pwnable.kr "codemap"

思路有两种，一种是逆向出原始的 C 文件，然后编译运行；一种是 patch 二进制文件，在运行过程中把 size 和 chunk 中存储的字符串打印出来，找到第二大的。

首先尝试了第一种方法，一个函数一个函数的分析程序的执行流程，程序虽然看上去很复杂，但是静下心来在纸上认真分析发现，程序的很多代码其实都是重复性的。逆向出源文件然后编译运行发现和原程序的运行结果不一致，猜想会不会是 srand 函数和 rand 函数在 linux 平台和 windows 平台实现不同所导致的。在 windows 下编译运行的结果果然和 linux 下不一样，但是和原程序还是不一致。去看源程序的 srand 函数和 rand 函数，居然不是动态链接的，程序里面有这两个函数的具体实现。依照它的实现，我自己定义了等价的 srand 和 rand 函数，再次编译运行，还是不对！这个时候就有点抓狂了，检查了好几遍也没找出错误。

下面是逆向出的 C 代码

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int next;
void _srand(int seed) {
    next = seed;
}

int _rand() {
    next = 214013 * next + 2531011;
    return (next >> 16) & 0x7fff;
}
int loop(int x) {
    x = (14*(x>>5)+33)*((x*x*x+4)>>2);
    int i;
    for(i = 0; i < 44; i++) {
        x = x + 1;
        x = x * x;
        x = x - 33;
        x = x % 2423400;
        x = x + 5555555;
        x = x + 123423431;
        x = x + 123423423;
        x = x * 234331;
        x = x >> 1;
        x = x + 331;
        x = x + 1131;
        x = x - 33321;
```

```
        x = x + 123131;
        x = x - 2342341;
        x = x + 1345345;
        x = x - 4564561;
        x = x + 5675671;
        x = x - 1678678;
        x = x + 4646461;
        x = x - 131231;
    }
    return x;
}
int main() {
  int rnum; // esi@3
  int rn6; // esi@4
  unsigned int n_rand; // eax@8
  unsigned int _msize; // esi@8
  char *pm; // ebx@8
  unsigned int i; // esi@9
  char *p_max; // [sp+10h] [bp-60h]@0
  unsigned int chunk_sz; // [sp+14h] [bp-5Ch]@8
  unsigned int j; // [sp+18h] [bp-58h]@1
  unsigned int max_size; // [sp+1Ch] [bp-54h]@1
  char s[64]; // [sp+20h] [bp-50h]@9
  int g; // [sp+6Ch] [bp-4h]@3

  memcpy(s,
"abcdefghijklmnopqrstubwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890", 63u);
  printf("I will make 1000 heap chunks with random size\n");
  printf("each heap chunk has a random string\n");
  printf("press enter to start the memory allocation\n");
  max_size = 0;
  j = 0;
  _srand(0);
  while ( 1 )
  {
    rnum = 10000 * _rand() % 1337;
    rn6 = (10000 * rnum >> 1) + 123;
    rn6 = 1000000000 * rn6 % 123456789;
    n_rand = loop(rn6);

    chunk_sz = n_rand % 100000;
    pm = new char[chunk_sz];
    if ( chunk_sz >= 0x10 )
    {
      i = 0;
      do {
```

```
        pm[++i - 1] = s[_rand() % 62];
      } while ( i < 15 );
      pm[15] = 0;

      if ( chunk_sz > max_size )
      {
        max_size = chunk_sz;
        p_max = pm;
      }
    }
    if ( ++j >= 1000 )
      break;
    _srand(j);
  }
  printf("the allcated memory size of biggest chunk is %d byte\n",
max_size);
  printf("the string inside that chunk is %s\n", p_max);
  printf("log in to pwnable.kr and anwer some question to get flag.\n");
  return 0;
}

```

只好尝试第二种方法，patch！想到了把打印 chunk size 和 chunk content 的两个 printf 函数放到循环里面的方法。但是运行的时候只打印了一次结果，然后直接崩溃了。后来发现 ida 的汇编窗口提示 sp-analysis failed，原因是连续调用两次 printf 所 push 的参数占用了栈空间，而这段空间没有被回收。加上了一句 add esp,0x10 这个错误就解决了。虽然f5反汇编查看结果没有问题，但是运行的时候还是只打印了一次结果。当时想拿 od 调试的，但是 od 好久没用了，不大会用了，就有点懒得搞了。后来发现 ida 自带的 debug 功能相当好用。以后碰到这种情况要学会用 debugger 调试。

看了题目的 hint，隐隐约约想到了第三种方法就是在0x403e65处下断点，调试程序的时候把断点处的 eax，ebx 的值给打印出来，然后就能知道哪个 chunk 是第二大的了。但是当时没想到该怎么弄，看了 writeup 得知 idc 可以完成这个工作。折腾了好久才大概明白 idc 怎么弄。

下面是 idc 代码

```
#include<idc.idc>
static main()
{
    auto i;
    auto e_ip;
    auto sz;
    auto ss;
    auto eax,ebx;
    auto max_size, max_str;
    auto sec_size, sec_str;
    auto thd_size, thd_str;
```

```
    max_size = max_str = sec_size = sec_str = thd_size = thd_str = 0;

    Message("idc script start\n");

    AddBpt(0x12b3e65);
    StartDebugger("", "", "");

    for(i=0;i<1000;i++) {
        auto code = GetDebuggerEvent(WFNE_SUSP|WFNE_CONT,-1);
      //WFNE_CONT 表示会将 suspend 的 program 恢复运行
        if(code == BREAKPOINT) {
            eax = GetRegValue("eax");
            ebx = GetRegValue("ebx");
            //Message("eax=%08x\n",eax);
            sz=eax;
            ss=ebx;
            if(sz > max_size) {
                sec_size = max_size;
                sec_str = max_str;
                max_size = sz;
                max_str = ss;
            } else if(sz > sec_size){
                thd_size = sec_size;
                thd_str = sec_str;
                sec_size = sz;
                sec_str = ss;
            } else if(sz > thd_size) {
                thd_size = sz;
                thd_str = ss;
            }
        }
    }

    Message("second Max size is:%08x, pointer to string
is:%08x\n",sec_size, sec_str);
    Message("third Max size is:%08x, pointer to string
is:%08x\n",thd_size, thd_str);
}
```

坑点

1. 用 ida 去 patch 二进制文件的时候一直 patch 失败，以为是 ida 这个插件出了什么问题，后来才发现是自己把 codemap.exe 这个文件重命名了，导致 ida 找不到这个文件，也就无法 patch，把 codemap.exe 改回原来的名字就可以了。

2. idc 的命令行不能定义 auto 变量，因为命令行会把用户的输入包装在一个函数中执行，如果定义 auto 变量的话，变量的作用域就只局限于函数内部。执行下一条命令的时候是看不到这个变量的，就会提示 undefined variable。

困惑

1. ida patch 二进制文件然后运行，在 patch 的地方下断点，却发现断点处的指令与原来 patch 的指令不一样了。立马终止调试去看程序的汇编代码，变了好多，反编译的源代码也变得乱七八糟，不知道怎么回事。
2. idc 脚本里面直接访问 eax 总是提示 undefined，在 StepInfo()或者 RunTo()后面调用 GetRegValue("eax")一开始也提示 undefined，在 GetDebuggerEvent()后面就没有报错了。