Write-up_picoCTF_2021

Chall: Rolling My Own

You can find out the challenge's source and solving script here(source, script)

> Firstly, I think two hints of this challenge is very important. Firstly, you should deeply the paper related directly this challenge. Besides, the second hint ***"Here's the start of the password:*** `D1v1` ***"*** is also worthy.

The main idea of paper

This paper suggested an anti-disassembly technique by using cryptographic hash. Particularly, a key and salt were chosen to generate the hash digest which contains some bytes of machine code(also called "running code"). In program there are a function which pick some bytes at arbitrary position in those hash digests (these positions will be decided by programmer before) and create running code. The disassembly of IDA or well-known disassembler will failed in disassembling those bytes because the byte arrays which they received are hash digests, no machine code.

The key steps to right password

The idea of challenge

- Based on above paper, author claims the right keys from us to concatenate them with available salts to create plaintexts, get MD5 digests of these plaintexts.
- In each generated md5 digests, there is a ***running*** code. Merging all running code at specified positions in MD5 digests, I have a function which can call flag-opening function.

Step-by-step to solve

Determine the salts and positions of running code in MD5 digest

> The pseudo-code generated by IDA was edited(function's name, variable's name...) to simplify analysis process.

```
v17 = 'WEjMaLpG';
v18 = 'kmnnjOVp';
v19 = '6pdeliGR';
v20 = 'slxzecvM';
v21 = 0;
v11 = 8;
v12 = 2;
v13 = 7;
v14 = 1;
memset(passwdInput, 0, 0x40uLL);
memset(&dest, 0, 0x40uLL);
printf("Password: ", 0LL);
fgets(passwdInput, 64, stdin);
```

Storing salts and positions of running code on stack

- While debugging and analyzing with arbitrary password, I recognized that v17 - v20 are salts of plaintext(easily concluded there are four 16-byte MD5 digests) and v11 - v14 are starting positions of running code in those digests in corresponding.

Determine the running code to open flag

```
for ( i = 0; i <= 3; ++i )
{
  strncat(&dest, &passwdInput[4 * i], 4uLL);
  strncat(&dest, (const char *)&v17 + 8 * i, 8uLL);
}
```

- Above snipped-code will concatenate the part of password(each 4 bytes at one time) with its salt(8 bytes). Finally, we will have 4 12-byte plaintexts to do cryptographic hash

```
someCryptoHashStuff((__int64)msg_Digest, (__int64)&dest, v3);
for ( ia = 0; ia <= 3; ++ia )
{
  for ( j = 0; j <= 3; ++j )
    *((_BYTE *)&runArray + 4 * j + ia) = msg_Digest[16 * j + ia + *(&v11 + j)];
}
v4 = (void (__fastcall *)(unsigned __int64 (__fastcall *)(__int64), signed __int64))mmap(0LL, 0x10uLL, 7, 34, -1, 0LL);
v5 = v16;
*(_QWORD *)v4 = runArray;
*((_QWORD *)v4 + 1) = v5;
v4(getFlag, 16LL);
```

- From the loops above, I easily discovered that there are 16 bytes running code(get from 4 bytes of each MD5 digests).

Conclude about running code: ***its length is 16 bytes and it can call the flag-opening function. (*)***

There are two noticeable points which I haven't yet mention above. They are: challenge's second hint and the flag-opening function.

1 - Hint:

> ***Here's the start of the password:*** `D1v1`

- I will calculate the MD5 digest of the first pass of password concatenating its corresponding salt "**GpLaMjEW**".
  MD5 digest in hexa: **23 f1 44 e0 8b 60 3e 72 48 89 fe 48 9f 78 fa 53**
- If I get 4 adjacent bytes starting digest[8](because the specified of running code in first MD5 digest is 8), I will have **48 89 fe 48**

, try disassembling it at here:

```
0: 48 89 fe          mov rsi,rdi
3: 48                rex.W( maybe miss some bytes to create a completed instruction)
```

2 - The flag- opening function

```
getFlag proc near

var_A8= qword ptr -0A8h
stream= qword ptr -98h
s= byte ptr -90h
var_8= qword ptr -8

; __unwind {
push    rbp
mov     rbp, rsp
sub     rsp, 0B0h
mov     [rbp+var_A8], rdi
mov     rax, fs:28h
mov     [rbp+var_8], rax
xor     eax, eax
mov     rax, 7B3DC26F1h
cmp     [rbp+var_A8], rax
jz      short loc_5631E3D4106D
```

```
loc_5631E3D4106D:
lea     rsi, modes        ; "r"
lea     rdi, filename     ; "flag"
call    _fopen
mov     [rbp+stream], rax
cmp     [rbp+stream], 0
jnz     short loc_5631E3D410A7
```

- To open ***flag.txt*** file, %rdi register must be assigned equally 0x7b3dc26f1. This is a task of running code. **(**)**

From (*) and (**), I will create running code appropriately to satisfy all above conditions:

```
#Before, %rdi stored the address of flag-opening function
mov     rsi,rdi             #%rsi = &flag-opening func
```

```
movabs rdi,0x7b3dc26f1    #%rdi = 0x7b3dc26f1
jmp    rsi                #jump to flag-opening func
ret

#and this is running code:
0:  48 89 fe                mov    rsi,rdi
3:  48 bf f1 26 dc b3 07    movabs rdi,0x7b3dc26f1
a:  00 00 00
d:  ff d6                   call   rsi
f:  c3                      ret
```

Brute-force three remaining keys

- Brute-force 3 remaining keys to create MD5 digest containing running code at specified positions.