2017_CodeGate_Junior

Nickname : Sori

Rank : 28

Score : 445

Name : 이진근

School : 한세사이버보안고

Mic Check

50 point

```
Mic Check
one two~ one two~

First Point : 500
Minimal Point : 50
Minus per one solver : -5

Here is Flag~ FLAG{Welcome_to_codegate2017}

real flag is in brackets.
```

flag : Welcome_to_codegate2017

BabyMISC

395 point

```
http://ctf.codegate.org/z/BabyMISC

nc 110.10.212.138 19090
nc 110.10.212.138 19091
```

바이너리와 nc주소를 준다.
바이너리를 IDA로 열어보았다

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  setbuf(stdout, 0LL);
  puts("[*] Ok, Let's Start. Input the write string on each stage!:)");
  if ( !sub_400EA8("[*] Ok, Let's Start. Input the write string on each stage!:)", 0LL) )
    exit(0);
  puts("[+] -- NEXT STAGE! ----------");
  if ( !sub_401005("[+] -- NEXT STAGE! ----------") )
    exit(0);
  puts("[+] -- NEXT STAGE! ----------");
  if ( !sub_401142("[+] -- NEXT STAGE! ----------") )
    exit(0);
  return 0LL;
}
```

총 함수가 3개 존재하는 것을 볼 수 있다. 함수의 리턴값을 모두 1로 만들어주어야 프로그램이 종료되지 않고 정상적으로 실행되는 것을 알 수 있다.

```
__int64 sub_400EA8()
{
  size_t v0; // rbx@1
  __int64 result; // rax@4
  __int64 v2; // rcx@6
  char *s2; // [sp+8h] [bp-98h]@1
  char *s1; // [sp+10h] [bp-90h]@1
  char *s; // [sp+18h] [bp-88h]@1
  char v6; // [sp+20h] [bp-80h]@1
  __int64 v7; // [sp+88h] [bp-18h]@1

  v7 = *MK_FP(__FS__, 40LL);
  s = "TjBfbTRuX2M0bDFfYWc0aW5fWTNzdDNyZDR5Oig=";
  setbuf(stdout, 0LL);
  puts("[*] -- STAGE 01 ----------");
  printf("[+] KEY : %s\n", s2);
  puts("[+] Input > ");
  setbuf(stdin, 0LL);
  __isoc99_scanf("%99s", &v6);
  sub_400D35("TjBfbTRuX2M0bDFfYWc0aW5fWTNzdDNyZDR5Oig=", &s2);
  sub_400D35(&v6, &s1);
  printf("[*] USER : %s\n", s1);
  v0 = strlen(s);
  result = v0 == strlen(&v6) && !strcmp(s1, s2) && strcmp(&v6, s);
  v2 = *MK_FP(__FS__, 40LL) ^ v7;
  return result;
}
```

stage1에서는 입력받은 값의 base64복호화 한 값 과 변수 s의 base64복호화 한 값이 서로 같아야 하고, 암호화 한 값이 서로 달라야 하고 글자 수는 같아야 한다.
input 받는 값을 TjBfbTRuX2M0bDFfYWc0aW5fWTNzdDNyZDR5Oig=에서 =한 글자만 다른걸로 바꾸어 우회 시켜주면 될 것 같았다. 옛날에 어떤 CTF에서 특수문자들이 무시가 되었던 것을 본 것 같아 특수문자를 넣어보았더니 아예 디코드가 되지 않았다. 이에 하나하나 값을 바꾸어 넣어보았더니 특수문자들은 아예 복호화가 되지 않았고 알파벳 A부터 넣어보았더니 A가 우회가 되었다.

```
__int64 sub_401005()
{
  size_t v0; // rbx@1
  __int64 result; // rax@3
  __int64 v2; // rcx@5
  char *s1; // [sp+0h] [bp-100h]@1
  char *s2; // [sp+8h] [bp-F8h]@1
  char s; // [sp+10h] [bp-F0h]@1
  char v6; // [sp+80h] [bp-80h]@1
  __int64 v7; // [sp+E8h] [bp-18h]@1

  v7 = *MK_FP(__FS__, 40LL);
  setbuf(stdout, 0LL);
  puts("[*] -- STAGE 02 ----------");
  puts("[+] Input 1 ");
  setbuf(stdin, 0LL);
  __isoc99_scanf("%99s", &s);
  puts("[+] Input 2 ");
  setbuf(stdin, 0LL);
  __isoc99_scanf("%99s", &v6);
  sub_400D35(&s, (void **)&s1);
  sub_400D35(&v6, (void **)&s2);
  v0 = strlen(&s);
  result = v0 != strlen(&v6) && !strcmp(s1, s2);
  v2 = *MK_FP(__FS__, 40LL) ^ v7;
  return result;
}
```

Stage2에선 입력을 두 개 받는다.

입력 받는 두 개의 문자의 수가 달라야 하고, 복호화 했을 때 A가 무시 되는걸로 글자 길이만 다르게 해주면 될 것 같았다.

base64encode(A)="QQ=="

input 1 : "QQAA",input 2 : "QQAAA"

왜 인지는 모르겠으나 2스테이지는 우회되는 것이 많았다. 처음에 풀었을 당시에는 Xshell의 backspace를 눌렀더니 우회가 되었다.

```
[+] -- NEXT STAGE! ----------
[*] -- STAGE 02 ----------
[+] Input 1
a^H^H
[+] Input 2
a
[+] -- NEXT STAGE! ----------
[*] -- STAGE 03 ----------
[+] Ok, It's easy task to you, isn't it? :)
[+] So I will give a chance to execute one command! :)
```

```
[*] -- STAGE 02 ----------
[+] Input 1
QQAA
[+] Input 2
QQAAA
[+] -- NEXT STAGE! ----------
[*] -- STAGE 03 ----------
[+] Ok, It's easy task to you, isn't it? :)
[+] So I will give a chance to execute one command! :)
[*] Input >
```

마지막 3스테이지에서는 문자를 입력받아 base64로 복호화 한 후 시스템 명령어를 하나 사용하게 해준다.
cat flag하면 될 줄 알았더니 안 되었다.

```
signed __int64 sub_401142()
{
  signed __int64 result; // rax@2
  char *v1; // rax@3
  __int64 v2; // rdx@4
  const char *v3; // [sp+8h] [bp-E8h]@1
  char src; // [sp+10h] [bp-E0h]@1
  __int64 dest; // [sp+80h] [bp-70h]@3
  char v6; // [sp+88h] [bp-68h]@3
  __int64 v7; // [sp+E8h] [bp-8h]@1

  v7 = *MK_FP(__FS__, 40LL);
  setbuf(stdout, 0LL);
  puts("[*] -- STAGE 03 ----------");
  puts("[+] Ok, It's easy task to you, isn't it? :)");
  puts("[+] So I will give a chance to execute one command! :)");
  puts("[*] Input > ");
  setbuf(stdin, 0LL);
  __isoc99_scanf("%99s", &src);
  sub_400D35(&src, (void **)&v3);
  if ( (unsigned int)sub_400E0B(v3) )
  {
    puts("[*] NoNoNo");
    result = 0LL;
  }
  else
  {
    dest = 233685487398318l669LL;
    v6 = 0;
    strcat((char *)&dest, &src);
    v1 = (char *)&dest + strlen((const char *)&dest);
    *(_QWORD *)v1 = 391966591279120284 8LL;
    *((_QWORD *)v1 + 1) = 829576699217793 0292LL;
    *((_WORD *)v1 + 8) = 104;
    printf("# %99s\n", &dest);
    system((const char *)&dest);
    result = 1LL;
  }
  v2 = *MK_FP(__FS__, 40LL) ^ v7;
  return result;
}
```

if분기점에서 사용할 명령어 체크를 하는데

```
__int64 __fastcall sub_400E0B(const char *a1)
{
  __int64 result; // rax@2
  int v2; // ST14_4@3
  __int64 v3; // rdx@4
  regex_t preg; // [sp+20h] [bp-50h]@1
  __int64 v5; // [sp+68h] [bp-8h]@1

  v5 = *MK_FP(__FS__, 40LL);
  if ( regcomp(&preg, "[/|$|-|_|&|>|`|'|\"|%|;]|(cat)|(flag)|(bin)|(sh)|(bash)", 1) )
  {
    result = 0LL;
  }
  else
  {
    v2 = regexec(&preg, a1, 0LL, 0LL, 0);
    regfree(&preg);
    result = v2 == 0;
  }
  v3 = *MK_FP(__FS__, 40LL) ^ v5;
  return result;
}
```

파일을 읽는 다른 명령어인 more와 와일드 카드로 우회하면 또 될 것 같았다.
baes64encode(“more f*”)= “bW9yZSBmKg==”

```
[*] -- STAGE 03 ----------
[+] Ok, It's easy task to you, isn't it? :)
[+] So I will give a chance to execute one command! :)
[*] Input >
bW9yZSBmKg==
#                                                    echo -n bW9yZSBmKg== | base64 -d | sh
::::::::::::::
flag
::::::::::::::
FLAG{Nav3r_L3t_y0ur_L3ft_h4nd_kn0w_wh4t_y0ur_r1ghT_h4nd5_H4ck1ng}
```

flag : Nav3r_L3t_yOur_L3ft_h4nd_knOw_wh4t_yOur_r1ghT_h4nd5_H4ck1ng