

### ### 1. ImageMagick 图片处理库简介

ImageMagick 是一个易于使用和功能强大的图片处理库，很重要的一点是ImageMagick 在比较多的后端应用中出现(比如说:WordPress和Discuz!).挖掘大型CMS 框架的漏洞的难度比较高，那么另一种思路则是通过挖掘CMS 框架使用到的第三方库漏洞让CMS 框架来触发(比如:PHPMailer等第三方库)。由 ImageMagick 库引发的CVE-2016-3714 CVE-2016-5118 远程命令执行和两个月前的Yahoo! ImageMagick Heartbleed 已经足以说明从ImageMagick 中触发的漏洞对主机安全性的危害。那么接下来我们来讨论怎么把Fuzzing 技术应用到ImageMagick 的漏洞挖掘上。

### ### 2. ImageMagick 的使用方式

Fuzzing ImageMagick 之前，先来了解ImageMagick 库是怎样使用的。

ImageMagick 在编译完成之后，在**utilities** 目录下会生成**magick**，这就是ImageMagick 图形库的命令行使用工具。**magick** 命令行工具使用如下：

```
./magick magick_command other_arguments
```

**magick\_command** 指的是执行命令。有：**identify** (查看图像信息)，**convert** (图像处理与格式转换)等，更多命令通过以下链接查询：<https://www.imagemagick.org/script/command-line-processing.php>

**other\_arguments** 指的是命令参数。对于Fuzzing ImageMagick 库，常用的命令是：**./magick convert** (接下来会慢慢讨论为何要用**convert** 命令而不用其它命令)。对ImageMagick 做读写测试的命令：**./magick convert input\_file.png output.bmp**。ImageMagick 库则会读取PNG 格式的图片并且转换成BMP 格式的图像；只对ImageMagick 做读测试的命令：**./magick convert input\_file.png /dev/null**，ImageMagick 库则解析完图片之后就退出了。

接下来，我们就开始要对ImageMagick 库进行Fuzzing 了。先从github 上复制最新的代码（建议使用 **git clone** 而不是直接下载源码），命令是：

```
git clone https://github.com/ImageMagick/ImageMagick.git
```

每次ImageMagick 更新代码，只需要在**git pull** 即可。

### ### 3. 如何使用AFL Fuzzing ImageMagick?

AFL 在现在是很流行的Fuzzing 测试工具，关于它的介绍在此不多复述。它的使用方法如下：

- 1.在编译项目时使用**af1-clang** 或**af1-gcc** 等AFL 编译器编译。
- 2.使用**af1-fuzz** 命令对程序进行fuzz,**af1-fuzz** 包含以下参数：

```
af1-fuzz -i %input% -o %output% -t %timeout% -m %memory% fuzz_program  
fuzz_program_arguments
```

**-i %input%**：使用指定目录下的测试样本。 **-o %output%**：把Fuzzing 结果保存到指定目录。 **-t %timeout%**：可选参数,程序运行超时值,以毫秒为单位。 **-m %memory%**：可选参数,最大运行内存,以MB 为单位。 **fuzz\_program fuzz\_program\_arguments**：指定fuzzing 程序和程序运行参数。

对Ubuntu 用户来说,AFL 可以从apt 中直接安装(**sudo apt install afl**)。或者从AFL 的官网下载源码,然后**make install**.关于Fuzzing 的样本下载,MozillaSecurity 团队有一份开源的测试样本 <https://github.com/MozillaSecurity/fuzzdata>

回到ImageMagick 库Fuzzing ,第一步先使用AFL 编译器编译：

```

1  cd ImageMagick
2
3  ./configure CC="afl-clang" CXX="afl-clang++"
4
5  make

```

编译完成之后,接下来使用AFL 对ImageMagick 库进行Fuzzing :

```

1  cd utilities
2
3  mkdir fuzzing_output
4
5  afl-fuzz -i ../../fuzzdata/samples/png -o fuzzing_output -t 300000 -m 200
   ./magick convert @@ /dev/null

```

这句afl-fuzz 命令行的意思是:使用fuzzdata 中的PNG 测试样本作为输入,把Fuzzing 的结果保存到fuzzing\_output 目录中.afl-fuzz 程序运行30 秒超时值是为了让ImageMagick 有更多的时间来执行代码,很有可能Fuzzing 到一些执行很久循环代码,被afl-fuzz 当作超时样本来记录,但这些样本并不是Out-of-Memory 或者CPU exhaustion 类的漏洞.最后利用ImageMagick 的convert 命令来测试ImageMagick 的读操作。

下面是Fuzzing 的情况,在AFL fuzzing 里获得了7 个CVE.

The screenshots show the libFuzzer interface with the following data:

- Process Timing:**
  - run time: 0 days, 17 hrs, 56 min, 26 sec
  - last new path: 0 days, 1 hrs, 16 min, 49 sec
  - last uniq crash: 0 days, 1 hrs, 12 min, 1 sec
  - last uniq hang: 0 days, 5 hrs, 25 min, 27 sec
- Cycle Progress:**
  - now processing: 215% (10.71%)
  - paths timed out: 0 (0.00%)
- Stage Progress:**
  - new trying: bitflip 1/1
  - stage execs: 13.4k/16.4k (81.63%)
  - total execs: 1.67M
  - exec speed: 35.27/sec (slow!)
- Overall Results:**
  - cycles done: 0
  - total paths: 1091
  - uniq crashes: 11
  - uniq hangs: 9
- Map Coverage:**
  - map density: 6971 (10.64%)
  - count coverage: 1.95 bits/tuple
- Findings in Depth:**
  - new edges on: 307 (20.14%)
  - new edges on: 355 (32.54%)
  - total crashes: 786 (11 unique)
  - total hangs: 248 (9 unique)
- Fuzzing Strategy Yields:**
  - bit flips: 207/75.9k, 55/75.9k, 18/75.8k
  - byte flips: 2/9490, 1/4577, 5/4705
  - arithmetic: 98/252k, 7/256k, 4/152k
  - known ints: 4/14.8k, 29/71.3k, 14/147k
  - dictionary: 0/0, 0/0, 40/188k
  - havoc: 265/318k, 0/0
  - trim: 58.74k/5381, 52.01k
- Path Geometry:**
  - levels: 2
  - pending: 1064
  - pend fav: 295
  - own finds: 728
  - imported: n/a
  - variable: 2
- CPU:** 19%

跑AFL 一定需要注意样本,好的样本能够大大提升Fuzzing 效率。

\*参考链接:<http://lcamtuf.coredump.cx/afl/>

## ### 4. AddressSanitizer (ASAN)与崩溃调试

跑到新的崩溃文件时,往往需要进一步定位崩溃点.笔者推荐使用AddressSanitizer ,它是非常强大的内存检测工具,它能够检测不同的内存漏洞以及内存泄漏(内存泄漏也算CVE ,具体请到CVE 官往搜索ImageMagick 的漏洞).我们需要在编译的时候引入ASAN:

```

...
1      cd ImageMagick
2
3      ./configure CC="gcc" CXX="g++" CFLAGS="-g -fsanitize=address"
4
5      make
...

```

在CFLAGS 参数里:-g 指编译的时候添加符号信息,-fsanitize=address 则是让编译器在编译时添加ASAN .然后在utilities 下生成的./magick 程序就带有ASAN 了.运行一个崩溃样本,输出如下:

```

...
1      fuzzing@ubuntu:~/fuzzing/ImageMagick/utilities$ ./magick convert
all_fuzzing_format_2017_7_16_5_13_10/crash/heap-buffer-overflow-READ-
0x7f58970bcdc4_output_ps_1500207243.43 output.ps
2      ==61378==ERROR: AddressSanitizer: heap-buffer-overflow on address
0x7f06a32fedcc at pc 0x7f06aec0f9c2 bp 0x7ffcb67d5a30 sp 0x7ffcb67d5a20
3      READ of size 4 at 0x7f06a32fedcc thread T0
4      #0 0x7f06aec0f9c1 in GetPixelAlpha MagickCore/pixel-accessor.h:59
5      #1 0x7f06aec17ff8 in writePSImage coders/ps.c:2046
6      #2 0x7f06ae7491c6 in writeImage MagickCore/constitute.c:1114
7      #3 0x7f06ae749e42 in writeImages MagickCore/constitute.c:1333
8      #4 0x7f06adf9c3eb in ConvertImageCommand Magickwand/convert.c:3280
9      #5 0x7f06ae094d98 in MagickCommandGenesis Magickwand/mogrify.c:183
10     #6 0x4017f1 in MagickMain utilities/magick.c:149
11     #7 0x4019d2 in main utilities/magick.c:180
12     #8 0x7f06ad80982f in __libc_start_main (/lib/x86_64-linux-
gnu/libc.so.6+0x2082f)
13     #9 0x401308 in _start
(/home/fuzzing/fuzzing/ImageMagick/utilities/.libs/lt-magick+0x401308)
14
15     0x7f06a32fedcc is located 12 bytes to the right of 556480-byte region
[0x7f06a3277000,0x7f06a32fedc0)
16     allocated by thread T0 here:
17     #0 0x7f06af3e5076 in __interceptor_posix_memalign (/usr/lib/x86_64-
linux-gnu/libasan.so.2+0x99076)
18     #1 0x7f06ae8ed8de in AcquireAlignedMemory MagickCore/memory.c:262
19     #2 0x7f06ae6e4731 in OpenPixelCache MagickCore/cache.c:3523
20     #3 0x7f06ae6dd0d1 in GetImagePixelCache MagickCore/cache.c:1667
21     #4 0x7f06ae6ec1f0 in SyncImagePixelCache MagickCore/cache.c:5222
22     #5 0x7f06ae8b9609 in SetImageExtent MagickCore/image.c:2554
23     #6 0x7f06aec53c99 in ReadSGIImage coders/sgi.c:374
24     #7 0x7f06ae746068 in ReadImage MagickCore/constitute.c:497
25     #8 0x7f06ae748267 in ReadImages MagickCore/constitute.c:866
26     #9 0x7f06adf060ad in ConvertImageCommand Magickwand/convert.c:641
27     #10 0x7f06ae094d98 in MagickCommandGenesis Magickwand/mogrify.c:183
28     #11 0x4017f1 in MagickMain utilities/magick.c:149
29     #12 0x4019d2 in main utilities/magick.c:180
30     #13 0x7f06ad80982f in __libc_start_main (/lib/x86_64-linux-
gnu/libc.so.6+0x2082f)
31
32     SUMMARY: AddressSanitizer: heap-buffer-overflow MagickCore/pixel-
accessor.h:59 GetPixelAlpha
33     Shadow bytes around the buggy address:
34     0x0fe154657d60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
35     0x0fe154657d70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
36     0x0fe154657d80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
37     0x0fe154657d90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
38     0x0fe154657da0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
39     =>0x0fe154657db0: 00 00 00 00 00 00 00 00 fa[fa]fa fa fa fa fa fa
40     0x0fe154657dc0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
41     0x0fe154657dd0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa

```

```

42 0x0fe154657de0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
43 0x0fe154657df0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
44 0x0fe154657e00: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
45 Shadow byte legend (one shadow byte represents 8 application bytes):
46 Addressable: 00
47 Partially addressable: 01 02 03 04 05 06 07
48 Heap left redzone: fa
49 Heap right redzone: fb
50 Freed heap region: fd
51 Stack left redzone: f1
52 Stack mid redzone: f2
53 Stack right redzone: f3
54 Stack partial redzone: f4
55 Stack after return: f5
56 Stack use after scope: f8
57 Global redzone: f9
58 Global init order: f6
59 Poisoned by user: f7
60 Container overflow: fc
61 Array cookie: ac
62 Intra object redzone: bb
63 ASan internal: fe
64 ==61378==ABORTING
...

```

ASAN 同样也能用在漏洞挖掘上(比如Heap-overflow READ ,这样的漏洞不容易触发崩溃(除非读到不能够访问的内存上,否则是会产生崩溃的)).但是在AFL 里,不可以在./configure 时把-fsanitize=address 直接添加在CFLAGS 里面,而是需要在make 的时候这样写:

```

...
1  AFL_USE_ASAN=1 make
...

```

把ASAN 作为异常捕获的,还有libFuzzer。

### ### 5. 如何使用libFuzzer Fuzzing ImageMagick?

libFuzzer 与AFL 的区别在于: libFuzzer 倾向于对某个功能或者函数来进行Fuzzing, AFL 则是对整体程序进行Fuzzing。下面是一段使用libFuzzer Fuzzing C-ares DNS 库的示例代码(CVE-2017-1000381):

```

...
1  #include <malloc.h>
2  #include <ares.h>
3
4  extern "C" int LLVMFuzzerTestOneInput(const unsigned char* data,size_t
size) {
5      ares_naptr_reply* naptr_out = 0;
6
7      ares_parse_naptr_reply(data,size,&naptr_out);
8
9      if (0 != naptr_out)
10         free(naptr_out);
11
12  }
...

```

LLVMFuzzerTestOneInput() 函数是libFuzzer 库进行单元测试的调用点,这里面的实现就是我们将要编写的测试用例.libFuzzer 能够更深入去挖掘潜在的漏洞,但是需要对源码有一定的了解.以ImageMagick 为例,convert 命令中的图片最后会被ReadImage() 调用去解析,ReadImage() 函数会根据目录去打开文件再从中读取数据来给指定的格式来解析.下面是针对ImageMagick 的图片解析Fuzzing 代码:

```
...
1      #include <malloc.h>
2      #include <memory.h>
3      #include <stdio.h>
4      #include <stdlib.h>
5
6      #define MAGICK_IMPLEMENTATION
7      #define MagickExport
8
9      #include "magick/common.h"
10     #include "magick/image.h"
11     #include "magick/error.h"
12     #include "magick/constitute.h"
13     #include "magick/magick_types.h"
14     #include "magick/utility.h"
15     #include "magick/magick.h"
16
17
18     static const struct {
19         char
20         *name;
21
22         unsigned char
23         *magic;
24
25         unsigned int
26         length,
27         offset;
28     } StaticMagic[] = {
29     #define MAGIC(name,offset,magic) {name,(unsigned char
*)magic,sizeof(magic)-1,offset}
30     MAGIC("WEBP", 8, "WEBP"),
31     MAGIC("AVI", 0, "RIFF"),
32     MAGIC("8BIMWTEXT", 0, "8\000B\000I\000M\000#"),
33     MAGIC("8BIMTEXT", 0, "8BIM#"),
34     MAGIC("8BIM", 0, "8BIM"),
35     MAGIC("BMP", 0, "BA"),
36     MAGIC("BMP", 0, "BM"),
37     MAGIC("BMP", 0, "CI"),
38     MAGIC("BMP", 0, "CP"),
39     MAGIC("BMP", 0, "IC"),
40     MAGIC("BMP", 0, "PI"),
41     MAGIC("CALS", 21, "version: MIL-STD-1840"),
42     MAGIC("CALS", 0, "srcdocid:"),
43     MAGIC("CALS", 9, "srcdocid:"),
44     MAGIC("CALS", 8, "rorient:"),
45     MAGIC("CGM", 0, "BEGMF"),
46     MAGIC("CIN", 0, "\200\052\137\327"),
47     MAGIC("DCM", 128, "DICM"),
48     MAGIC("DCX", 0, "\261\150\336\72"),
49     MAGIC("DIB", 0, "\050\000"),
50     MAGIC("DOT", 0, "digraph"),
51     MAGIC("DPX", 0, "SDPX"),
52     MAGIC("DPX", 0, "XPDS"),
53     MAGIC("EMF", 40, "\040\105\115\106\000\000\001\000"),
54     MAGIC("EPT", 0, "\305\320\323\306"),
55     MAGIC("FAX", 0, "DFAX"),
56     MAGIC("FIG", 0, "#FIG"),
57     MAGIC("FITS", 0, "IT0"),
```

```
58     MAGIC("FITS", 0, "SIMPLE"),
59     MAGIC("FPX", 0, "\320\317\021\340"),
60     MAGIC("GIF", 0, "GIF8"),
61     MAGIC("HDF", 1, "HDF"),
62     MAGIC("HPGL", 0, "IN;"),
63     MAGIC("HPGL", 0, "\033E\033"),
64     MAGIC("HTML", 1, "HTML"),
65     MAGIC("HTML", 1, "html"),
66     MAGIC("ILBM", 8, "ILBM"),
67     MAGIC("IPTCWTEXT", 0,
"\062\000#\000\060\000=\000\042\000&\000#\000\060\000;\000&\000#\000\062\00
0;\000\042\000"),
68     MAGIC("IPTCTEXT", 0, "2#0=\042&#0;&#2;\042"),
69     MAGIC("IPTC", 0, "\034\002"),
70     MAGIC("JNG", 0, "\213JNG\r\n\032\n"),
71     MAGIC("JPEG", 0, "\377\330\377"),
72     MAGIC("JPC", 0, "\377\117"),
73     MAGIC("JP2", 4, "\152\120\040\040\015"),
74     MAGIC("MAT", 0, "MATLAB 5.0 MAT-file,"),
75     MAGIC("MIFF", 0, "Id=ImageMagick"),
76     MAGIC("MIFF", 0, "id=ImageMagick"),
77     MAGIC("MNG", 0, "\212MNG\r\n\032\n"),
78     MAGIC("MPC", 0, "id=MagickCache"),
79     MAGIC("MPEG", 0, "\000\000\001\263"),
80     MAGIC("PCD", 2048, "PCD_"),
81     MAGIC("PCL", 0, "\033E\033"),
82     MAGIC("PCX", 0, "\012\002"),
83     MAGIC("PCX", 0, "\012\005"),
84     MAGIC("PDB", 60, "vIMGview"),
85     MAGIC("PDF", 0, "%PDF-"),
86     MAGIC("PFA", 0, "%!PS-AdobeFont-1.0"),
87     MAGIC("PFB", 6, "%!PS-AdobeFont-1.0"),
88     MAGIC("PGX", 0, "PG ML"),
89     MAGIC("PGX", 0, "PG LM"),
90     MAGIC("PICT", 522, "\000\021\002\377\014\000"),
91     MAGIC("PNG", 0, "\211PNG\r\n\032\n"),
92     MAGIC("PBM", 0, "P1"),
93     MAGIC("PGM", 0, "P2"),
94     MAGIC("PPM", 0, "P3"),
95     MAGIC("PBM", 0, "P4"),
96     MAGIC("PGM", 0, "P5"),
97     MAGIC("PPM", 0, "P6"),
98     MAGIC("P7", 0, "P7 332"), /* xv Thumbnail */
99     MAGIC("PAM", 0, "P7"), /* Should be listed after "P7 332" */
100    MAGIC("PS", 0, "%!"),
101    MAGIC("PS", 0, "\004%!"),
102    MAGIC("PS", 0, "\305\320\323\306"),
103    MAGIC("PSD", 0, "8BPS"),
104    MAGIC("PWP", 0, "SFW95"),
105    MAGIC("RAD", 0, "#?RADIANCE"),
106    MAGIC("RAD", 0, "VIEW= "),
107    MAGIC("RLE", 0, "\122\314"),
108    MAGIC("SCT", 0, "CT"),
109    MAGIC("SFW", 0, "SFW94"),
110    MAGIC("SGI", 0, "\001\332"),
111    MAGIC("SUN", 0, "\131\246\152\225"),
112    MAGIC("SVG", 1, "?XML"),
113    MAGIC("SVG", 1, "?xml"),
114    MAGIC("TIFF", 0, "\115\115\000\052"),
115    MAGIC("TIFF", 0, "\111\111\052\000"),
116    MAGIC("BIGTIFF", 0, "\115\115\000\053\000\010\000\000"),
117    MAGIC("BIGTIFF", 0, "\111\111\053\000\010\000\000\000"),
118    MAGIC("VICAR", 0, "LBLSIZE"),
119    MAGIC("VICAR", 0, "NJPL1I"),
120    MAGIC("VIFF", 0, "\253\001"),
```

```

121     MAGIC("WMF", 0, "\\327\\315\\306\\232"),
122     MAGIC("WMF", 0, "\\001\\000\\011\\000"),
123     MAGIC("WPG", 0, "\\377WPC"),
124     MAGIC("XBM", 0, "#define"),
125     MAGIC("XCF", 0, "gimp xcf"),
126     MAGIC("XPM", 1, "* XPM *"),
127     MAGIC("XWD", 4, "\\007\\000\\000"),
128     MAGIC("XWD", 5, "\\000\\000\\007")
129 };
130
131     int random(const unsigned char* data,unsigned int size) {
132         unsigned int random_code = 0;
133
134         for (unsigned int data_index = 0;data_index < size;++data_index)
135             random_code += data[data_index];
136
137         return random_code % (sizeof(StaticMagic) /sizeof(StaticMagic[0]));
138     }
139
140     #define GENERATE_FILE_NAME "./libFuzzerGenarateImageSample"
141
142     extern "C" int LLVMFuzzerTestOneInput(const unsigned char*
data,unsigned int size) {
143         int random_image_flag_index = random(data,size);
144         unsigned int random_image_flag_offset =
StaticMagic[random_image_flag_index].offset;
145         unsigned int random_image_flag_length =
StaticMagic[random_image_flag_index].length;
146         unsigned int image_buffer_length = random_image_flag_offset +
random_image_flag_length + size;
147         unsigned char* image_buffer = (unsigned
char*)malloc(image_buffer_length);
148
149         memset(image_buffer,0,image_buffer_length);
150
151         memcpy(image_buffer,StaticMagic[random_image_flag_index].name,StaticMagic[r
andom_image_flag_index].length);
152
153         FILE* file = fopen(GENERATE_FILE_NAME,"w");
154
155         if (NULL != file) {
156             fwrite(image_buffer,1,image_buffer_length,file);
157             fclose(file);
158
159             printf("buffer=%s(0x%X), size=%d,input
format=%s\n",image_buffer,image_buffer_length,StaticMagic[rando
m_image_flag_index].name);
160
161             ExceptionInfo exception;
162             ImageInfo* read_image_info;
163             ImageInfo* write_image_info;
164             Image* image;
165
166             GetExceptionInfo(&exception);
167
168             read_image_info = CloneImageInfo((ImageInfo*)NULL);
169             write_image_info = CloneImageInfo((ImageInfo*)NULL);
170
171             strcpy(read_image_info->filename,GENERATE_FILE_NAME,MaxTextExtent);
172             strcpy(write_image_info->filename,"/dev/null",MaxTextExtent);
173             SetImageInfo(read_image_info,SETMAGICK_READ,&exception);
174             SetImageInfo(write_image_info,SETMAGICK_WRITE,&exception);
175
176             image = ReadImage(read_image_info,&exception);

```



```

176
177         if (NULL != image)
178             writeImage(write_image_info,image);
179
180         DestroyImageInfo(read_image_info);
181         DestroyImageInfo(write_image_info);
182         DestroyExceptionInfo(&exception);
183     }
184
185     free(image_buffer);
186
187     return 0;
188 }
189
190 extern "C" int LLVMFuzzerInitialize(int *argc, char ***argv) {
191     InitializeMagick((const char*)argv[0]);
192
193     return 1;
194 }
195
...

```

使用libFuzzer Fuzzing 时,编译时一定要添加ASAN :

```

...
1     cd ImageMagick
2
3     ./configure CC="clang" CXX="clang++" CFLAGS="-O2 -g -fsanitize-
coverage=trace-pc-guard -fsanitize=address"
4
5     make
6
7     cd utilities
8
9     clang++ read_image_fuzzer.cc -O2 -g -fsanitize-coverage=trace-pc-guard -
fsanitize=address -I .. -ljpeg -lwebp -llcms2 -ltiff -lfreetype -ljpeg -
lpng12 -lwmflite -lXext -lSM -lICE -lX11 -llzma -lbz2 -lxml2 -lz -lm -lgomp -
lpthread ../magick/.libs/libMagickCore-7.Q16HDRI.a
    ../../libFuzzer/Fuzzer/libFuzzer.a -o read_image_fuzzer
...

```

第一次编译ImageMagick 时,一定要把ASAN 也编译进去,否则在ImageMagick 库里面跑出来漏洞 libFuzzer 不会提示!接下来编译我们的测试Fuzzer ,注意需要这两次编译都要用到clang 5 ,因为插件-fsanitize-coverage=trace-pc-guard 需要在高版本的clang 里才有,否则libFuzzer 会没有数据输入(笔者也是在读libFuzzer 的源码里发现的),后面需要引用的库则根据自己的系统来添加,会有些差异。

关于更多libFuzzer 的知识,可以参考这两个链接:

1、<https://github.com/Dor1s/libfuzzer-workshop>

2、<http://llvm.org/docs/LibFuzzer.html>

### ### 6. 程序模型与代码覆盖

程序模型简单地来说可以分为三部分:输入,处理,输出.无论是WEB 还是二进制都一样成立,漏洞会隐藏在这三个阶段的某个角落里.举个例子,输入阶段有:ETERNALBLUE ,S2-045 (上传组件的异常信息中可以执行OGNL 导致GetShell ),CVE-2017-7269—IIS 6.0 WebDAV ;处理阶段:各大SRC 里的业务逻辑漏洞,BadKernel ;输出阶段:XSS 漏洞.Fuzzing 的时候也需要考虑到这点,尽可能让程序所有的阶段都能够执行,达到更高的代码覆盖率



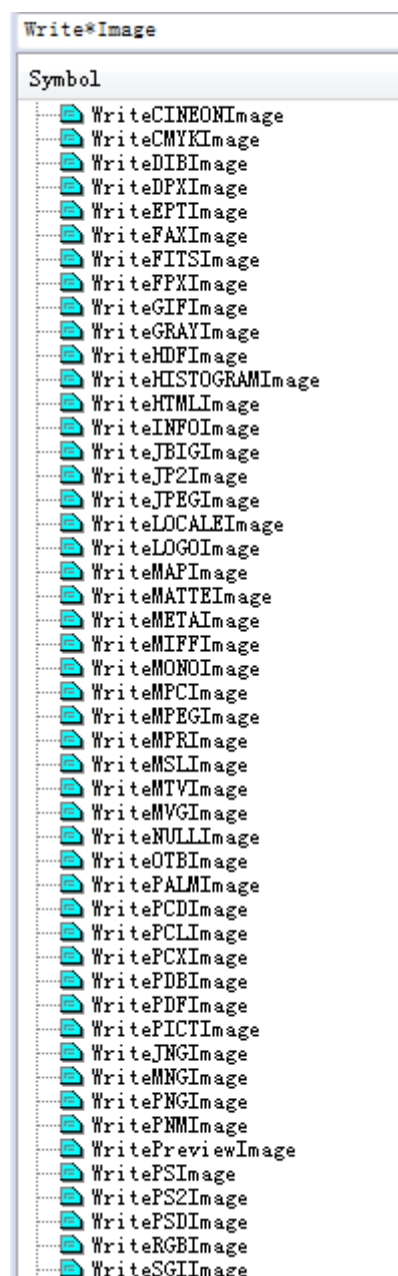
在前面AFL 与libFuzzer 的例子中,我们暂时只Fuzzing ImageMagick 读取图片部分(对应到代码的 ReadXXXImage() 函数),接下来我们要针对ImageMagick 的写图片部分进行Fuzzing(对应到代码的 WriteXXXImage() 函数)

### ### 7. 使用Python 实现ImageMagick Fuzzer

我们自己在实现Fuzzer 的时候,如何准确地检测触发的漏洞点是一个难题,现在已经有ASAN 作为强力的 Bug 检测工具,我们只需要针对Fuzzing 的逻辑来编写即可.前面提到,ImageMagick 可以实现不同图像格式之间的转换,所以在Fuzzing 时要触发到WriteXXXImage() 这类函数,必须要以指定的格式来输出.比如说触发WritePNGImage() 函数,magick 就必须输出.png 格式图像;触发WriteARTImage() 函数,magick 就必须输出.art 格式图像,命令如下:

```
1 ./magick convert test_image output.png
2 ./magick convert test_image output.art
```

ImageMagick 库支持很多格式的图像转换,更多信息可以在SourceInsight 里搜索Write\*Image() 函数:



最后得到支持输出格式的列表:

```

...
1     imagemagick_output_format =
    ['output.aai', 'output.art', 'output.avs', 'output.bgr', 'output.bmp', 'output.bra
    ille', 'output.cals', 'output.cin', 'output.cip', 'output.clipboard', 'output.cmyk
    ', 'output.dds', 'output.debug', 'output.dib', 'output.dpx', 'output.ept', 'output.
    exr', 'output.fax', 'output.fits', 'output.flif', 'output.fpx', 'output.gif', 'outp
    ut.gray', 'output.histogram', 'output.hrz', 'output.html', 'output.icon', 'output.
    info', 'output.inline', 'output.ipl', 'output.jbig', 'output.jp2', 'output.jpeg', '
    output.json', 'output.magick', 'output.map', 'output.mask', 'output.mat', 'output.
    matte', 'output.meta', 'output.miff', 'output.mono', 'output.mpc', 'output.mpeg', '
    output.mpr', 'output.msl', 'output.mtv', 'output.mvg', 'output.null', 'output.otb'
    ', 'output.palm', 'output.pcd', 'output.pcl', 'output.pcx', 'output.pdb', 'output.pd
    f', 'output.pgx', 'output.pict', 'output.jng', 'output.mng', 'output.png', 'output.
    pnm', 'output.ps', 'output.ps2', 'output.ps3', 'output.psd', 'output.raw', 'output.
    rgb', 'output.rgf', 'output.sgi', 'output.sixel', 'output.sun', 'output.svg', 'out
    put.tga', 'output.thumbnail', 'output.ptif', 'output.tiff', 'output.txt', 'output.
    uil', 'output.uvyv', 'output.vicar', 'output.vid', 'output.viff', 'output.vips', 'o
    utput.wbmp', 'output.webp', 'output.xbm', 'output.picon', 'output.xpm', 'output.xt
    rn', 'output.xwd', 'output.ycbr', 'output.ps3mask', 'output.group4', 'output.yuv'
    'output.x']
...

```

然后给ImageMagick 来运行即可,代码如下:

```

...
1     def run_imagemagick_convert(input_file,output_file) :
2         process =
        subprocess.Popen(['./magick','convert',input_file,output_file],stdout =
        subprocess.PIPE,stderr = subprocess.PIPE) # 使用样本运行magick convert
3         process_timeout_exit = lambda : process.kill()
4         timeout =
        threading.Timer(MAX_PROCESS_WAIT_TIME,process_timeout_exit) # 执行超时退出
5
6         timeout.start()
7         process.wait()
8         timeout.cancel()
9
10        std_error_output = process.stderr.read() # 从stderr 中读取ASAN 的崩
        溃信息输出
11
12        if len(std_error_output) and not -1 ==
        std_error_output.find('====='): # ASAN Check ..
13            flag_address_sanitize = 'ERROR: AddressSanitizer: '
14            flag_address_sanitize_offset =
            std_error_output.find(flag_address_sanitize)
15            flag_leak_sanitize = 'ERROR: LeakSanitizer: '
16            flag_leak_sanitize_offset =
            std_error_output.find(flag_leak_sanitize)
17
18            if not -1 == flag_address_sanitize_offset : # 内存漏洞分析
19                crash_type = std_error_output[flag_address_sanitize_offset +
                len(flag_address_sanitize) : ]
20                crash_type = crash_type[ : crash_type.find('on') ].strip()
21                crash_type_detail = ''
22
23                flag_at_pc = 'pc '
24                flag_at_pc_offset = std_error_output.find(flag_at_pc)
25
26                crash_point = std_error_output[flag_at_pc_offset +
                len(flag_at_pc) : ]
27                crash_point = crash_point[ : crash_point.find(' bp')]
28

```

```

29         if not -1 == crash_type.find('buffer-overflow') : # stack
and heap
30             flag_of_size = ' of size'
31             flag_of_size_offset =
std_error_output.find(flag_of_size)
32
33             crash_type_detail = std_error_output[ :
flag_of_size_offset]
34             crash_type_detail =
crash_type_detail[crash_type_detail.rfind('\n') : ].strip()
35             elif 'SEGV' == crash_type : # Null point reference ..
36                 flag_of_address = 'unknown address'
37                 flag_of_address_offset =
std_error_output.find(flag_of_address)
38
39                 crash_type_detail =
std_error_output[flag_of_address_offset + len(flag_of_address) : ]
40                 crash_type_detail = crash_type_detail[ :
crash_type_detail.find('(')].strip()
41
42             return
True,crash_type,crash_type_detail,crash_point,std_error_output
43             elif not -1 == flag_leak_sanitiz_offset : # 内存泄漏分析
44                 flag_direct_leak = 'Direct leak'
45                 flag_indirect_leak = 'Indirect leak'
46                 memory_leak_id = std_error_output.count(flag_direct_leak) +
std_error_output.count(flag_indirect_leak)
47
48                 return True , 'Memory-Leak' , str(memory_leak_id) , None ,
std_error_output
49
50                 return True , None , None , None , std_error_output
51
52                 return False , None , None , None , None
53
54             # 省略无关代码
55
56             for graphicsmagick_input_file_index in
graphicsmagick_input_file_list : # 样本列表
57                 for graphicsmagick_output_format_index in
imagemagick_output_format : # 测试输出格式列表
58                     graphicsmagick_output = crash_dir_crash_output_dir + '/' +
graphicsmagick_input_file_index + '_' + graphicsmagick_output_format_index
59
60                     result =
run_graphicsmagick_convert(graphicsmagick_input_file_index_path,graphicsmagi
ck_output) # 调用magick convert
61
...

```

还需要注意的一点是,ImageMagick 会在/tmp 目录下生成临时文件而且它不会去删除这些文件,所以还需要去手动删除:

```

...
1     import os
2     import time
3
4
5     if __name__ == '__main__' :
6         while True :
7             file_list = os.listdir('/tmp')
8
9             for file_index in file_list :

```

```

10         if file_index.startswith('magick') :
11             try :
12                 os.remove('/tmp/' + file_index)
13             except :
14                 pass
15
16
17     time.sleep(10)
18
19

```

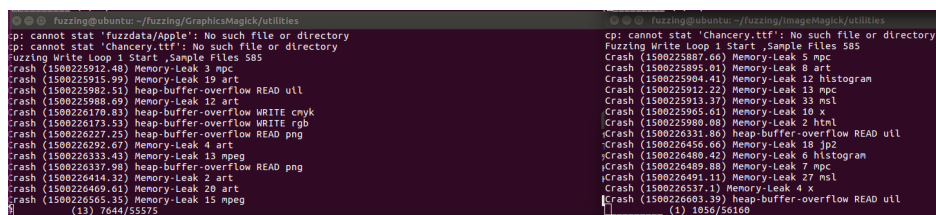
上面的Fuzzer 需要依赖样本,我们也可以使用Python 像libFuzzer 那样生成数据到给ImageMagick 测试:

```

...
1  def build_random_data(data_length) : # 随机生成指定长度的数据
2      data = b''
3
4      for data_index in range(data_length) :
5          data += chr(random.randrange(255))
6
7      return data
8
9  def build_random_image(data_length) : # 生成测试图像
10     random_image_header = random.choice(image_header) # image_header
    和libFuzzer 里的ImageMagick 头一样
11
12     return build_random_data(random_image_header[1]) +
    random_image_header[2] + build_random_data(data_length)
13
14     # 省去多余代码
15
16     while True :
17         random_image = build_random_image(max_fuzzing_image_length)
18
19         write_file('fuzzing_generate_image', random_image)
20
21         for graphicsmagick_output_format_index in imagemagick_output_format
    :
22             result =
    run_graphicsmagick_convert('fuzzing_generate_image', fuzzing_dir + '/'
    +graphicsmagick_output_format_index)
23
24             if result[0] : # catch crash
25                 ...
26
27

```

在此限于篇幅,所有Fuzzing 的完整代码在我的github .执行效果如下:



















```

fuzzing@ubuntu:~/fuzzing/graphicsmagick/utilities$ ./fuzzing_generate_image.py
cp: cannot stat 'fuzzdata/apple': No such file or directory
cp: cannot stat 'chancery.ttf': No such file or directory
Fuzzing Write Loop 1 Start ,sample Files 585
Crash (1500225912.40) Memory-Leak 3 mpc
Crash (1500225915.99) Memory-Leak 19 art
Crash (1500225982.51) heap-buffer-overflow READ u11
Crash (1500225988.60) Memory-Leak 12 art
Crash (1500226170.83) heap-buffer-overflow WRITE cnk
Crash (1500226227.25) heap-buffer-overflow WRITE rgb
Crash (1500226416.35) Memory-Leak 2 art
Crash (1500226469.61) Memory-Leak 20 art
Crash (1500226565.35) Memory-Leak 15 mpeg
(13) 7644/55575











fuzzing@ubuntu:~/fuzzing/graphicsmagick/utilities$ ./fuzzing_generate_image.py
cp: cannot stat 'chancery.ttf': No such file or directory
Fuzzing Write Loop 1 Start ,sample Files 585
Crash (1500225887.66) Memory-Leak 5 npc
Crash (1500225895.61) Memory-Leak 8 art
Crash (1500225904.41) Memory-Leak 12 histogram
Crash (1500225912.22) Memory-Leak 13 mpc
Crash (1500225913.37) Memory-Leak 33 nsl
Crash (1500225965.61) Memory-Leak 10 x
Crash (1500225980.08) Memory-Leak 2 html
Crash (1500226331.86) heap-buffer-overflow READ u11
Crash (1500226456.66) Memory-Leak 18 jp2
Crash (1500226480.42) Memory-Leak 6 histogram
Crash (1500226489.88) Memory-Leak 7 mpc
Crash (1500226491.11) Memory-Leak 27 nsl
Crash (1500226537.1) Memory-Leak 4 x
Crash (1500226603.39) heap-buffer-overflow READ u11
(1) 1056/56168

```

笔者已经使用这个Python Fuzzer 收获了至少20 个CVE (数据生成挖到了3 个崩溃,利用样本测试 ImageMagick 写测试至少获得了17 个崩溃),在此通过这篇文章和各位读者分析对于二进制软件的 Fuzzing 的一些常用技术,预祝大家顺利挖到CVE .下面是挖掘到的漏洞列表:

 FPE--0x7eff23c45e38_output_palm_1500208096.66	2017/7/16 21:35	66 文件	832 KB
 heap-buffer-overflow-READ-0x7f58970bcd4_output_ps_1500207243.43	2017/7/16 21:34	43 文件	17 KB
 heap-buffer-overflow-READ-0x7fd806e82db2_output_uil_1500210468.72	2017/7/16 21:33	72 文件	2 KB
 heap-overflow_output_picon_READ_GetPixelIndex	2017/7/18 10:50	文件	5 KB
 imagemagick_output_mpc_memory_leak_WriteMPCImage	2017/7/16 23:18	文件	26 KB
 memory-leak_output_art_lite_font_map	2017/7/17 2:21	文件	5 KB
 memory-leak_output_art_ReadOnePNGImage	2017/7/18 12:46	文件	1 KB
 memory-leak_output_histogram_ReadTTFImage	2017/7/17 1:59	文件	2 KB
 memory-leak_output_histogram_WriteHISTOGRAMImage	2017/7/17 2:22	文件	5 KB
 memory-leak_output_jp2_WriteJP2Image	2017/7/17 2:06	文件	59 KB
 memory-leak_output_mpc_ReadOnePNGImage	2017/7/17 2:04	文件	6 KB
 memory-leak_output_msl_ReadPWPIImage	2017/7/17 1:04	文件	5 KB
 memory-leak_output_png_WriteOnePNGImage	2017/7/17 2:01	文件	59 KB
 memory-leak_output_x_WriteXImage	2017/7/17 1:57	文件	2 KB
 memory-leak_ReadMATImage	2017/7/19 0:47	文件	1 KB
 SEGV_output_ptif_WritePTIFImage	2017/7/19 0:36	文件	5 KB

 gm_1.3.26_convert_output_cmyk_ExportCMYKQuantumType_heap-overflow	2017/7/18 22:14	26_CONVERT_O...	325 KB
 gm_1.3.26_convert_output_map_SEGV-0x000000000000	2017/7/18 22:13	26_CONVERT_O...	35 KB
 gm_1.3.26_convert_output_map_WriteMAPImage_null_point_reference	2017/7/18 22:14	26_CONVERT_O...	103 KB
 gm_1.3.26_convert_output_mpc_memory_leak	2017/7/18 22:13	26_CONVERT_O...	4 KB
 gm_1.3.26_convert_output_mpeg_memory_leak	2017/7/18 22:13	26_CONVERT_O...	305 KB
 gm_1.3.26_convert_output_pcl_WritePCLImage_null_point_reference	2017/7/18 22:14	26_CONVERT_O...	4 KB
 gm_1.3.26_convert_output_png_WriteOnePNGImage_heap-overflow	2017/7/18 22:14	26_CONVERT_O...	1 KB
 gm_1.3.26_convert_output_png_WriteOnePNGImage_stack-overflow	2017/7/18 22:13	26_CONVERT_O...	2 KB
 gm_1.3.26_convert_output_rgb_heap-buffer-overflow	2017/7/18 22:13	26_CONVERT_O...	325 KB
 gm_1.3.26_convert_output_uil_WriteUILImage_heap-overflow	2017/7/18 22:14	26_CONVERT_O...	2 KB