0x1 – MISC

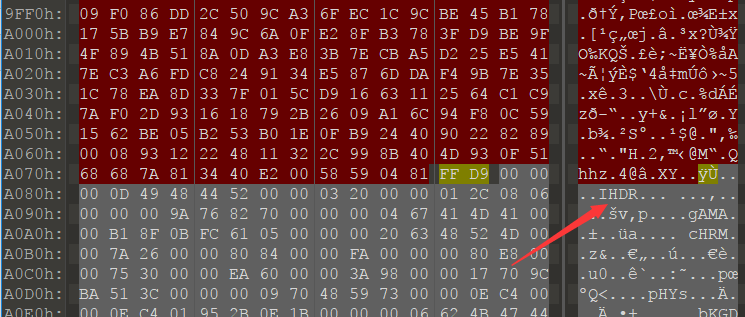
　　MISC100

　　一张帅行的照片



目测是图片隐写，但是binwalk并没有出来，应该是对文件头进行了修改

　　010editor查看一下，发现在jpg文件尾之后还有大量的数据



　　而且在灰色部分发现了IHDR，是png文件的一个标志，所以应该是文件连缀之后删去了png的文件头、尾

　　所以可以进行修复，在灰色部分前后分别加上png的文件头尾，并删去前面帅行的数据

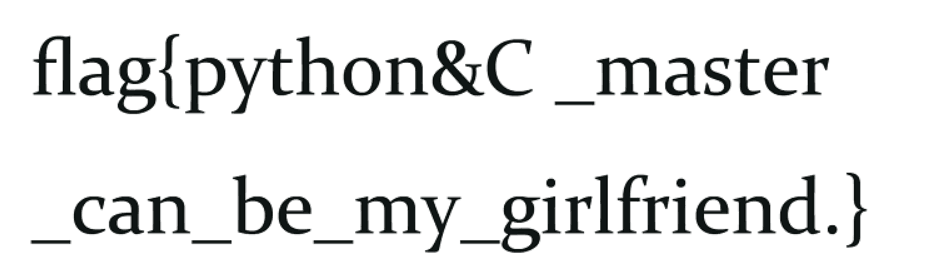
　　保存之后发现图片是纯白的，而且在linux下无法打开

　　->有可能是文件长宽被修改之后crc校验不匹配导致的

　　详情见M4X司机http://www.cnblogs.com/WangAoBo/p/7108278.html

　　提供两种方法

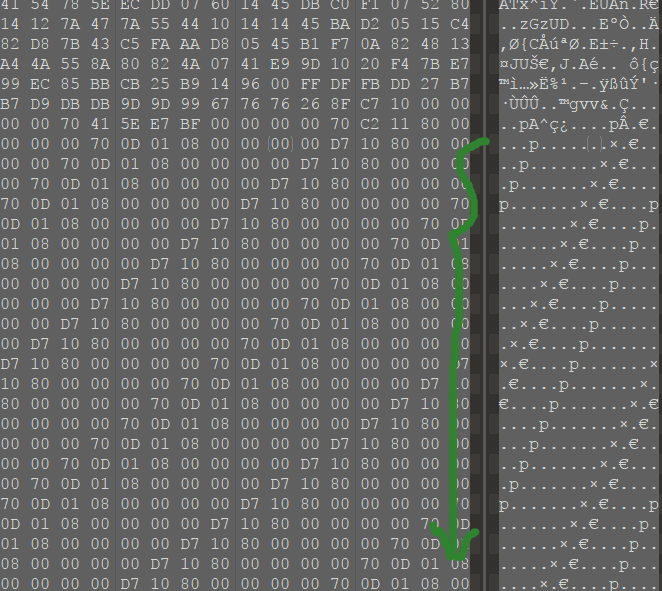
1. 利用crc进行宽和高的爆破，得到正确的尺寸，修改得到完整图片



　　flag就是帅行的女朋友标准

②直接删去部分数据

　　这个操作不是很稳，但是比写脚本爆破要快一点



　　可以看到在图片中存在大量的单调规则的数据，不会是有价值的信息

　　所以可以直接把这部分删掉，把下面被隐藏的部分暴露出来



　　效果也还不错

　　MISC200

　　一个多重压缩包，tar.gz方式压缩的

　　经过简单查看，每两层一个数字作为名字，先解压gz，之后是tar包

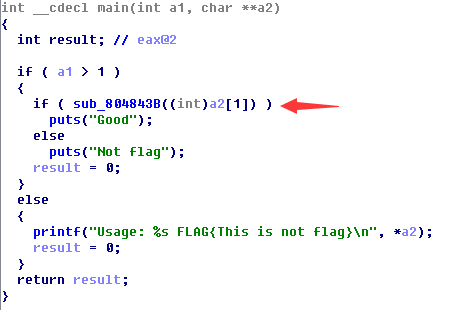
　　exp：

get flag：

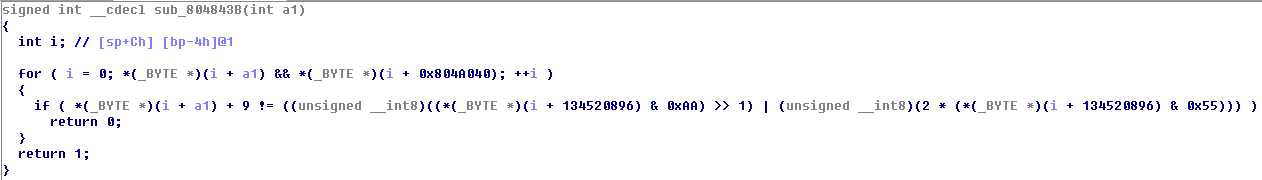
http://images2017.cnblogs.com/blog/1251324/201712/1251324-20171218111649678-975707094.png

0x2 - Reverse

　　Reverse100



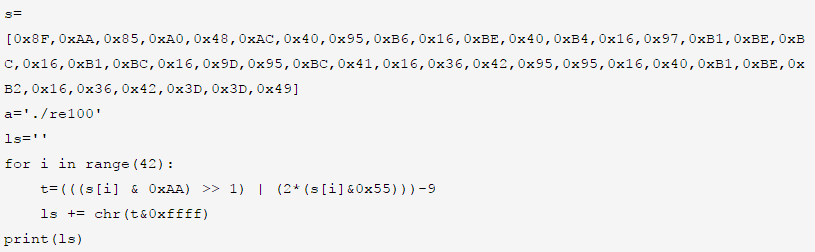
　　这就是关键函数了



　　把a1的每一位和0x804A040的每一位进行比对

　　比较简单的逻辑，把运算的优先级看好，按照逻辑写出反向计算脚本

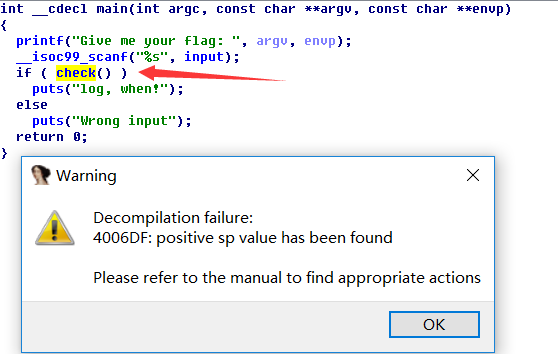
　　exp：



　　需要注意的是，减号的优先级还挺高的，漏掉原来的一个括号，就得不到正确的结果了

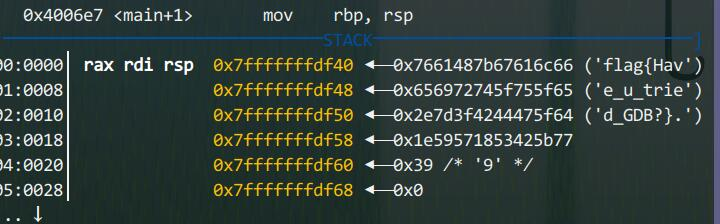
http://images2017.cnblogs.com/blog/1251324/201712/1251324-20171218112720131-137989198.png

Reverse200



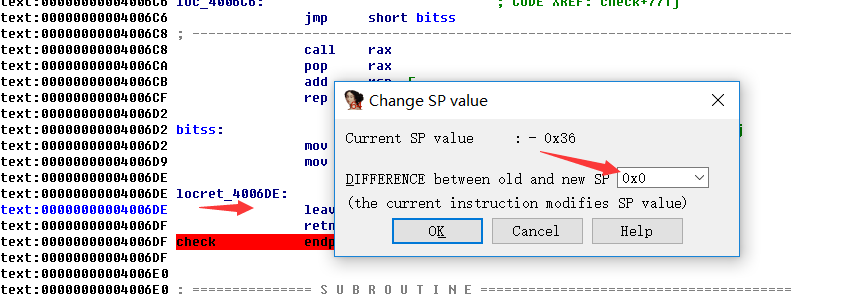
　　一个64位程序，check被破坏了，不能用ida使用F5，下面给出三个方法

1. 使用gdb调试

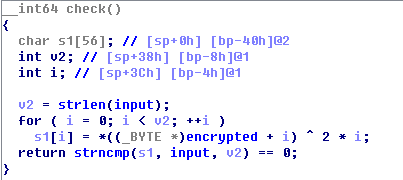


　　在gdb中，找到strcmp函数，查看参数就能直接看到flag

1. 修复程序，使其可以使用F5查看伪代码



　　左箭头指向处ALT+ k，把右箭头指向处改为0即可

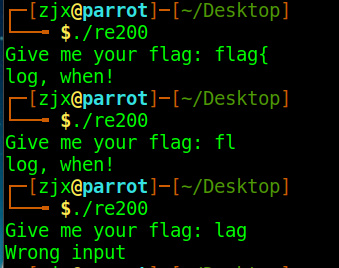


　　下面应该就比较好弄了

③利用程序漏洞爆破

　　也就我这么皮，上面删图片数据，下面用漏洞爆破了

　　在题目更新之前，存在一个逻辑错误



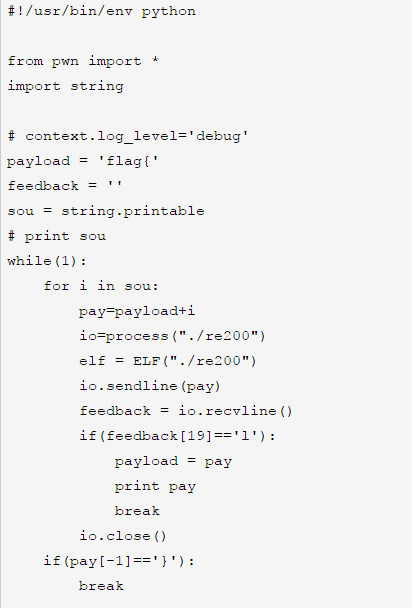
　　在试运行时发现输入“flag”，也能得到输入正确的反馈

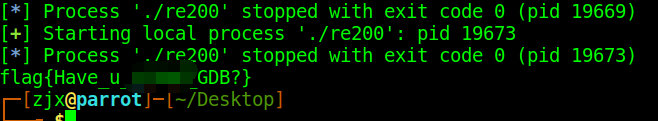
　　所以很容易想到如果输入预期字符串的前n的字符，那么会得到正确回馈

　　所以可以根据这个特性进行逐位爆破：

　　　　在已知flag后加入一个可打印字符，如果回馈正确，就把这个字符加入flag中

　　　　到最后能得到完整的flag

exp：



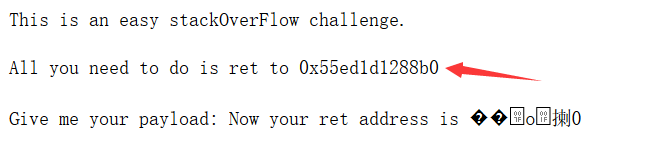
　　也就几十秒就出来了，看来官方解答还是用gdb嘛

0x3 - PWN

pwn100

　　没有elf的pwn

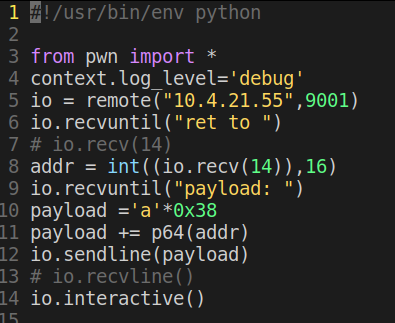
　　将ip和端口在浏览器打开就是程序的输出



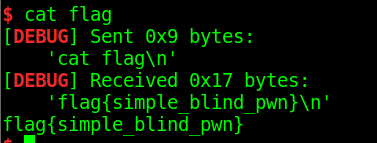
　　给出了溢出跳转的函数地址，而且是64位的地址

　　所以只需要我们根据输入payload之后的回馈，将该地址覆写到合适的位置，即找出填充字符串的长度

　　经过尝试，填充字符串为0x38,所以得到exp



　　地址是随机化的，按照小端序和返回提示，找到正确的填充长度



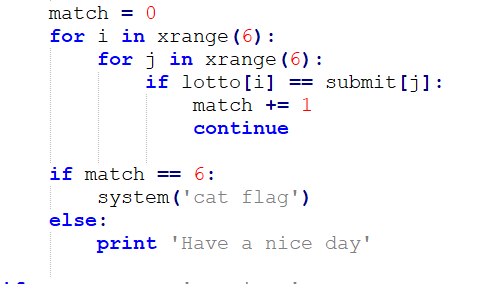
pwn200

　　好题！开创了py出pwn的先河！

　　一个pyc文件，可以很容易反编译出准确的源码

~~那么面临两个问题：找出程序中system函数的地址，确定溢出长度~~

　　好了，上一行划掉。这不是一个溢出的漏洞，而是程序逻辑错误

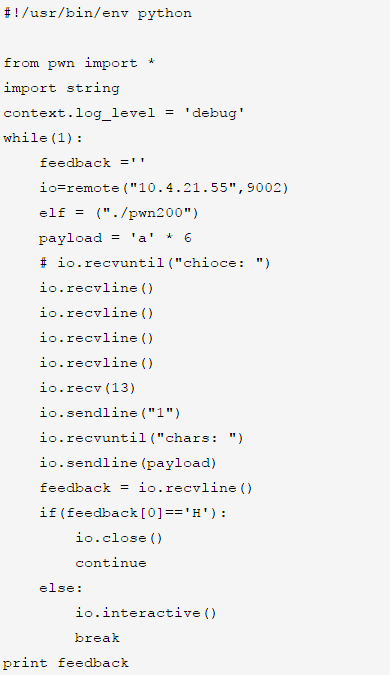


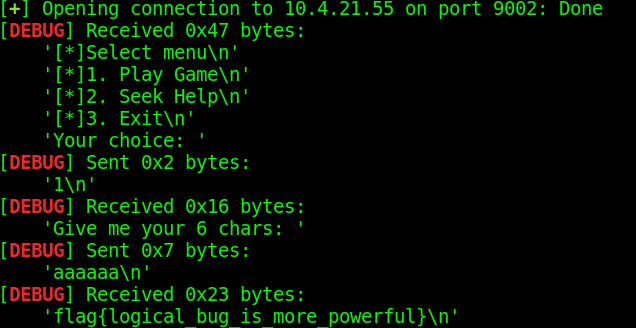
　　可以发现，在程序进行随机字符比对时，没有进行一对一的比较，而是6\*6的比较方式

　　这就造成，只要我们的输入和随机字符串有一个相同，就能实现match==6

　　所以我们可以采用相同的key = 'aaaaaa',循环的执行程序，直到遇到含有一个‘a'的随机字符组合

exp：

很快就能跑出结果



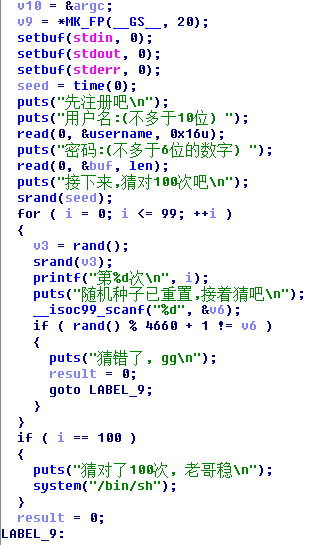
pwn400

有点坑是真的。。



　　美(sang)丽(xin)壮(bing)观(kuang)的全保护题目，而且确实限制了读取长度

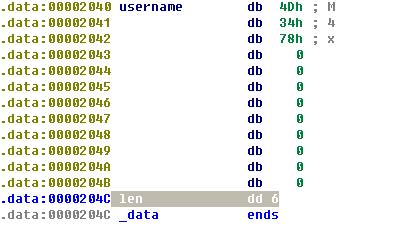
　　在ida中查看



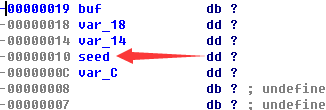
　　开了金丝雀，在目前没有get绕过canary的情况下，栈溢出控制返回地址就行不通了

　　但是可以发现两点比较有价值的：

1. username在data区，下面紧邻了len，也就是控制密码读取长度的变量



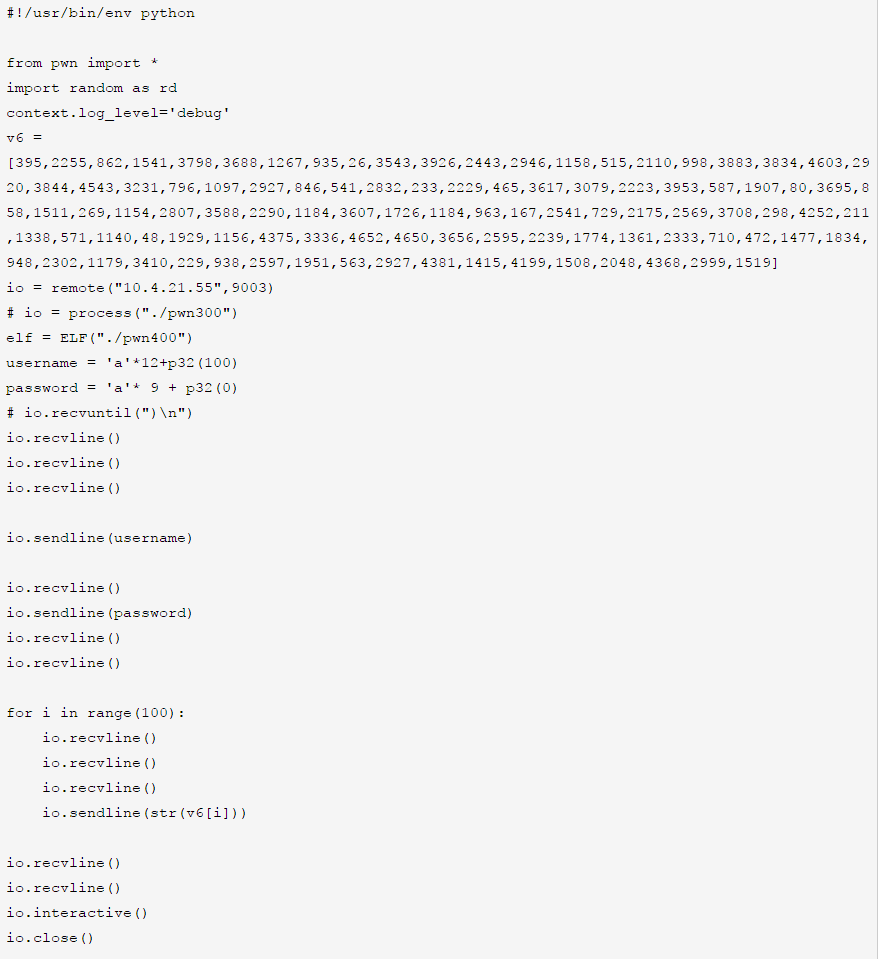
1. 密码(buf)的下面有seed



　　所以我们可以有以下思路：

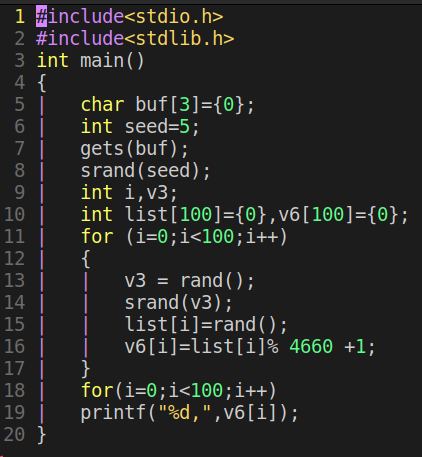
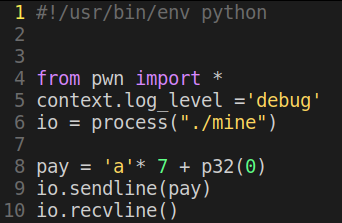
　　　　先在username输入时覆盖len，使其长度足够

　　　　输入password时，覆盖seed，使随机数种子已知

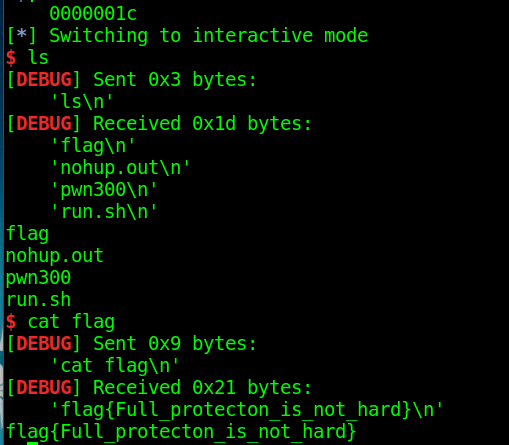
如此控制了随机数的种子，便掌控一切

　　(在尝试过程中遇到几次recvuntil不能正常recv的情况，便改成了多个recvline)

需要注意的是，此时虽然在seed处覆盖了“p32(0)”,但是并不是“srand(0)”

　　所以随机数表可通过自己编写C脚本，使用相同的值覆盖seed，得到和程序相同的随机数表

掌控了随机数，便可以按顺序输入给程序，使得猜对100次，获得shell



0x4~~总结~~扯皮

　　这次的训练赛收获还是很大的，虽然有些地方是投机取巧的，但pwn做的还是很舒服（全靠M4x司机带）

　　三个pwn题都比较有新意，突破了之前练习的常规思路

　　第一个没有elf，还是第一次遇到，不能对程序进行分析，抱着试试的态度解锁盲pwn新姿势

　　第二个突破天际，竟然是py，之前都是常规套路的溢出漏洞，找了半天怎么对pyc文件进行溢出的，经过提醒才仔细看代码的逻辑漏洞，关注点还是要全面一点吧

　　第三个吊炸天的全保护。最开始注意到len和seed的位置特殊时，还真有过一丝控制随机数的想法，但总觉得太过暴力，不符合pwn的美学，想去找适合新手用的绕过金丝雀进行溢出，然而事实就是这么粗暴。。。等我get了绕canary，再回来收拾收拾这家伙，美学不容破坏