二进制安全

从CTF到漏洞挖掘

简介

- 本议题是分享本人在二进制安全学习路上的见与闻，由于本人水平浅薄，因此存在一定的含水量
- CTF部分内容主要是：
  - 什么是CTF？
  - 为什么我选择从CTF入门二进制安全？
  - CTF PWN方向该怎么学？
  - CTF的现状
- 漏洞挖掘部分内容主要是：
  - 二进制漏洞该怎么挖？
  - 什么是模糊测试？
  - 如何对开源项目进行模糊测试？
  - 如何对闭源项目进行模糊测试？

关键词

萌新的、入门的、小白的、简单的、初级的

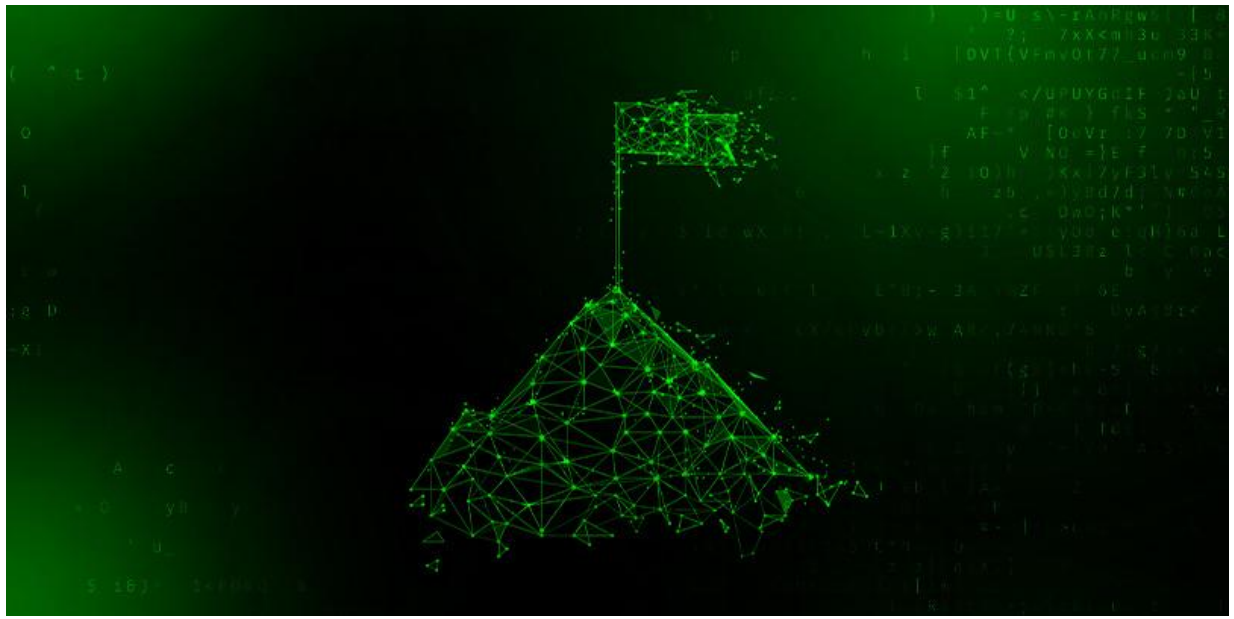
----- 毕竟难的我也不会QwQ -----

----- 此外：**本次分享以学习研究为目的** -----

CTF篇

什么是CTF?

CTF是一种流行的信息安全竞赛形式，其英文名可直译为“夺得Flag”，也可意译为“夺旗赛”。其大致流程是，参赛团队之间通过进行攻防对抗、程序分析等形式，率先从主办方给出的比赛环境中得到一串具有一定格式的字符串或其他内容，并将其提交给主办方，从而夺得分数。为了方便称呼，我们把这样的内容称之为“Flag”。



CTF篇

- 为什么我选择从CTF入门二进制安全？
  - CTF的知识体系较为明确。这点得益于CTF有浓厚的开源氛围，有类似CTF-Wiki这样的项目。
  - CTF赛题众多，初级和中级题目存在详细的题解，拜读不同的WP可以从同一题目获取不同思路。
  - 做出一道CTF题目可以让初学者有满满的成就感
  - 除了上述这些理由，还有很多别的，但这不是重点~~

CTF-PWN篇

- CTF PWN方向该怎么学?
- 什么是PWN?
  - 首先我们要了解软件安全:

    软件安全：软件安全专注于研究软件的设计和实现的安全。

    研究对象：代码（源码、字节码、汇编等）。

    研究目标：发掘漏洞、利用漏洞、修补漏洞。

    研究技术：逆向工程、漏洞挖掘与利用、漏洞防御技术。
  - 那么可以类比引出Pwn:

    软件安全研究的一个缩影。

    研究对象：可执行文件，主要是ELF文件（linux下）

    研究目标：拿到flag/shell

身为二进制安全相关模块，偏向于底层方向，学习曲线陡峭，要求学习者对操作系统的理解更加深入。

CTF-PWN篇

- PWN的特点?

  关键词：痛苦、坐牢、折磨

  - 刚入门的PWN选手，最基础的PWN题目可能会让你想不明白 -----> 放弃
  - 刚入门的PWN选手，做进阶的PWN题目可能会让你眩晕头疼 -----> 放弃
  - 有一定水平的PWN选手，比较难的PWN题目需要大量的耐心 -----> 放弃

因此，学PWN需要耐心

CTF-PWN篇

- CTF-PWN中漏洞利用技巧该怎么学习？

  对于**初学者**来说CTF-PWN主要是linux用户态下的栈溢出利用和堆漏洞利用两大类。

CTF-PWN篇

- 栈溢出利用技巧：

关键词：**套路**

- 学习CTFwiki中的所有技巧：
  - https://ctf-wiki.org/pwn/linux/user-mode/stackoverflow/x86/stack-intro/

  （但这是不够的，毕竟安全的本质是对抗😈，题目不会老考老掉牙的考点）
- 收集近期比赛，查看比赛中刁钻的题目，查看wp来学习trick：
  - **magic read**实现栈迁移：https://blog.csdn.net/fzucaicai/article/details/130735983
  - **magic gadget**无泄漏任意地址写：https://fmyy.pro/2020/04/26/Magic/Magic-Gadget/
  - 一次**无泄漏**gets的极端利用：https://www.roderickchan.cn/2022-sekaictf-pwn-wp-gets-bfs/
  - 等等
- 自己根据已有知识研究出极端条件下通用的trick（重要）
  - 在这不过多讲解

CTF-PWN篇

- 堆漏洞利用技巧：

关键词：**套路**、要过气了

在过去，堆漏洞利用技巧是一场CTF比赛的关键题目，它能衡量一个人是否在CTF-PWN的水平，但是随着比赛不断内卷，如今CTF比赛在这方面考的越来越少。

- 学习**How2heap**中的所有技巧：
  - https://github.com/shellphish/how2heap
- 学习**IO_file**结构体的利用：
  - 主要学house of apple2、house of cat、house of Lys这些只需要一次任意地址写就能劫持程序流的利用。
- 自己根据已有知识研究出极端条件下通用的trick（重要）
  - 在这不过多讲解

CTF篇

- 我眼中的CTF现状

现状：随着本人的比赛年龄逐渐上升，本人发现目前CTF的题目越来越接近实战，比如会直接拿pwn2own往年的披露来出题。

- 2024-AliyunCTF-netatalk：

  https://research.nccgroup.com/2022/03/24/remote-code-execution-on-western-digital-pr4100-nas-cve-2022-23121/

- 2024-realworld-忘了什么题目：

  一个摄像头的设备

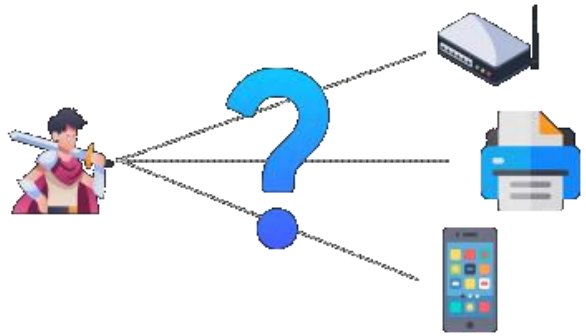
  https://teamt5.org/tw/posts/teamt5-pwn2own-contest-experience-sharing-and-vulnerability-demonstration/

可以看出上面两个题目都是今年的比赛，而且都是pwn2own披露的漏洞🤔

因此，可以简单的得出结论：目前CTF的题目越来越接近实战，主要考察选手对现实漏洞的信息收集与利用，而不是对模板的记忆与利用。

漏洞挖掘篇

- 二进制漏洞该怎么挖？
- 首先要进行的目标的选择，我们知道：IOT设备、Windows端、Linux端等环境中出现的**程序**内存漏洞都是二进制漏洞。下面简单介绍它们主要不同的特点：



1、IOT设备：

- 缺点：
  - IOT设备的固件主要是异构固件，主流反汇编软件在反汇编固件时可能存在问题、也可能难以搜到IOT设备的有关资料。
  - 获取了IOT设备的固件可能**难以在本地仿真进行调试**。
  - IOT设备的固件难以获取，需要有一定的渠道和人脉或者启动资金。
- 好处：
  - IOT设备的运行环境往往没有很多漏洞保护措施，如果发现漏洞，那么进行利用也相对容易。

漏洞挖掘篇

2、Linux端：

- 缺点：
  - 由于大多数程序是开源的，难挖出心仪的漏洞（太卷了！）
- 好处：
  - 大部分可以在linux端安装的程序都是开源的，我们可以在源码中遨游，审计或者模糊测试出漏洞。

3、Winodw端：

- 缺点：
  - 大多数程序都是闭源的，逆向分析的过程会十分痛苦。
- 好处：
  - 由于大多数程序都是闭源的，可以挖出较多的漏洞（没那没卷！）

漏洞挖掘篇

漏洞挖掘方法：

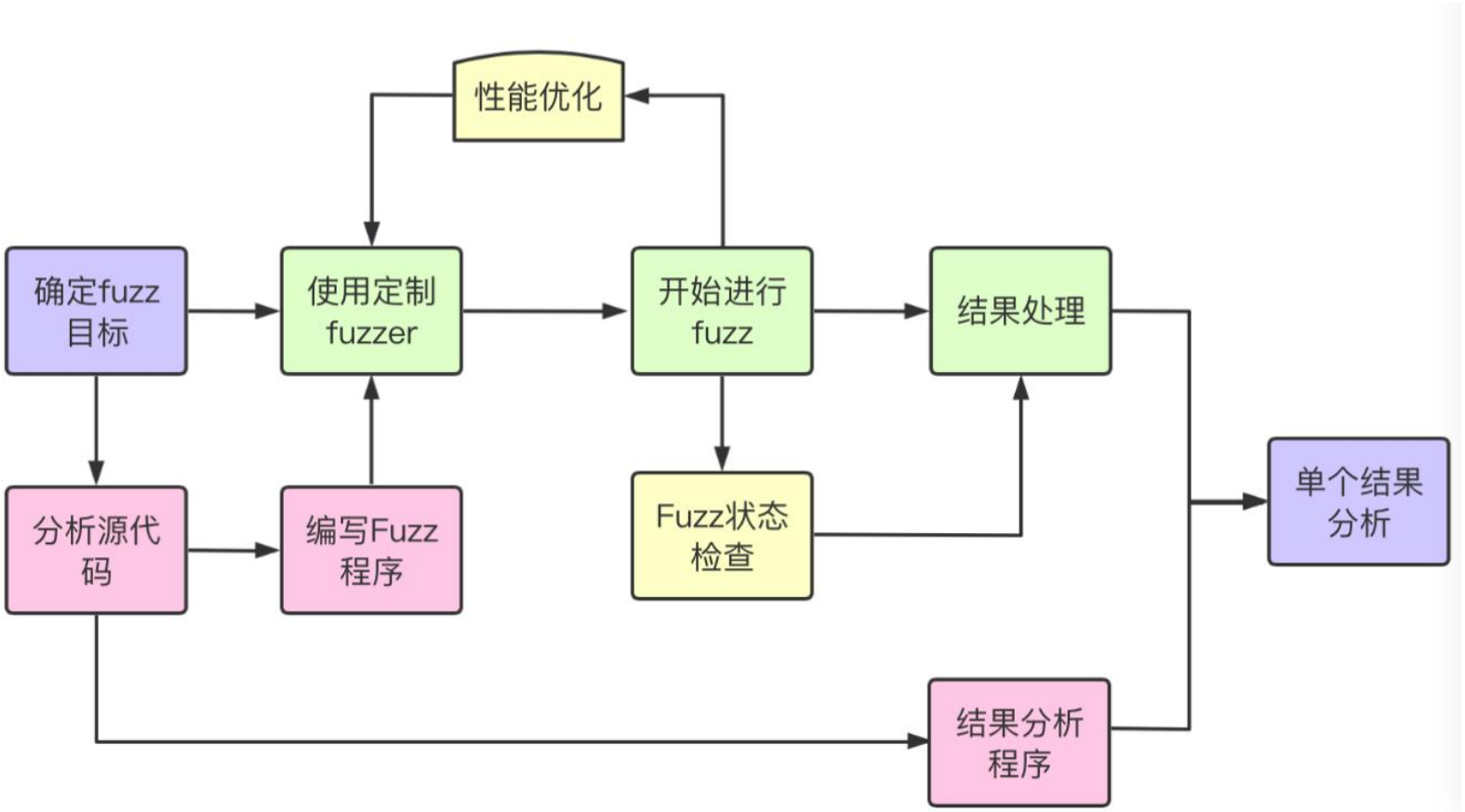
- 源代码审计
  - 需要源代码、需要丰富的经验。
- 逆向工程
  - 需要丰富的经验和深厚的逆向功底。
- Fuzzing（模糊测试）
  - 需要对目标程序攻击面有充足的理解。

An IDA a day keeps the girls away

在此，我将分享我**利用模糊测试技术**挖到开源、闭源程序漏洞的经历。

漏洞挖掘篇

- 什么是模糊测试？



漏洞挖掘篇

- 如何对开源项目进行模糊测试?
- 在这里举一个利用AFL对gpac这个音视频转换程序进行fuzz的过程:
  - 预处理: 安装AFL
  - 编译:

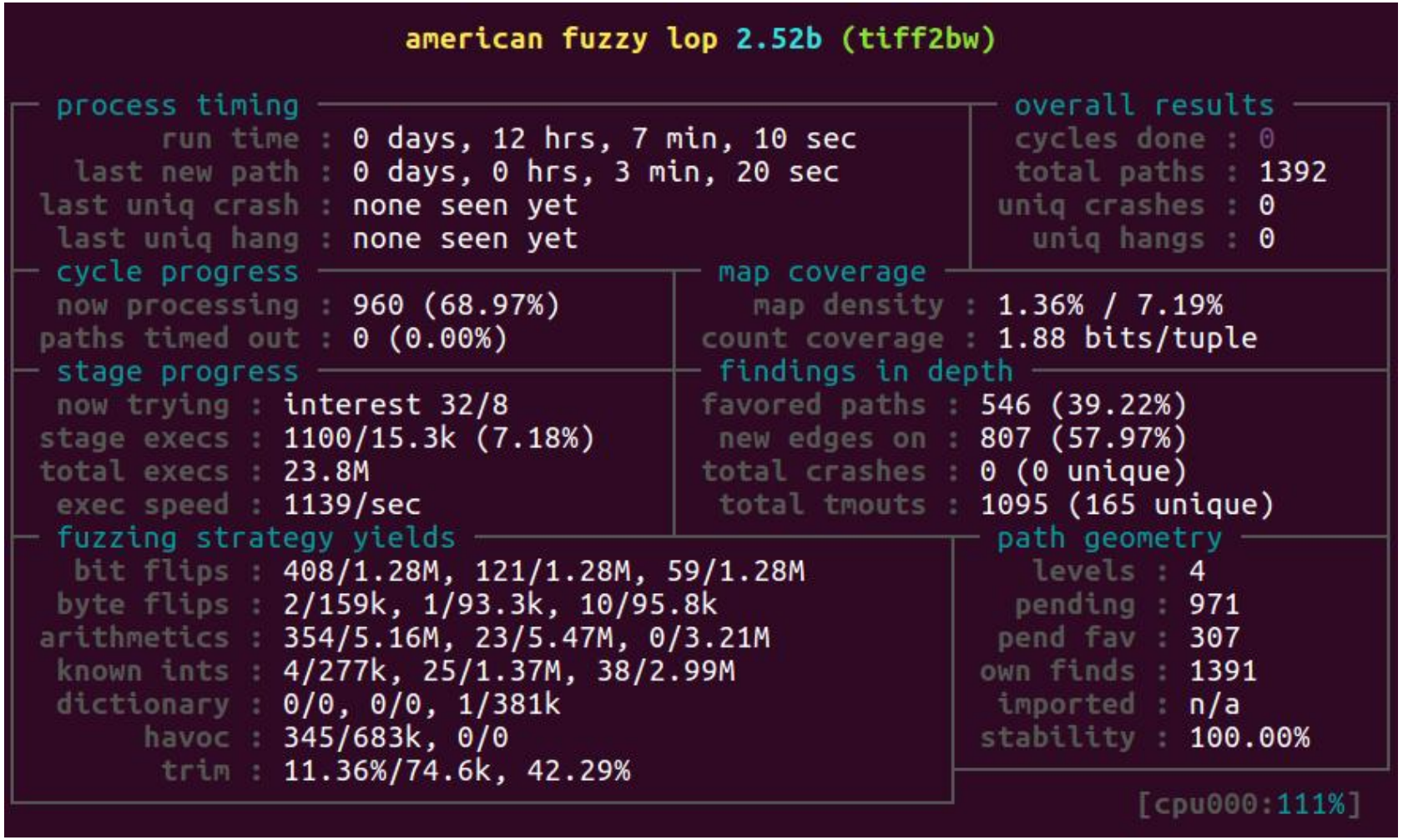
```
export AR=llvm-ar RANLIB=llvm-ranlib AS=llvm-as CC=afl-clang-lto CXX=afl-clang-lto++;
./configure ;
make
```

  - 进行模糊测试:

```
afl-fuzz -m none -i in_seed -o out -D ./MP4Box -info @@
```

  - 等待Crash

漏洞挖掘篇


american fuzzy lop 2.52b (tiff2bw)
process timing
run time : 0 days, 12 hrs, 7 min, 10 sec
last new path : 0 days, 0 hrs, 3 min, 20 sec
last uniq crash : none seen yet
last uniq hang : none seen yet
overall results
cycles done : 0
total paths : 1392
uniq crashes : 0
uniq hangs : 0
cycle progress
now processing : 960 (68.97%)
paths timed out : 0 (0.00%)
map coverage
map density : 1.36% / 7.19%
count coverage : 1.88 bits/tuple
stage progress
now trying : interest 32/8
stage execs : 1100/15.3k (7.18%)
total execs : 23.8M
exec speed : 1139/sec
findings in depth
favored paths : 546 (39.22%)
new edges on : 807 (57.97%)
total crashes : 0 (0 unique)
total tmouts : 1095 (165 unique)
fuzzing strategy yields
bit flips : 408/1.28M, 121/1.28M, 59/1.28M
byte flips : 2/159k, 1/93.3k, 10/95.8k
arithmetics : 354/5.16M, 23/5.47M, 0/3.21M
known ints : 4/277k, 25/1.37M, 38/2.99M
dictionary : 0/0, 0/0, 1/381k
havoc : 345/683k, 0/0
trim : 11.36%/74.6k, 42.29%
path geometry
levels : 4
pending : 971
pend fav : 307
own finds : 1391
imported : n/a
stability : 100.00%
[cpu000:111%]

漏洞挖掘篇

- 如何对闭源项目进行模糊测试?

  本次分享主要参考这篇优秀的文章：https://signal-labs.com/fuzzing-wechats-wxam-parser/

  我们可以按以下几步走，实现对闭源程序进行fuzz：

  - 0、预处理：安装Jackalope：https://cloud.tencent.com/developer/article/2316144
  - 1、逆向分析，获取要进行fuzz的函数以及它的参数意义

```
char __fastcall wxam_decoder_helper_DecodeWxamToJpeg(_DWORD *a1, int a2)
{
  int savedregs; // [esp+0h] [ebp+0h] BYREF

  sub_107A73F0(127, "WxAMDecoderHelper::DecodeWxamToJpeg", "WxAMDecoderHelper", "DecodeWxamToJpeg");
  return wxam_decoder_helper_DecodeWxam(a1, (int)&savedregs, (int **)a2);
}
```

漏洞挖掘篇

- 2、在vs2019中编写harness，也就是自己写一个程序引入要fuzz的函数。

```
void init() {
        // Load target DLLs
        hDll_voip = LoadLibraryA("voipEngine.dll");
        if (hDll_voip == NULL) {
                printf("Failed to load voipEngine.dll\n");
                exit(1);
        }
        hDll_win = LoadLibraryA("WeChatWin.dll");
        if (hDll_win == NULL) {
                printf("Failed to load WeChatWin.dll, err:%x\n", GetLastError());
                exit(1);
        }
        // Create function pointers
        isWxGF = (_isWxGF)GetProcAddress(hDll_voip, "isWxGF");
        wxam_decoder_helper_DecodeWxamToJpeg = (_wxam_decoder_helper_DecodeWxamToJpeg)((unsigned int)hDll_win + 0x7D3DA0);
}
```

漏洞挖掘篇

```
#pragma optimize( "", off )
extern "C"
__declspec(dllexport)
bool fuzz() {

        uint32_t sample_size = 0;
        sample_size = *(uint32_t*)(shm_data);
        if (sample_size > MAX_SAMPLE_SIZE) sample_size = MAX_SAMPLE_SIZE-4;
        // Update the bytes of our buffer
        memcpy(sample_bytes, shm_data + sizeof(uint32_t), sample_size);
        // Input struct is actually:
        // 0: pointer to input buffer
        // 1: input buffer size
        // so, lets create that:
        InputStruct inputStruct;
        inputStruct.inputBuf = sample_bytes;
        inputStruct.inputBuf_sz = sample_size;



        // Call first function as per wxam
        bool res = isWxGF(inputStruct.inputBuf, sample_size);
        // 1 = error, 0 = success
        if (res) {
                //printf("isWxGF failed\n");
                return false;
        }
        int pOut = 0;
        res = wxam_decoder_helper_DecodeWxamToJpeg(&inputStruct, &pOut);
        // 1 = success, 0 = error
        return res;


}
#pragma optimize( "", on )
```

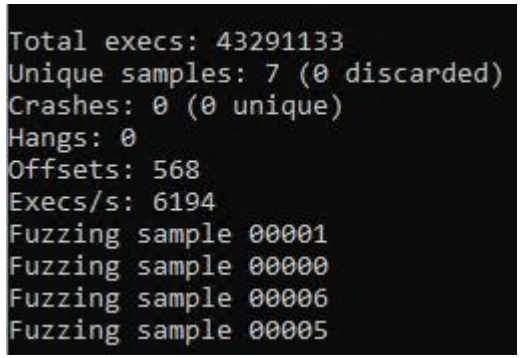
要注意传入的参数可以做一定的预处理

漏洞挖掘篇

- 3、选好种子（符合条件的输入），启动fuzzer。

```
fuzzer.exe -in C:\tmp\wxam_inputs -out C:\tmp\jOut -t 30000 -nthreads 2 -delivery shmem -max_sample_size 1000000 -instrument_module WeChatWin.dll -instrument_module voipEngine.dll -persist -target_module WxamFuzzer.exe -target_method fuzz -nargs 0 -iterations 80000 -cmp_coverage -- WxamFuzzer.exe @@
```

- 4、等待Crash：



漏洞挖掘篇

深入

- 文章中详细介绍了如何对微信进行逆向，主要思路就是搜索日志打印函数。
- 并且，闭源fuzz主要要求要逆懂程序，因此我们可以通过文章中的逆向方案找到别的可能出现漏洞的函数，对它进行fuzz，挖掘出新的漏洞。
- 以下是我研究的成果：

提交时间	漏洞名称	当前状态
2024/03/14	微信 Windows端[redacted]存在堆溢出漏洞	已修复
2024/03/12	企业微信 Windows端[redacted]存在栈溢出漏洞	已修复
2024/03/12	QQNT Windows端[redacted]存在栈溢出漏洞	已修复
2024/01/19	微信[redacted]存在栈溢出漏洞	已修复
2024/01/16	微信[redacted]存在栈溢出漏洞	已修复

漏洞挖掘篇

我的个人经验（逆向）

- 针对性逆向
  - 针对性逆向与攻击面有关的逻辑
  - 针对性看协议文档
- 举个例子
  - 跟着解析文件数据流走
  - 跟着传入数据流走

漏洞挖掘篇

我的个人经验（Fuzzing）

- 通过攻击面来选择/定制fuzzer
  - 文件格式类：afl、winafl、Jackalope
  - 浏览器：fuzzilli
  - 等等

漏洞挖掘篇

我的个人经验（攻击面选取）

- 通过攻击面来选择/定制fuzzer
  - 尽量选择解析类函数：decode、parse等等
  - 尽量选择非第三库函数：不要对着libpng、libwebp这样的库的函数挖
  - 尽量选择历史漏洞少的方向
  - 尽量关注自研的文件格式有关函数

漏洞挖掘篇

我的个人经验（漏洞检测机制）

背景：

开源程序可以通过ASAN对轻松检测到栈内存和堆内存漏洞。而windows**闭源程序**一般都开启了GS保护，可以用GS保护来检测栈溢出漏洞。但是**堆内存漏洞就比较难检测。**

2022 SDC 议题回顾 | Dumart fuzz：让黑盒像白盒一样fuzz

然后我malloc的时候，我malloc一个size，我们就在边界的地方，往回减一个size，减了size的地址后，就作为一个指针返回的给用户使用。用户使用的时候，如果在size内的话，它这是正常的，如果访问超过了这个size就会落到红色这一部分，这里是不可读写，那就会触发一个断错误。我们就等于检测到了。

然后下溢出的检测就反过来，这次低地址是一个不可读写的，然后我们返回的就是一个就是边界点，如果它往这边界点以下的地址再访问，那就访问到不可读写的部分。

然后UAF的检测也能够去实现，有点像延时释放，因为我们第一次Free的时候我们就不重用，直接把它的页属性改成是不可读写。如果他再使用到这一个内存的话，这里就会发生断错误，那我们也检测出来了。

漏洞挖掘篇

缓解方案：我们可以使用完全页堆机制来缓解检测堆内存漏洞，为什么说是缓解？因为开启**页堆机制**会让fuzz效率大大的下降！

- 如何开启完全页堆机制？
  - 对要fuzz的程序以**管理员权限**执行：gflags.exe /i <target_executable> +hpa即可

THE END