12/30/2017 Team: rm-ft/

Junior CTF(34C3) Writeup

Weizhou Wang

]

Crypto:

1. Kim:

It provided a python file source code for reviewing, shown as following:

```
from bottle import route, run, template, request, redirect, static_file
import hashlib
from secret import SECRET

INDEX = '<html><body><p>Download a sample <a href=/files/{{mac}}/?f={{file}}>her
FILES = '<html><body><ul><li>sample.gif</li><li>dont.gif</li><li>flag</li>'

def mac(msg):
    return hashlib.sha1(SECRET + msg).hexdigest()

@route('/')
def index():
    f = 'sample.gif'
    return template(INDEX, mac=mac('f=' + f), file=f)

@route('/files/')
def dir():
    return FILES

@route('/files/<umac>/')
def download(umac):
    delim = msg = ''
    for k,v in request.query.allitems():
        msg += delim + k + '=' + v
        delim = '&'
    if mac(msg) == umac:
        return static_file(request.query.f, root='./files')
    else:
        return redirect('/files/' + mac('f=dont.gif') + '/?f=dont.gif')

|

run(host='0.0.0.0', port=8888, server='paste')
```

The vulnerability should be "hashlib.sha1(SECRET + msg)", because the SECRET directly appended by the message which will lead to a HASH extension attack. More details about Hash extension attack can be obtained by searching online, but the basic idea of hash extension attack is:

Use known plaintext and hash code, hashed with a secrete salt value in the server, to make a extended plaintext with corresponded hash bypass the hash authenticated verification. The basic method is appending 0x80 0x00… source:

http://blog.csdn.net/syh_486_007/article/details/51228628

According to the source code above, to get the flag we have to access:

http://35.198.133.163:1337/files/<UNKNOWN_HASH>/?f=sample.gif&f=dont.gif&flag

And the <UNKNOWN_HASH> should be the correct hash value of the “f=sample.gif & f=don’t.gif & flag”. But we do not have the SECRET string of the hash, we cannot calculate the correct value of <UNKNOWN_HASH>.

However now we can successfully access to:

http://35.198.133.163:1337/files/952bb2a215b032abe27d24296be099dc3334755c/?f=sample.gif

and according to the hash extension attack we could make a string like “f=sample.gif %0x80%0x00 … &f=don’t.gif & flag” and a corresponded hash value to bypass the authentication and get the flag. The tool we used for attacking is python with hashpumpy module. It can automatically generate attacking string and hash value with the correct hash, string, appendix and key length input. However, in this scenario, key length is unknown; and we assume it should less than 64 bytes, then we brute force and generate all the possible values and try everyone from the beginning. Finally, the flag was captured with 16 lengths of the key.

There is a video illustrated all the detailed attacking steps:

https://www.youtube.com/watch?v=6QQ4kgDWQ9w

2. Top secret:

Description:

It provides us a python source code which can be independently executed, and a secret file should be corrupted by breaking the encryption algorithm.

The source code shown as following:

```
import random
import sys
import time

cur_time = str(time.time()).encode('ASCII')
random.seed(cur_time)

msg = input('Your message: ').encode('ASCII')
key = [random.randrange(256) for _ in msg]
c = [m ^ k for (m,k ) in zip(msg + cur_time, key + [0x88]*len(cur_time))]

with open(sys.argv[1], "wb") as f:
    f.write(bytes(c))
```

Lets trace the output message to the very beginning:

First, the encrypted message is the combination of encrypted messages and time stamp. Then by reviewing the source code, the encrypted key is related to time stamp. Since we could know the length of the timestamp easily, extracting encrypted message is simple, and if we could figure out the key string, then the secret message would be decrypted.

The key point or vulnerability is the random function is Pseudorandom that it used time stamp as randomized seed, which means every time the random number set can be the same. Therefore, based on the discovery, I figured out the time stamp in the encrypted message, which is xor with the “0x88”, and calculate the key with time stamp inputting to the random generation engine.

Then the message can be decrypted easily, and the flag is right there.

PWN

1. Digital Billboard:

```
struct billboard {
    char text[256];
    char devmode;
};
struct billboard bb = { .text="Placeholder", .devmode=0 };

void set_text(int argc, char* argv[]) {
    strcpy(bb.text, argv[1]);
    printf("Successfully set text to: %s\n", bb.text);
    return;
}

void shell(int argc, char* argv[]) {
    if (bb.devmode) {
        printf("Developer access to billboard granted.\n");
        system("/bin/bash");
    } else {
        printf("Developer mode disabled!\n");
    }
    return;
}
```

To capture the flag, we need to buffer over flow the text field. In the set_text method, it directly uses strcpy() function leading to buffer overflow exploitation. Also in the shell, if the devmode is 1 we can execute the /bin/bash and the flag must in it. To change the devmode we can input more than 256 characters of ‘1’, then the devmode field can be overwrite to ‘1’. After overwriting the parameter, we can successfully run the /bin/bash, and the flag is under the directory.

REVERSE ENGINEERING

1. ARM1:

The flag can be directly captured by reversing the binary file. And the flag is in the data segment.

```
ROM:00000544                  DCB 0x54 ; T
ROM:00000545 aHeFlagIs34c3_i DCB "he flag is: 34C3_I_4dm1t_it_1_f0und_th!s_with_str1ngs",0xD,0xA,0
ROM:0000057D                  DCB 0, 0, 0
ROM:00000580                  DCD 0x20000000, 0x800027D, 0x8000131, 0x800010D, 0
ROM:00000594                  DCB 0xEC ; 8
ROM:00000595                  DCB 2, 0, 0x20
ROM:00000598                  DCD 0x20000354, 0x200003BC, 0, 0, 0, 0, 0, 0, 0, 0, 0
ROM:00000598                  DCD 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
ROM:00000598                  DCD 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
ROM:00000638                  DCB    1
ROM:00000639                  DCB 0, 0, 0
ROM:0000063C                  ALIGN 0x10
ROM:00000640                  ADDS            R3, #0xE
ROM:00000642                  ADD             R3, SP, #0x334
ROM:00000644                  ASRS            R4, R6, #8
ROM:00000646                  B               loc_324
```

2. ARM2:

After reverse the binary file with the IDA pro we can find the hint of the flag in the data segment:

```
ROM:00000575 aHeFlagIs        DCB "he flag is: ",0
ROM:00000582                  ALIGN 4
ROM:00000584                  DCB  0xD
ROM:00000585                  DCB 0xA, 0, 0
ROM:00000588                  DCD 0x20000000, 0x66166166, 0x27650D0A, 0x363B660A, 0x21252C27
ROM:00000588                  DCD 0xA3B653C, 0x370A2664, 0xA212666, 0x252C2736, 0x6521
ROM:00000588                  DCD 0x800027D, 0x8000131, 0x800010D, 0
ROM:000005C0                  DCB 0xEC ; 8
ROM:000005C1                  DCB 2, 0, 0x20
ROM:000005C4                  DCD 0x20000354, 0x200003BC, 0, 0, 0, 0, 0, 0, 0, 0, 0
ROM:000005C4                  DCD 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
ROM:000005C4                  DCD 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
ROM:00000664                  DCB    1
ROM:00000665                  DCB 0, 0, 0
ROM:00000668                  DCD 0
```

Obviously, there is a segment of hex message is unreadable, and, therefore, it must be processed by some function.

```
void __fastcall __noreturn sub_290(int a1, int a2)
{
  int v2; // r0@1
  _BYTE *i; // [sp+Ch] [bp+Ch]@1

  v2 = sub_428(a1, a2);
  sub_334(v2);
  sub_400(134219124);
  for ( i = (_BYTE *)0x800058C; *i; ++i )
    sub_3D4(*i ^ 0x55);
  sub_400(0x8000584);
  while ( 1 )
    ;
}
```

While searching for some method for processing hex code, here is a suspicious piece: it seems a function which means xor a segment of data with 0x55. Then I tried this and xor for original plaintext, and the flag was in it.

3. Saturn

Description:

We were given a source code of program running on the remote server: 35.198.169.47 1337. By logging on the remote server, nothing shown on the screen. Then by analyzing the binary file with ida pro and inspecting every function listed, I can find a piece of code related to flag:

```
signed int __cdecl sub_843(int a1)
{
  signed int result; // eax@3
  int v2; // [sp+0h] [bp-20h]@1
  int v3; // [sp+4h] [bp-1Ch]@4
  int v4; // [sp+8h] [bp-18h]@7
  int v5; // [sp+Ch] [bp-14h]@10
  int v6; // [sp+10h] [bp-10h]@1
  int v7; // [sp+14h] [bp-Ch]@1
  int *v8; // [sp+1Ch] [bp-4h]@1

  v8 = &a1;
  v7 = *MK_FP(__GS__, 20);
  v6 = 1;
  sub_7DC();
  __isoc99_scanf();
  if ( v2 > 123456788 && v2 <= 987654321 )
  {
    if ( v3 > 123456788 && v3 <= 987654321 )
    {
      if ( v4 > 123456788 && v4 <= 987654321 )
      {
        if ( v5 > 123456788 && v5 <= 987654321 )
        {
          v6 = v4 * v3 * v2 * v5 * ((v2 << 25) % 30);
          v6 ^= v3 >> 3;
          v6 += v4 - v5;
          if ( v6 == 842675475 )
          {
            puts("Correct! Here is your flag:\n");
            sub_76D();
          }
          else
          {
            puts("Wrong!\n");
          }
          result = 0;
        }
        else
        {
          result = -1;
        }
      }
      else
      {
        result = -1;
      }
    }
    else
    {
      result = -1;
```

It is obvious that we have to find the value which can pass through all the "if" condition clause. Soon we can list a serial of function for calculation. Classic brute force method for getting the number is inefficient. We have run the fundamental brute force method for an hour, and nothing shown on the screen. Then by assuming there should be many solutions set of the puzzle, we decide to let v4 = v5, which can dramatically reduce the searching scope. Then by running the python script, we luckily found a set of the value. Finally, we captured the flag consequently.

MISC:

Babybash:

Description:

It is a bash-jail, to get the flag we have to execute /get_flag command. But, some characters are forbidden to be input in the bash:

- a-z
- *
- ?
- .

There is a method that we can execute the command with capital characters:

STR="/GET_FLAG"; ${STR,,}

By inputting above command in line, the command can be executed successfully. Then there fore with following instructions we could get the flag.