XCTF嘉年华体验赛逆向writeup

ptyin1604 于 2018-08-29 12:49:44 发布 754 收藏

文章标签： ctf 逆向 IT 计算机 信息安全

本文链接：https://blog.csdn.net/weixin_43090100/article/details/82180752

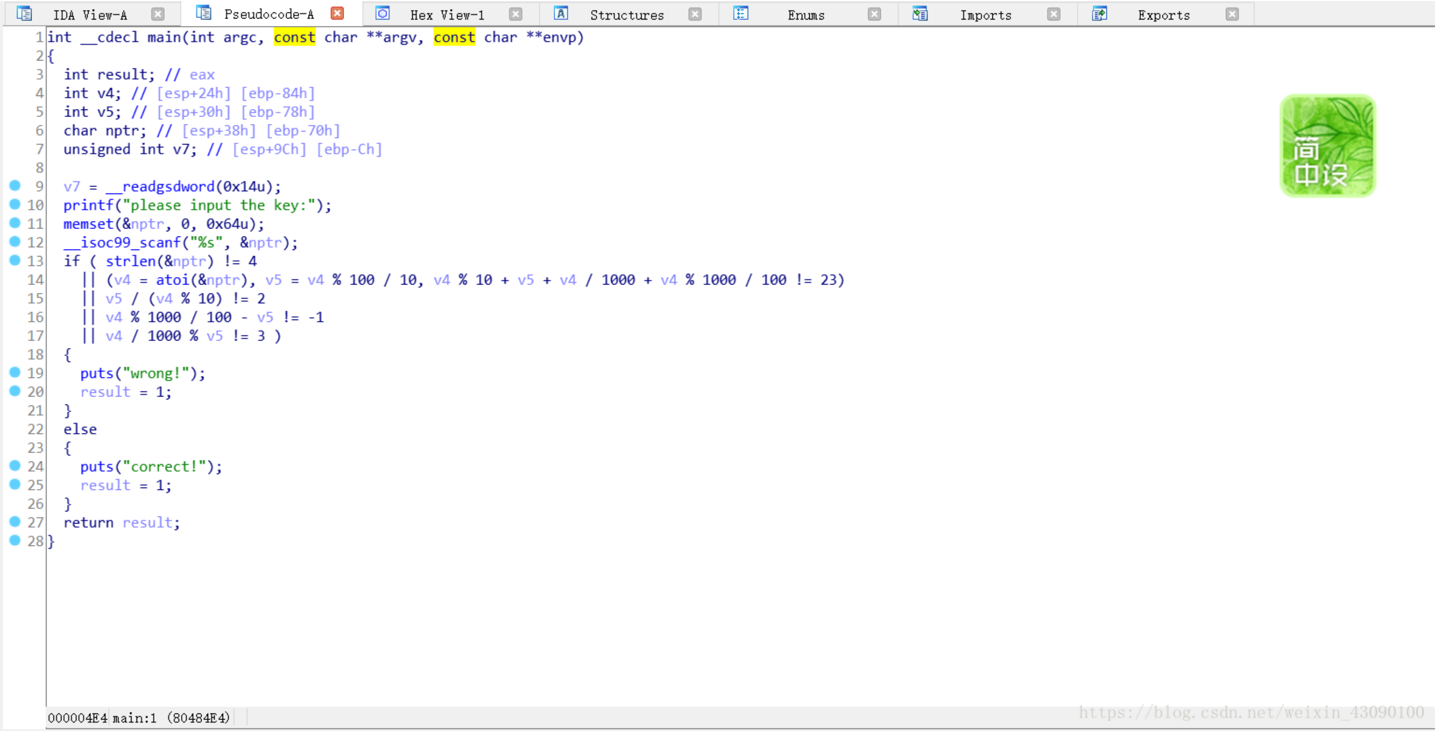
版权

mark一次ctf逆向

一共有两道题：re1和re2

re1签到题

下载附件发现是elf文件，直接上ida，F5之后代码一目了然


IDA View-A
Pseudocode-A
Hex View-1
Structures
Enums
Imports
Exports
000004E4 main:1 (80484E4)
https://blog.csdn.net/weixin_43090100

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int result; // eax
  int v4; // [esp+24h] [ebp-84h]
  int v5; // [esp+30h] [ebp-78h]
  char nptr; // [esp+38h] [ebp-70h]
  unsigned int v7; // [esp+9Ch] [ebp-Ch]

  v7 = __readgsdword(0x14u);
  printf("please input the key:");
  memset(&nptr, 0, 0x64u);
  __isoc99_scanf("%s", &nptr);
  if ( strlen(&nptr) != 4
    || (v4 = atoi(&nptr), v5 = v4 % 100 / 10, v4 % 10 + v5 + v4 / 1000 + v4 % 1000 / 100 != 23)
    || v5 / (v4 % 10) != 2
    || v4 % 1000 / 100 - v5 != -1
    || v4 / 1000 % v5 != 3 )
  {
    puts("wrong!");
    result = 1;
  }
  else
  {
    puts("correct!");
    result = 1;
  }
  return result;
}
```

可以看出来flag是一个四位数，每一位相加是23，十位除个位等于2，百位减十位等于-1，千位模十位等于3，解四元方程求出来就可以了。

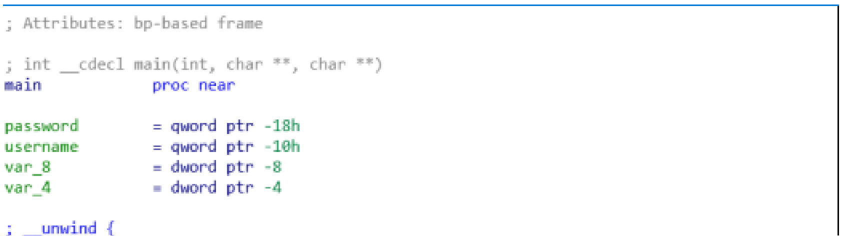
flag是9563

---

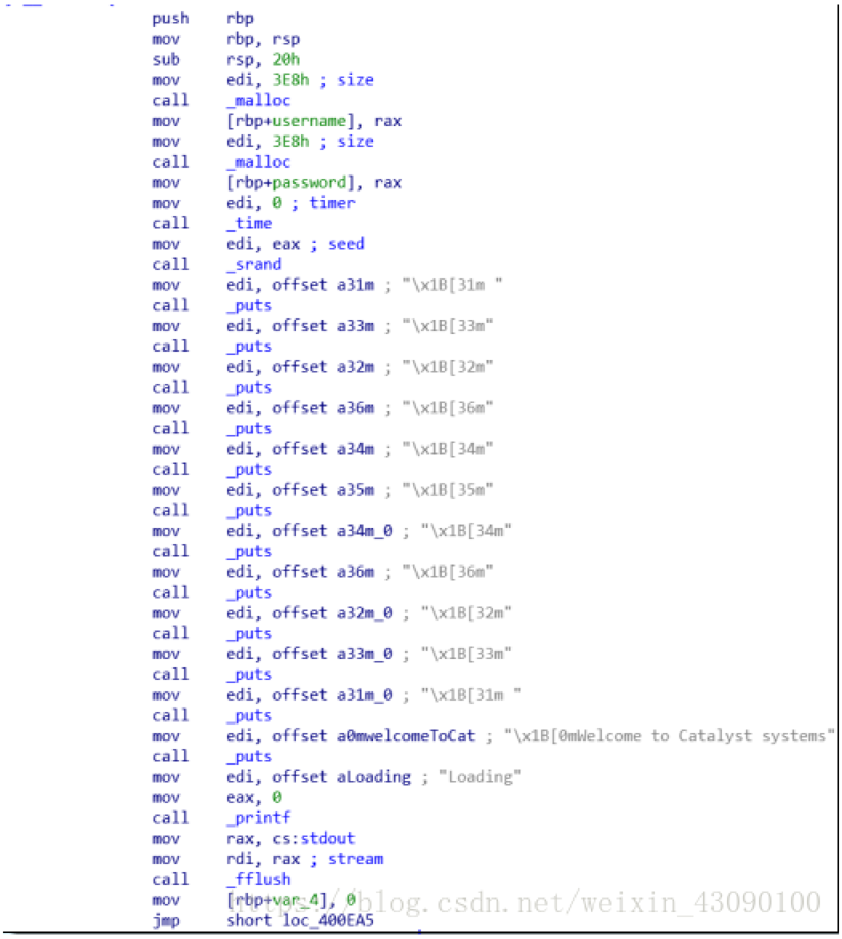
re2

re2有点意思但也没有很复杂，同是elf文件

先定位出main函数，空格转换成图形视图看一下逻辑

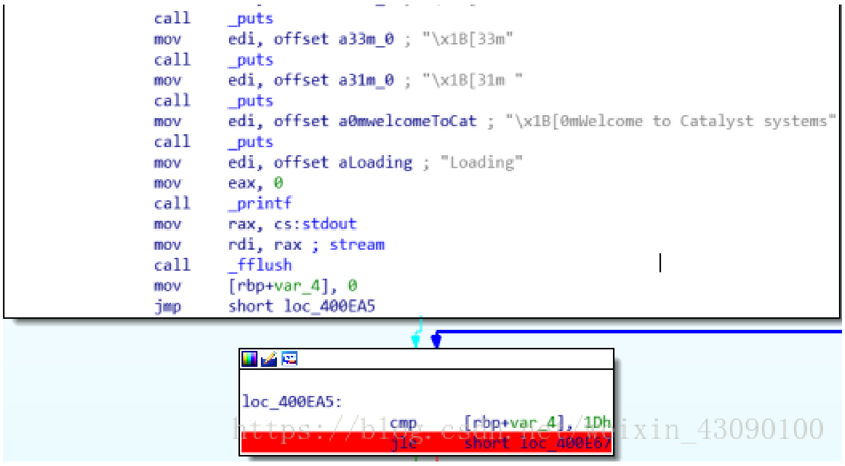


```
; Attributes: bp-based frame

; int __cdecl main(int, char **, char **)
main            proc near

password        = qword ptr -18h
username        = qword ptr -10h
var_8           = dword ptr -8
var_4           = dword ptr -4

; __unwind {
```



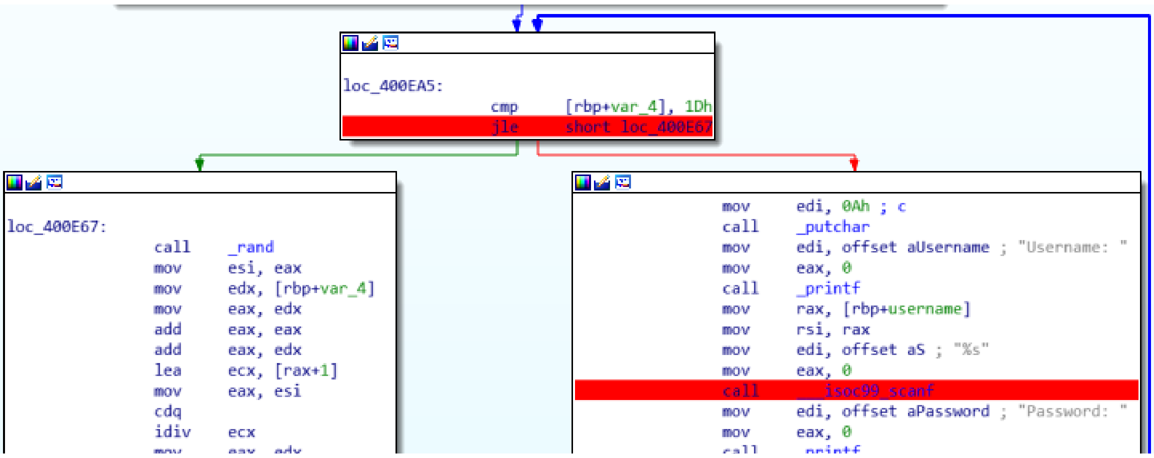
```
push    rbp
mov     rbp, rsp
sub     rsp, 20h
mov     edi, 3E8h ; size
call    _malloc
mov     [rbp+username], rax
mov     edi, 3E8h ; size
call    _malloc
mov     [rbp+password], rax
mov     edi, 0 ; timer
call    _time
mov     edi, eax ; seed
call    _srand
mov     edi, offset a31m ; "\x1B[31m "
call    _puts
mov     edi, offset a33m ; "\x1B[33m"
call    _puts
mov     edi, offset a32m ; "\x1B[32m"
call    _puts
mov     edi, offset a36m ; "\x1B[36m"
call    _puts
mov     edi, offset a34m ; "\x1B[34m"
call    _puts
mov     edi, offset a35m ; "\x1B[35m"
call    _puts
mov     edi, offset a34m_0 ; "\x1B[34m"
call    _puts
mov     edi, offset a36m ; "\x1B[36m"
call    _puts
mov     edi, offset a32m_0 ; "\x1B[32m"
call    _puts
mov     edi, offset a33m_0 ; "\x1B[33m"
call    _puts
mov     edi, offset a31m_0 ; "\x1B[31m "
call    _puts
mov     edi, offset a0mwelcomeToCat ; "\x1B[0mWelcome to Catalyst systems"
call    _puts
mov     edi, offset aLoading ; "Loading"
mov     eax, 0
call    _printf
mov     rax, cs:stdout
mov     rdi, rax ; stream
call    _fflush
mov     [rbp+var_4], 0
jmp     short loc_400EA5
```

前面是一段输出，可以不用管了往下继续看



```
call    _puts
mov     edi, offset a33m_0 ; "\x1B[33m"
call    _puts
mov     edi, offset a31m_0 ; "\x1B[31m "
call    _puts
mov     edi, offset a0mwelcomeToCat ; "\x1B[0mWelcome to Catalyst systems"
call    _puts
mov     edi, offset aLoading ; "Loading"
mov     eax, 0
call    _printf
mov     rax, cs:stdout
mov     rdi, rax ; stream
call    _fflush
mov     [rbp+var_4], 0
jmp     short loc_400EA5

loc_400EA5:
cmp     [rbp+var_4], 1Dh
jle     short loc_400E67
```

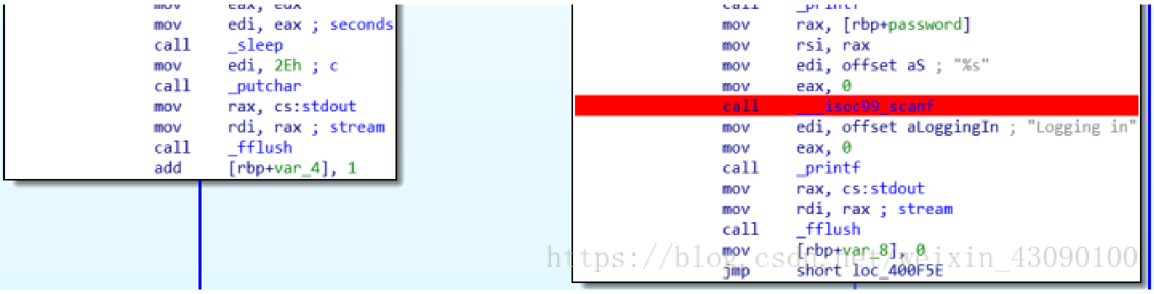
var_4在这是一个循环变量，一共循环1D次



```
loc_400EA5:
cmp     [rbp+var_4], 1Dh
jle     short loc_400E67
```

```
loc_400E67:
call    _rand
mov     esi, eax
mov     edx, [rbp+var_4]
mov     eax, edx
add     eax, eax
add     eax, edx
lea     ecx, [rax+1]
mov     eax, esi
cdq
idiv    ecx
mov     eax, edx
```

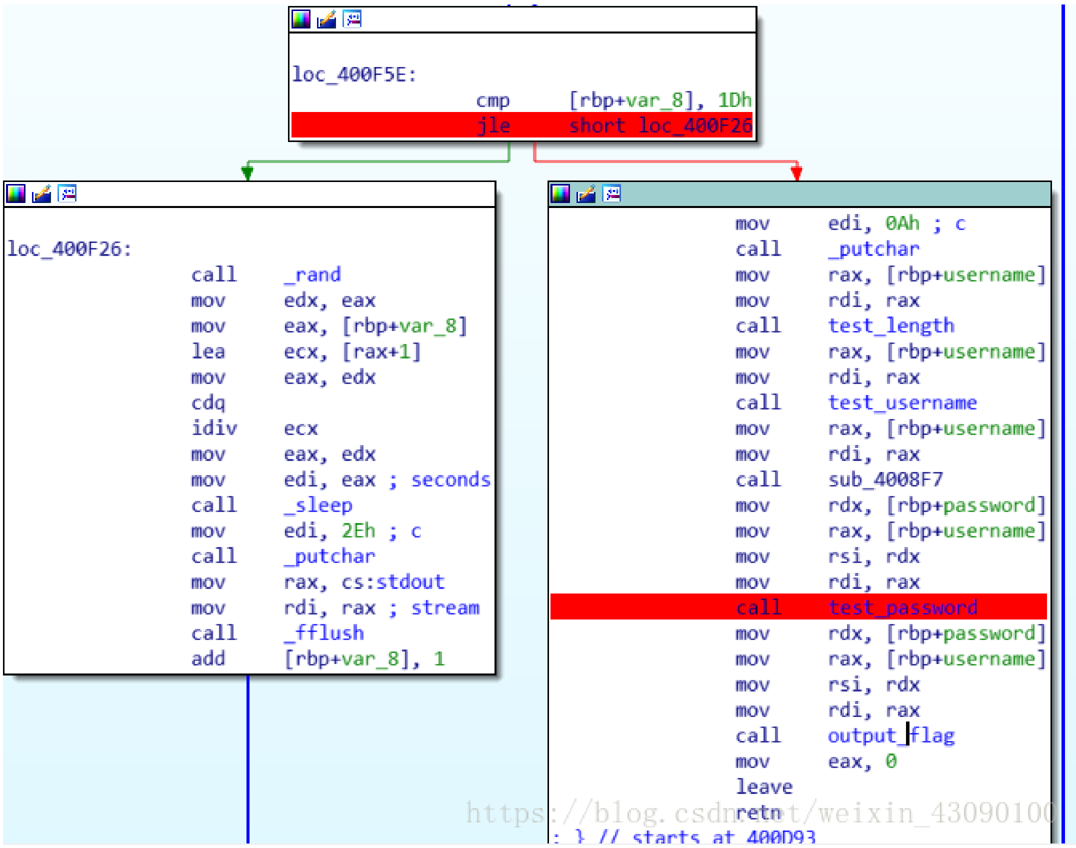
```
mov     edi, 0Ah ; c
call    _putchar
mov     edi, offset aUsername ; "Username: "
mov     eax, 0
call    _printf
mov     rax, [rbp+username]
mov     rsi, rax
mov     edi, offset aS ; "%s"
mov     eax, 0
call    ___isoc99_scanf
mov     edi, offset aPassword ; "Password: "
mov     eax, 0
call    _printf
```



```
mov     eax, edx
mov     edi, eax ; seconds
call    _sleep
mov     edi, 2Eh ; c
call    _putchar
mov     rax, cs:stdout
mov     rdi, rax ; stream
call    _fflush
add     [rbp+var_4], 1
```

```
call    _printf
mov     rax, [rbp+password]
mov     rsi, rax
mov     edi, offset aS ; "%s"
mov     eax, 0
call    ___isoc99_scanf
mov     edi, offset aLoggingIn ; "Logging in"
mov     eax, 0
call    _printf
mov     rax, cs:stdout
mov     rdi, rax ; stream
call    _fflush
mov     [rbp+var_8], 0
jmp     short loc_400F5E
```

结合loading字符串还有sleep、rand函数可以知道这里就是拖延时间，1D次循环之后来到输入用户名和密码的环节，但是当输入完username和password之后发现后面有一个”logging”字符串，猜想可能下面又有熟悉的拖延时间套路…



```
loc_400F5E:
                cmp     [rbp+var_8], 1Dh
                jle     short loc_400F26
```

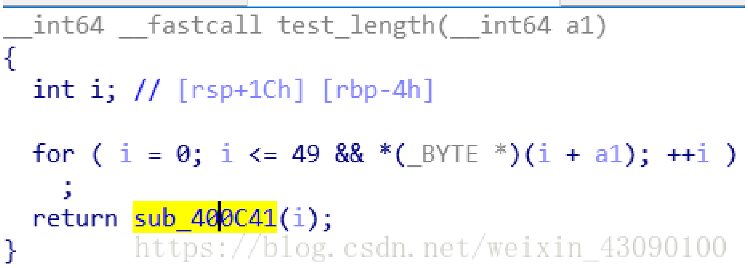
```
loc_400F26:
                call    _rand
                mov     edx, eax
                mov     eax, [rbp+var_8]
                lea     ecx, [rax+1]
                mov     eax, edx
                cdq
                idiv    ecx
                mov     eax, edx
                mov     edi, eax ; seconds
                call    _sleep
                mov     edi, 2Eh ; c
                call    _putchar
                mov     rax, cs:stdout
                mov     rdi, rax ; stream
                call    _fflush
                add     [rbp+var_8], 1
```

```
mov     edi, 0Ah ; c
call    _putchar
mov     rax, [rbp+username]
mov     rdi, rax
call    test_length
mov     rax, [rbp+username]
mov     rdi, rax
call    test_username
mov     rax, [rbp+username]
mov     rdi, rax
call    sub_4008F7
mov     rdx, [rbp+password]
mov     rax, [rbp+username]
mov     rsi, rdx
mov     rdi, rax
call    test_password
mov     rdx, [rbp+password]
mov     rax, [rbp+username]
mov     rsi, rdx
mov     rdi, rax
call    output_flag
mov     eax, 0
leave
retn
; } // starts at 400D93
```

事实证明确实是的。主要就是看时间磨完之后上图右段代码

（ps：*我这里是分析完之后的界面，所以部分变量名和函数名有些改动*）

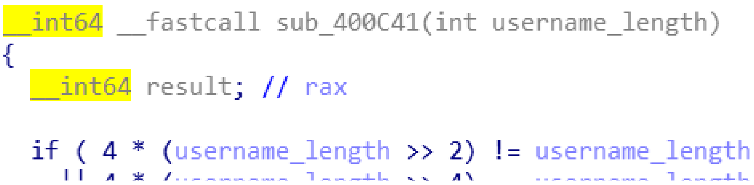
先进入第一个函数看看



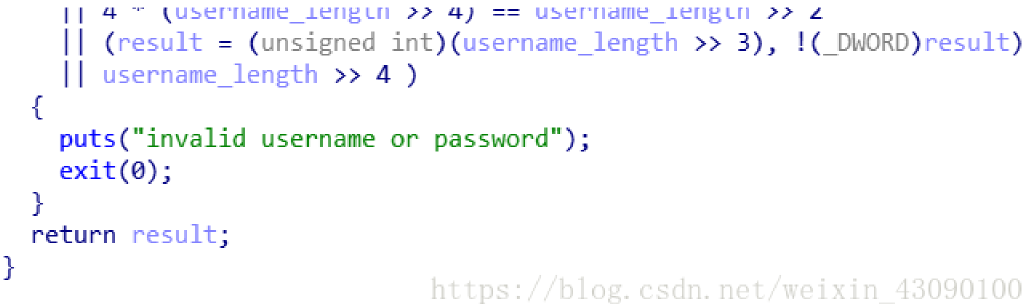
```
__int64 __fastcall test_length(__int64 a1)
{
  int i; // [rsp+1Ch] [rbp-4h]

  for ( i = 0; i <= 49 && *(_BYTE *)(i + a1); ++i )
    ;
  return sub_400C41(i);
}
```

f5之后：

这个没什么限定条件，就是如果你输入的username大于50就当成50字节的来算了…主要还是看循环内的那个函数：



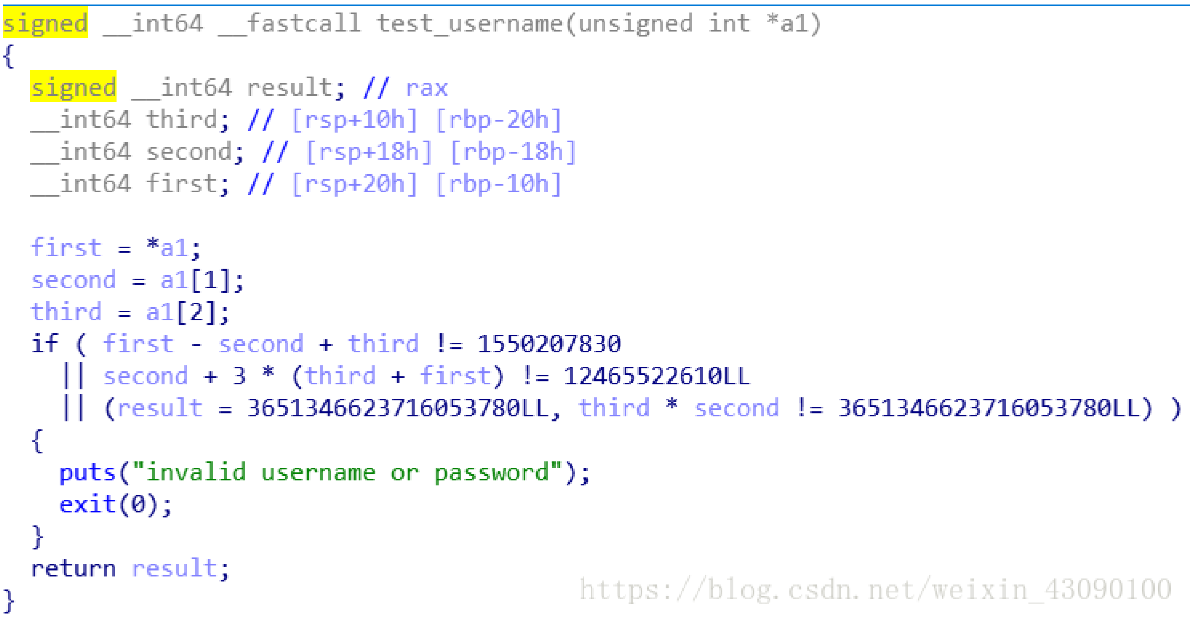
```
__int64 __fastcall sub_400C41(int username_length)
{
  __int64 result; // rax

  if ( 4 * (username_length >> 2) != username_length
```



```
    || 4 * (username_length >> 4) == username_length >> 2
    || (result = (unsigned int)(username_length >> 3), !(_DWORD)result)
    || username_length >> 4 )
  {
    puts("invalid username or password");
    exit(0);
  }
  return result;
}
```

又是熟悉的配方，解多元方程，最后得到username长度是8或者12

退出来看第二个函数

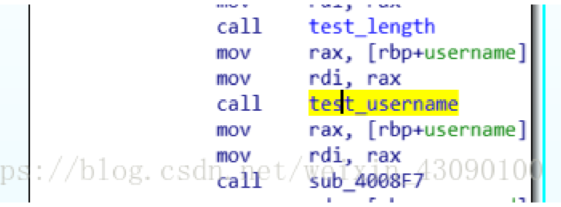


```
signed __int64 __fastcall test_username(unsigned int *a1)
{
  signed __int64 result; // rax
  __int64 third; // [rsp+10h] [rbp-20h]
  __int64 second; // [rsp+18h] [rbp-18h]
  __int64 first; // [rsp+20h] [rbp-10h]

  first = *a1;
  second = a1[1];
  third = a1[2];
  if ( first - second + third != 1550207830
    || second + 3 * (third + first) != 12465522610LL
    || (result = 3651346623716053780LL, third * second != 3651346623716053780LL) )
  {
    puts("invalid username or password");
    exit(0);
  }
  return result;
}
```

解方程+1…把username分成了三个双字，一共是12个字节，分别计算出来first=61746163,second=7473796C,third=6F65635F.但这里应该注意的问题是小端模式是小对小、大对大，将以上转换成字符串的时候不要忘了每四个字节倒序输出。

最后得到username：**catalyst_ceo**

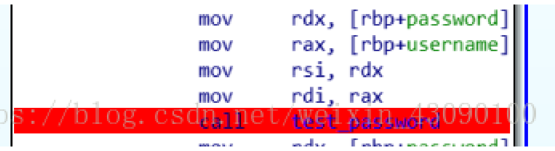
再退出来



```
mov     rdi, rax
call    test_length
mov     rax, [rbp+username]
mov     rdi, rax
call    test_username
mov     rax, [rbp+username]
mov     rdi, rax
call    sub_4008F7
```

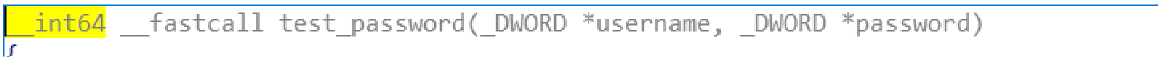
可以发现第三个函数的参数也是username，推测应该是对username的再验证，所以并没有再去分析，而且动态调试的时候这里也没结束进程。

进入第四个函数

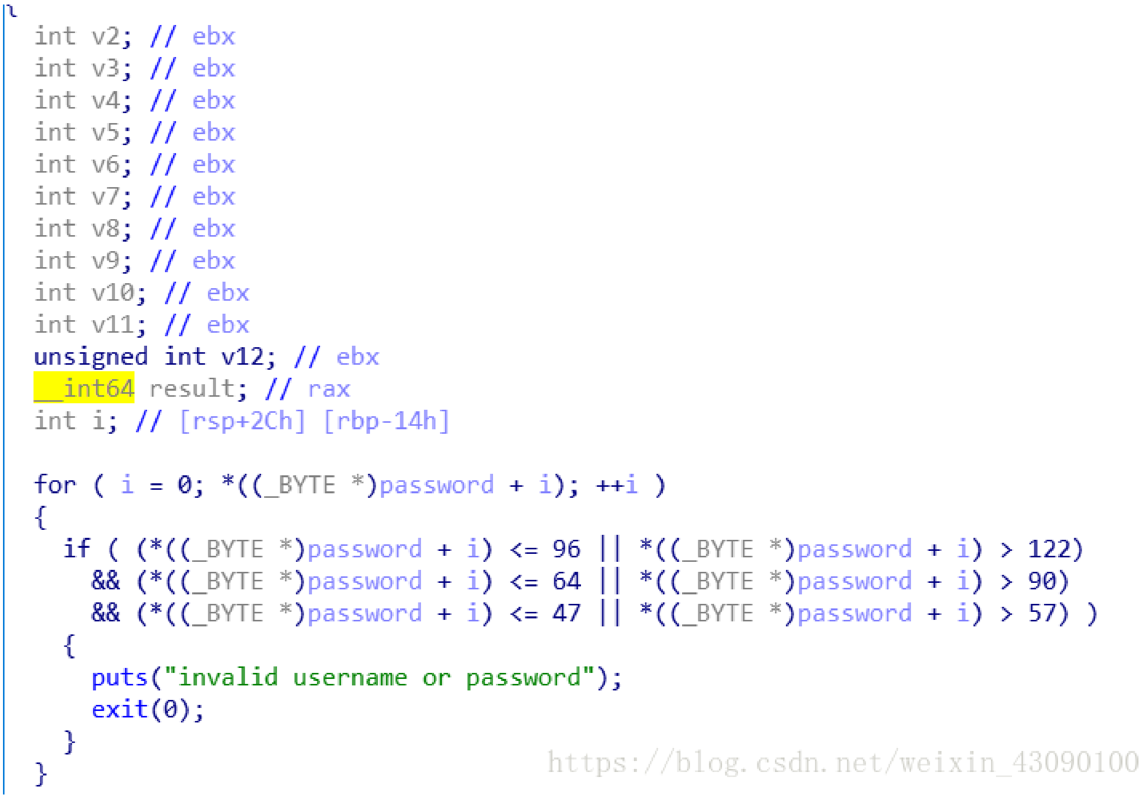


```
mov     rdx, [rbp+password]
mov     rax, [rbp+username]
mov     rsi, rdx
mov     rdi, rax
call    test_password
mov     rdx, [rbp+password]
```

这里两个参数分别是username和password，应该这里就是验证password部分了，进去f5：

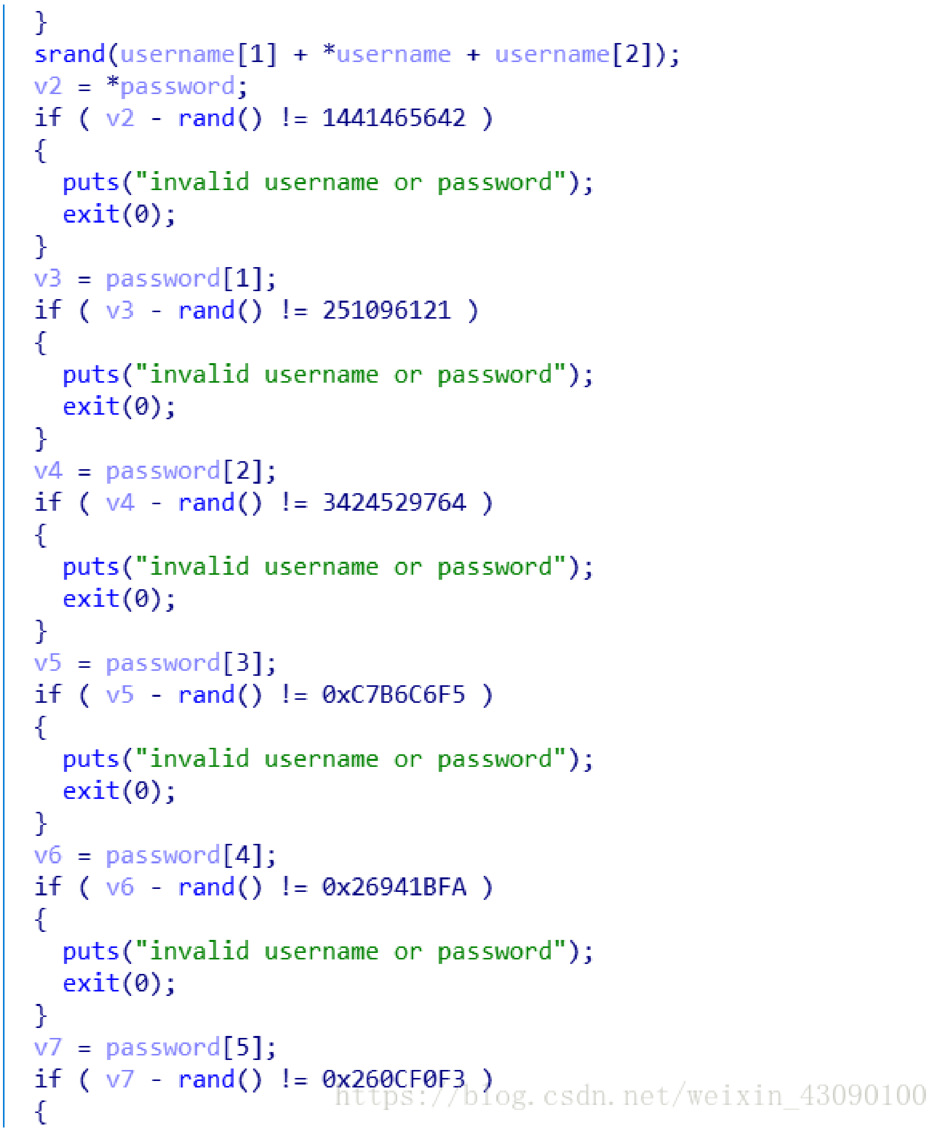


```
__int64 __fastcall test_password(_DWORD *username, _DWORD *password)
{
```



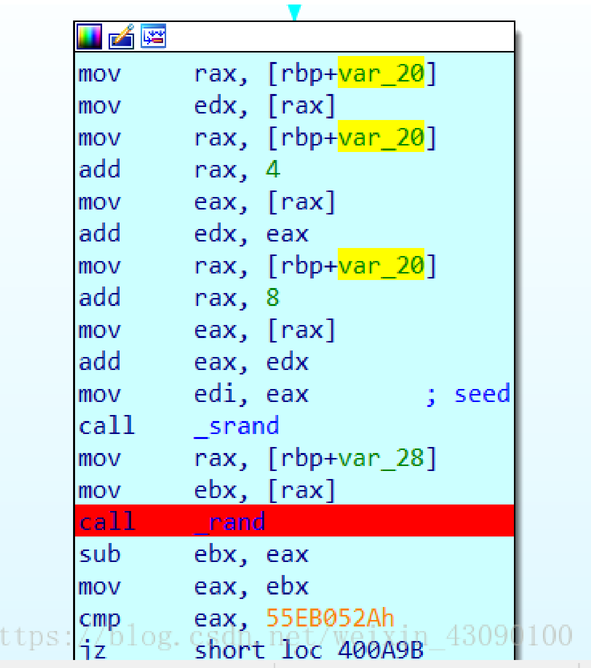
```
{
  int v2; // ebx
  int v3; // ebx
  int v4; // ebx
  int v5; // ebx
  int v6; // ebx
  int v7; // ebx
  int v8; // ebx
  int v9; // ebx
  int v10; // ebx
  int v11; // ebx
  unsigned int v12; // ebx
  __int64 result; // rax
  int i; // [rsp+2Ch] [rbp-14h]

  for ( i = 0; *((_BYTE *)password + i); ++i )
  {
    if ( (*((_BYTE *)password + i) <= 96 || *((_BYTE *)password + i) > 122)
      && (*((_BYTE *)password + i) <= 64 || *((_BYTE *)password + i) > 90)
      && (*((_BYTE *)password + i) <= 47 || *((_BYTE *)password + i) > 57) )
    {
      puts("invalid username or password");
      exit(0);
    }
  }
```

前面这一段循环是检验password每个字符的ascii码范围，接着往下看：



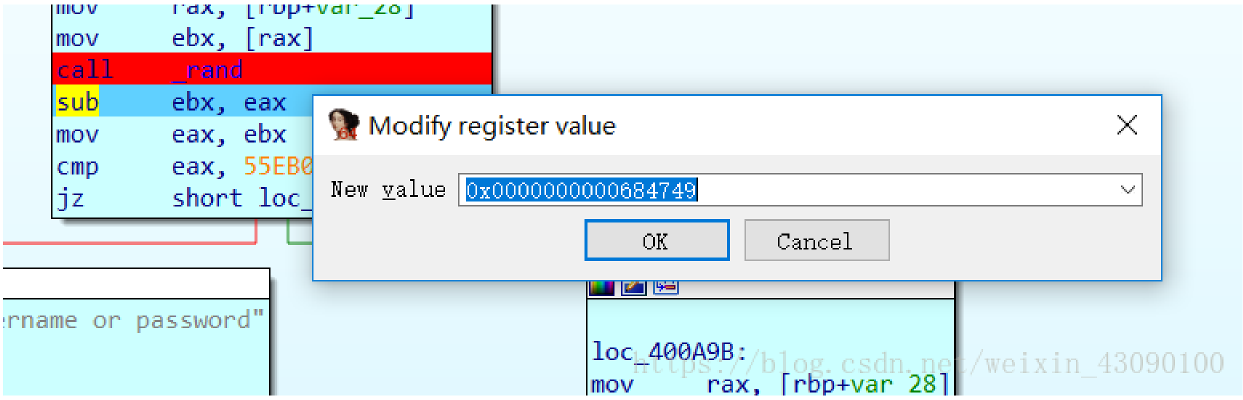
```
  }
  srand(username[1] + *username + username[2]);
  v2 = *password;
  if ( v2 - rand() != 1441465642 )
  {
    puts("invalid username or password");
    exit(0);
  }
  v3 = password[1];
  if ( v3 - rand() != 251096121 )
  {
    puts("invalid username or password");
    exit(0);
  }
  v4 = password[2];
  if ( v4 - rand() != 3424529764 )
  {
    puts("invalid username or password");
    exit(0);
  }
  v5 = password[3];
  if ( v5 - rand() != 0xC7B6C6F5 )
  {
    puts("invalid username or password");
    exit(0);
  }
  v6 = password[4];
  if ( v6 - rand() != 0x26941BFA )
  {
    puts("invalid username or password");
    exit(0);
  }
  v7 = password[5];
  if ( v7 - rand() != 0x260CF0F3 )
  {
```

这里有一个srand和rand函数的搭配，接着往下是一共10个双字的验证，每个双字都等于rand()+k(k是常数)，因为我并不知道rand和srand具体关系所以我选择去动态调试，再次转换成汇编，用remote linux debugger调试，再rand函数上下断点



```
mov     rax, [rbp+var_20]
mov     edx, [rax]
mov     rax, [rbp+var_20]
add     rax, 4
mov     eax, [rax]
add     edx, eax
mov     rax, [rbp+var_20]
add     rax, 8
mov     eax, [rax]
add     eax, edx
mov     edi, eax        ; seed
call    _srand
mov     rax, [rbp+var_28]
mov     ebx, [rax]
call    _rand
sub     ebx, eax
mov     eax, ebx
cmp     eax, 55EB052Ah
jz      short loc_400A9B
```

这里rand函数返回值是保存在eax当中，f8步过rand函数查看eax的值



```
mov     rax, [rbp+var_28]
mov     ebx, [rax]
call    _rand
sub     ebx, eax
mov     eax, ebx
cmp     eax, 55EB0
jz      short loc_
```

```
rname or password"
```

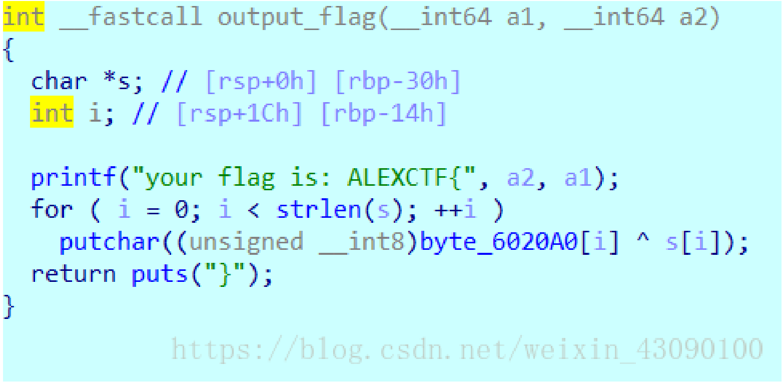
```
loc_400A9B:
mov     rax, [rbp+var_28]
```

Modify register value

New value 0x0000000000684749

用当前eax的值加上cmp的后一个操作数就得出来当前双字的具体值，对于后面9个双字也是同样的操作，同样注意小端模式，得到的password是**sLSVpQ4vK3cGWyW86AiZhggwLHBjmx9CRspVGggj**

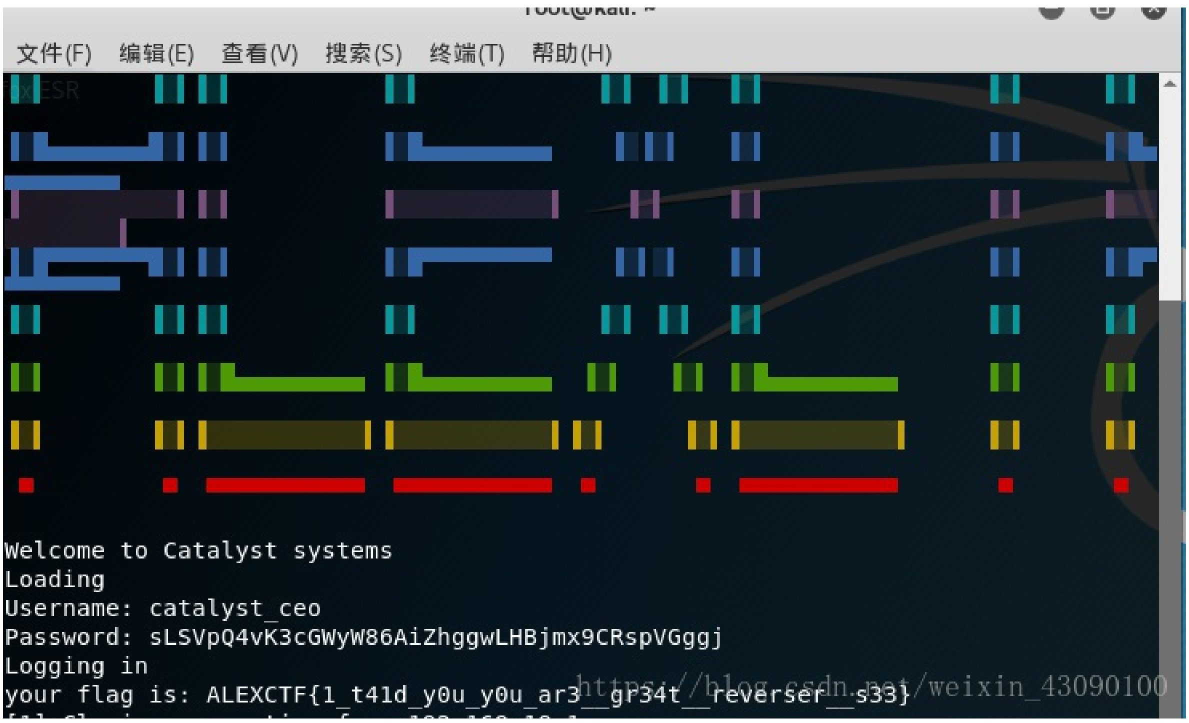
退出来再看最后一个函数



```
int __fastcall output_flag(__int64 a1, __int64 a2)
{
  char *s; // [rsp+0h] [rbp-30h]
  int i; // [rsp+1Ch] [rbp-14h]

  printf("your flag is: ALEXCTF{", a2, a1);
  for ( i = 0; i < strlen(s); ++i )
    putchar((unsigned __int8)byte_6020A0[i] ^ s[i]);
  return puts("}");
}
```

两个参数分别是username和password，可以看出来这里根据username和password输出flag，因为我们的username和password都是正确的所以这个函数也就跳过分析，动态调试直接f9运行完，最后成果图如下：

root@kali: ~



```
Welcome to Catalyst systems
Loading
Username: catalyst_ceo
Password: sLSVpQ4vK3cGWyW86AiZhggwLHBjmx9CRspVGggj
Logging in
your flag is: ALEXCTF{1_t41d_y0u_y0u_ar3__gr34t__reverser__s33}
```

最后的flag:**ALEXCTF{1_t41d_y0u_y0u_ar3__gr34t__reverser__s33}**