02X 0500 01:02D 0400 01:02D 1000 01:04X 96e60000 01:02X 0400
decode_cmap4:
01:02X 2900 01:02D 0c00 01:02S 0400 01:02D 0a00 01:02D 0600 01:02X 0700 01:02D 0300 01:02D 4e00 01:04X bee60000 01:02X 0600
decode_cmap6:
01:02X 0200 01:02D 0100 01:02D 0f00 01:02S 0100 01:02D 0100 01:02D 0d00 01:04X cae60000 00
decode_cmap0:
01:02X 0601 01:02D 1000 04:01N a4000000 01:02D 0100 01:02D 0a00 01:04X d0e70000 01:02X 0200
decode_cmap2:
01:02X c006 01:02D 0d00 01:02D 0300 01:02D 0700 01:04X 90ee0000 01:02X 0400
decode_cmap4:
01:02X 5200 01:02D 0200 01:02S 1000 01:02D 0500 01:02D 0700 01:02X 0100
```



提取结构树：
输入大量字体->结构树

重组与生成
两种生成方式：1、按照结构树去生成； 2、根据提供的结构树和具体数据表来生成字体

变异：
1、对结构树变异->生成字体。
2、采用afl 算法对生成字体变异

和aflsmart统称 结构感知fuzzing

fuzzing 升级与探索：

Formartfuzzer + AFL + **TinyInst** + **结构树？** -> **apple font frame**

fuzzing start
条件：

1、自动启动和关闭；
2、产生畸形数据
3、可以检测crash与保存crash
4、可以获取程序路径覆盖率
5、输入语料库进行解析生成结构树

PART 05

继续fuzzing，收获漏洞！

收集ttf字体语料库

1、爬虫
爬虫网站主要是 ：一些国外的字体网站 ：www.dafont.com、www.fontsquirrel.com、www.myfonts.com等
2、历史字体issue crash
3、云网盘下载字体集合

结果：2w+字体

策略 结构树总结

1、字体到结构树的生成

2、结构树结构序列化文件变异

3、ttf table拆分原始数据分类保存

4、结构树结构序列化ttf table拆分

5、拆分的ttf table重组等

6、afl 算法的加持

反复筛选与测试和重复率比对的fuzzing等等工作

最终整个工程fuzzing之旅收获约7个bugs

结果

Module	Type	Memory access type at crash	table(s)	Reported	Fixed
libFontParser.dylib	.ttf	out-of-bound reads and writes	cmap	CVE-2021-30755	11.4 beta
libFontParser.dylib	.ttf	Integer overflow	glyf	CVE-2021-30760	11.5 beta
CoreText	.ttf	out-of-bound reads	GPOS	CVE-2021-30733	11.4 beta
CoreText	.ttf	out-of-bound writes	glyf	yes	11.4 beta
CoreText	.ttf	Null pointer	glyf	no	no
CoreText	.otf	out-of-bound reads	BASE	CVE-2021-30789	11.5 beta
Font Book	.ttf	Parameter abnormal	appkit	no	no

对fuzzing的探索之旅还未结束，也许只是才开始.....

工程涉及部分源文件和poc可以在以下地址找到：

https://github.com/sunglinl/ttffuzz

感谢观看！

KCon 汇聚黑客的智慧

知道创宇 | KCon