攻防世界Reverse进阶区-simple-check-100-writeup

原创

y4ung 于 2020-10-28 15:25:09 发布 144 收藏 1
分类专栏： ctf 文章标签： ctf

ctf 专栏收录该内容
35 篇文章 0 订阅
订阅专栏

1. 介绍

本题是xctf攻防世界中Reverse的进阶区的题simple-check-100

题目来源： school-ctf-winter-2015

题目提供了三个文件：

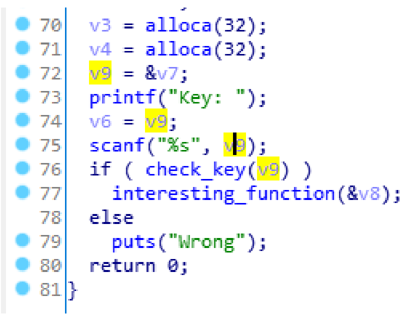
1. task9_x86_ed82b6faaf979658e040c77422d01b1b3db183f7.exe => windows下的可执行程序
2. task9_x86_2fb0b7e96f097597851f24faaf664fdb20ad8b8a => Linux 32bit
3. task9_x86_64_46d01fe312d35ecf69c4ff8ab8ace75d080891dc => Linux 64bit

2. 分析

2.1 静态分析

```
$ file task9_x86_ed82b6faaf979658e040c77422d01b1b3db183f7.exe
task9_x86_ed82b6faaf979658e040c77422d01b1b3db183f7.exe: PE32 executable (console)
Intel 80386 (stripped to external PDB), for MS Windows
# windows
task9_x86_ed82b6faaf979658e040c77422d01b1b3db183f7.exe
Key: 123
# 退出命令行窗口
```

`Key:` 在main函数中使用。在main函数中，用户输入赋值给v9，然后调用check_key函数



```
  v3 = alloca(32);
  v4 = alloca(32);
  v9 = &v7;
  printf("Key: ");
  v6 = v9;
  scanf("%s", v9);
  if ( check_key(v9) )
    interesting_function(&v8);
  else
    puts("Wrong");
  return 0;
}
```

check_key函数中，每次循环中，v3等于v3加上`*(4 * i + a1)`，其中`*(4 * i + a1)`就是将`4 * i + a1`的值作为地址，从该地址里取数据。
那么我们需要从最终的数值 -559038737，也就是0xDEADBEEF入手，反着推回去。
emm好像有点难度啊。这个v3是`[a1], [4+a1], [8+a1], [12+a1], [16+a1]`值作为地址，地址中数据的和，[16+a1]表示a1[16]。这里的a1是用户的输入。如果要逆的话，需要找到内存中这么一块连续的地址，地址里的值作为地址所取到的值必须满足条件才行。

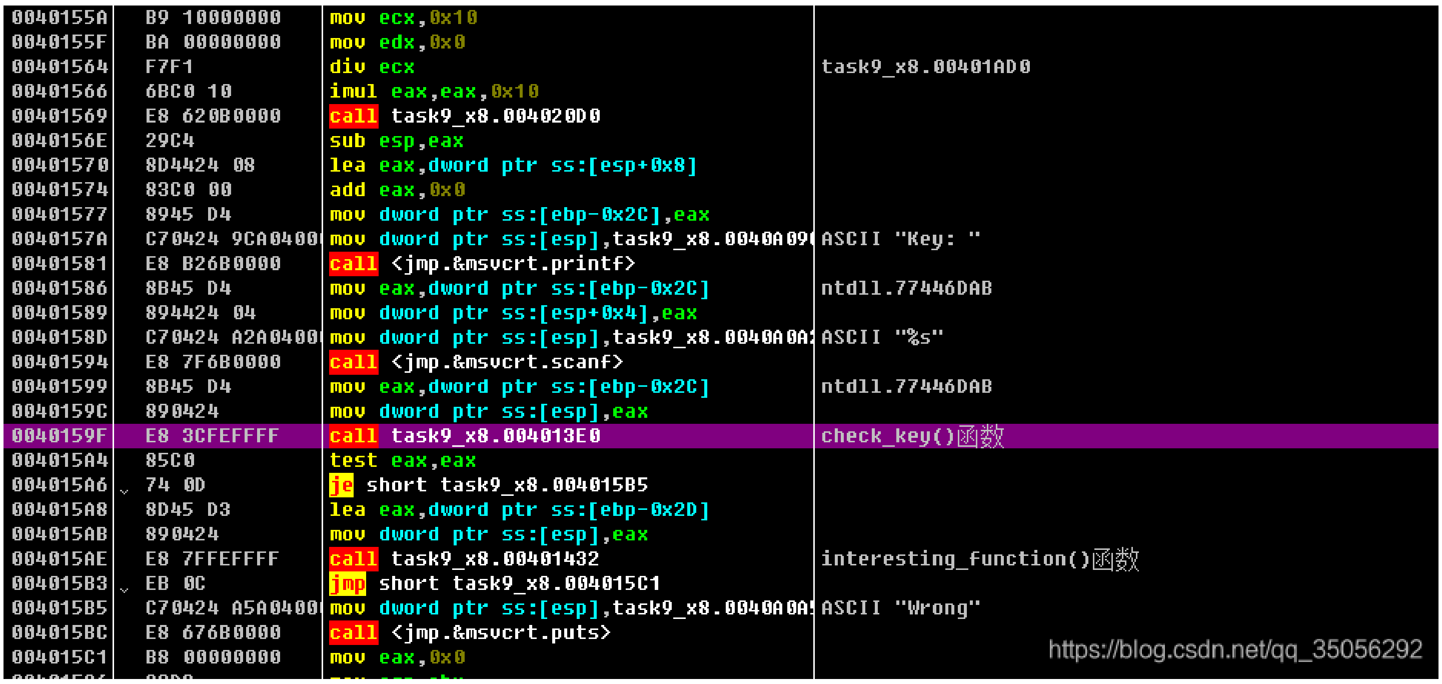
```
BOOL __cdecl check_key(int a1)
{
    signed int i; // [esp+8h] [ebp-8h]
    int v3; // [esp+Ch] [ebp-4h]

    v3 = 0;
    for ( i = 0; i <= 4; ++i )     # [0, 4]
        v3 += *(_DWORD *)(4 * i + a1);
    return v3 == -559038737;
}
```

2.2 动态分析

真的非逆它不可吗？不是的。
想想main函数，只有当check_key函数返回值为1时，才会进入interesting_function函数，而这个函数应该就是打印flag的函数了。

因此我们的思路其实可以变成，随便输入一个数据，在check_key函数之后修改函数的返回值（保存在eax寄存器中，修改寄存器中的值），进入到调用interesting_function函数的块中，到时候运行完看结果即可。

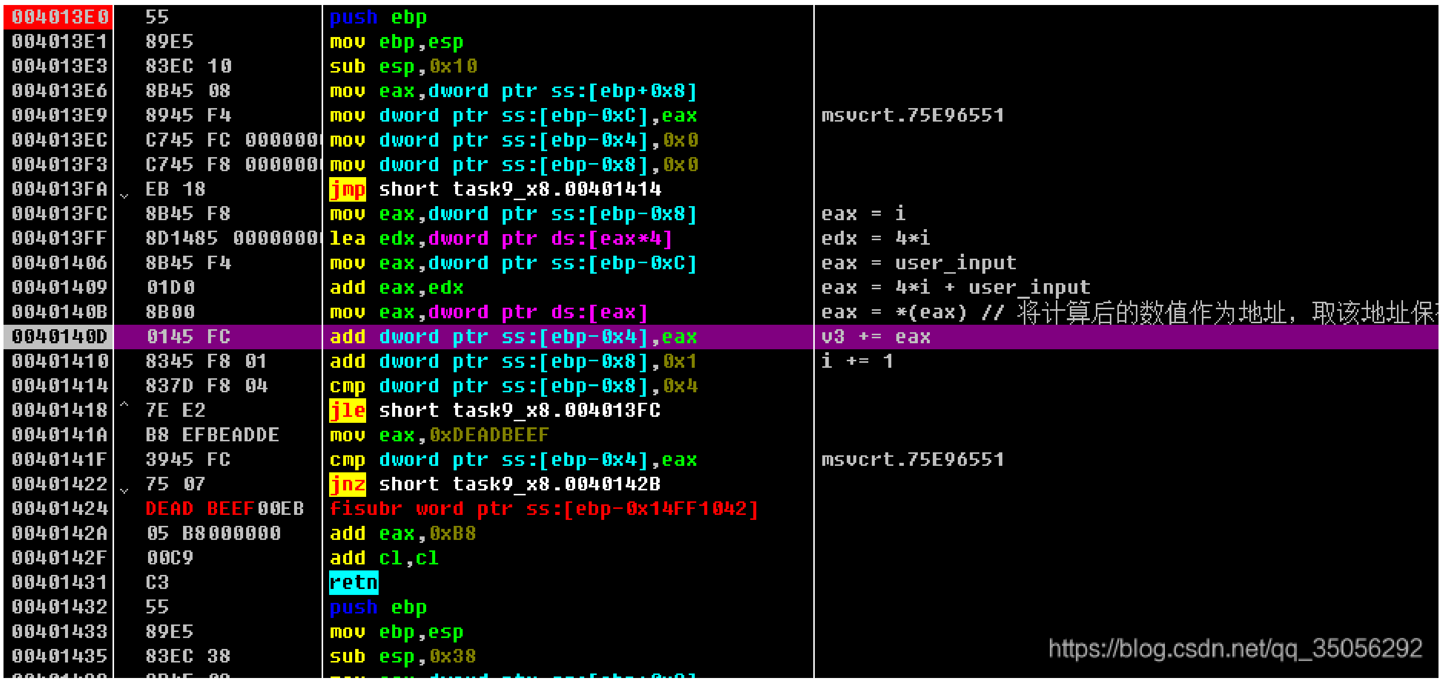
打开ollydbg。`查找 => 所有参考文本字串`，找到`Key:`即可找到main函数。然后在call scanf下面的两条指令即为check_key函数。



```
0040155A    B9 10000000     mov ecx,0x10
0040155F    BA 00000000     mov edx,0x0
00401564    F7F1            div ecx                                     task9_x8.00401AD0
00401566    6BC0 10         imul eax,eax,0x10
00401569    E8 620B0000     call task9_x8.004020D0
0040156E    29C4            sub esp,eax
00401570    8D4424 08       lea eax,dword ptr ss:[esp+0x8]
00401574    83C0 00         add eax,0x0
00401577    8945 D4         mov dword ptr ss:[ebp-0x2C],eax
0040157A    C70424 9CA0400  mov dword ptr ss:[esp],task9_x8.0040A09     ASCII "Key: "
00401581    E8 B26B0000     call <jmp.&msvcrt.printf>
00401586    8B45 D4         mov eax,dword ptr ss:[ebp-0x2C]             ntdll.77446DAB
00401589    894424 04       mov dword ptr ss:[esp+0x4],eax
0040158D    C70424 A2A0400  mov dword ptr ss:[esp],task9_x8.0040A0A     ASCII "%s"
00401594    E8 7F6B0000     call <jmp.&msvcrt.scanf>
00401599    8B45 D4         mov eax,dword ptr ss:[ebp-0x2C]             ntdll.77446DAB
0040159C    890424          mov dword ptr ss:[esp],eax
0040159F    E8 3CFEFFFF     call task9_x8.004013E0                      check_key()函数
004015A4    85C0            test eax,eax
004015A6  ⌄ 74 0D           je short task9_x8.004015B5
004015A8    8D45 D3         lea eax,dword ptr ss:[ebp-0x2D]
004015AB    890424          mov dword ptr ss:[esp],eax
004015AE    E8 7FFEFFFF     call task9_x8.00401432                      interesting_function()函数
004015B3  ⌄ EB 0C           jmp short task9_x8.004015C1
004015B5    C70424 A5A0400  mov dword ptr ss:[esp],task9_x8.0040A0A     ASCII "Wrong"
004015BC    E8 676B0000     call <jmp.&msvcrt.puts>
004015C1    B8 00000000     mov eax,0x0
```

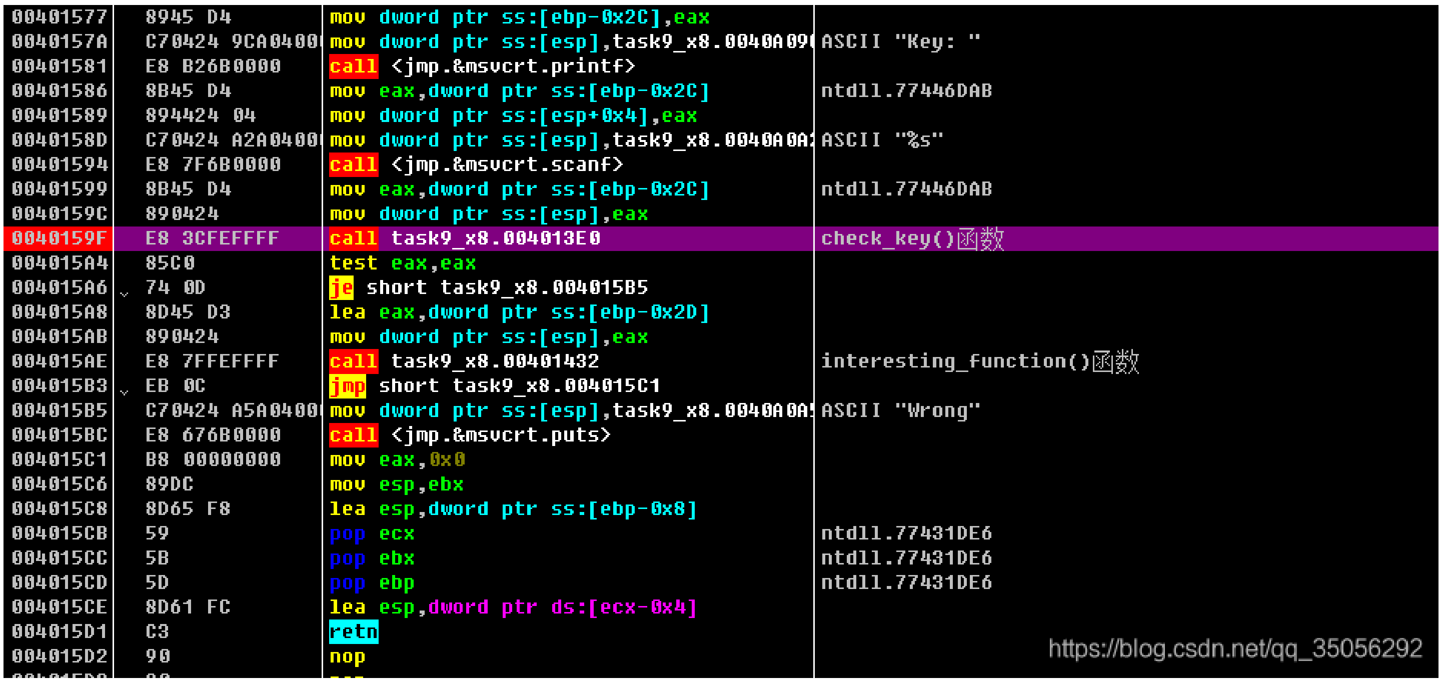
> 这里为check_key函数的汇编理解：

ctrl + g，输入check_key函数的地址：0x004013E0，跳到该函数。在函数一开始时下个断点。接下来按F9，运行程序，直到遇到断点。



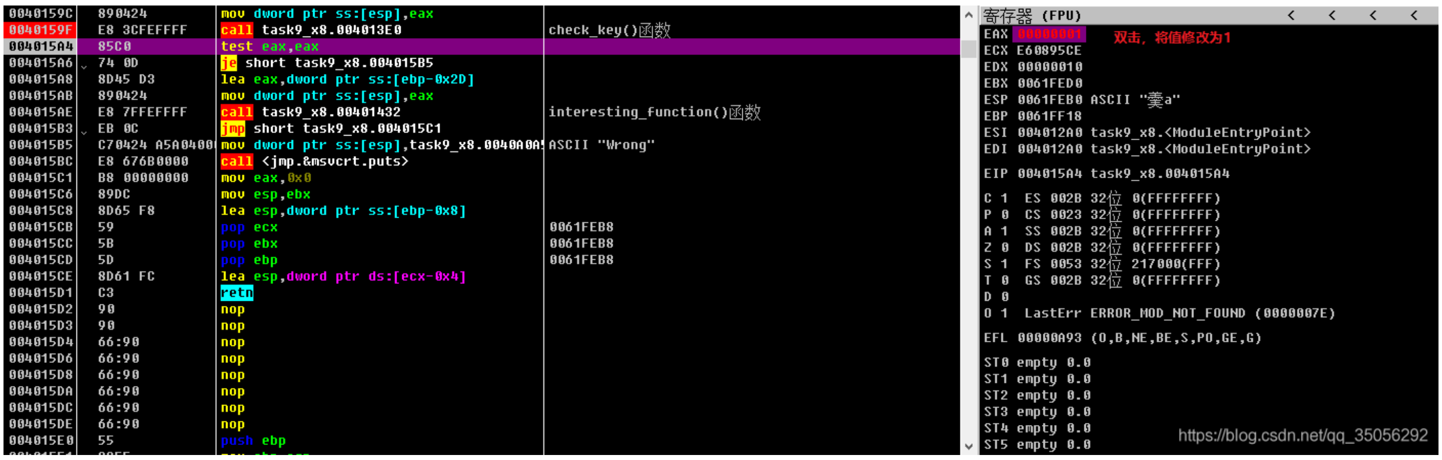
```
004013E0     55            push ebp
004013E1     89E5          mov ebp,esp
004013E3     83EC 10       sub esp,0x10
004013E6     8B45 08       mov eax,dword ptr ss:[ebp+0x8]
004013E9     8945 F4       mov dword ptr ss:[ebp-0xC],eax              msvcrt.75E96551
004013EC     C745 FC 000000 mov dword ptr ss:[ebp-0x4],0x0
004013F3     C745 F8 000000 mov dword ptr ss:[ebp-0x8],0x0
004013FA  ⌄  EB 18         jmp short task9_x8.00401414
004013FC     8B45 F8       mov eax,dword ptr ss:[ebp-0x8]               eax = i
004013FF     8D1485 0000000 lea edx,dword ptr ds:[eax*4]                edx = 4*i
00401406     8B45 F4       mov eax,dword ptr ss:[ebp-0xC]               eax = user_input
00401409     01D0          add eax,edx                                  eax = 4*i + user_input
0040140B     8B00          mov eax,dword ptr ds:[eax]                   eax = *(eax) // 将计算后的数值作为地址，取该地址保
0040140D     0145 FC       add dword ptr ss:[ebp-0x4],eax               v3 += eax
00401410     8345 F8 01    add dword ptr ss:[ebp-0x8],0x1               i += 1
00401414     837D F8 04    cmp dword ptr ss:[ebp-0x8],0x4
00401418  ^  7E E2         jle short task9_x8.004013FC
0040141A     B8 EFBEADDE   mov eax,0xDEADBEEF
0040141F     3945 FC       cmp dword ptr ss:[ebp-0x4],eax               msvcrt.75E96551
00401422  ⌄  75 07         jnz short task9_x8.0040142B
00401424     DEAD BEEF00EB fisubr word ptr ss:[ebp-0x14FF1042]
0040142A     05 B8000000   add eax,0xB8
0040142F     00C9          add cl,cl
00401431     C3            retn
00401432     55            push ebp
00401433     89E5          mov ebp,esp
00401435     83EC 38       sub esp,0x38
```

ollydbg中在call check_key时下断点，然后运行到这里时，直接按F8 Step over过该函数，



```
00401577     8945 D4       mov dword ptr ss:[ebp-0x2C],eax
0040157A     C70424 9CA0400 mov dword ptr ss:[esp],task9_x8.0040A09( ASCII "Key: "
00401581     E8 B26B0000   call <jmp.&msvcrt.printf>
00401586     8B45 D4       mov eax,dword ptr ss:[ebp-0x2C]              ntdll.77446DAB
00401589     894424 04     mov dword ptr ss:[esp+0x4],eax
0040158D     C70424 A2A0400 mov dword ptr ss:[esp],task9_x8.0040A0A: ASCII "%s"
00401594     E8 7F6B0000   call <jmp.&msvcrt.scanf>
00401599     8B45 D4       mov eax,dword ptr ss:[ebp-0x2C]              ntdll.77446DAB
0040159C     890424        mov dword ptr ss:[esp],eax
0040159F     E8 3CFEFFFF   call task9_x8.004013E0                       check_key()函数
004015A4     85C0          test eax,eax
004015A6  ⌄  74 0D         je short task9_x8.004015B5
004015A8     8D45 D3       lea eax,dword ptr ss:[ebp-0x2D]
004015AB     890424        mov dword ptr ss:[esp],eax
004015AE     E8 7FFEFFFF   call task9_x8.00401432                       interesting_function()函数
004015B3  ⌄  EB 0C         jmp short task9_x8.004015C1
004015B5     C70424 A5A0400 mov dword ptr ss:[esp],task9_x8.0040A0A! ASCII "Wrong"
004015BC     E8 676B0000   call <jmp.&msvcrt.puts>
004015C1     B8 00000000   mov eax,0x0
004015C6     89DC          mov esp,ebx
004015C8     8D65 F8       lea esp,dword ptr ss:[ebp-0x8]
004015CB     59            pop ecx                                      ntdll.77431DE6
004015CC     5B            pop ebx                                      ntdll.77431DE6
004015CD     5D            pop ebp                                      ntdll.77431DE6
004015CE     8D61 FC       lea esp,dword ptr ds:[ecx-0x4]
004015D1     C3            retn
004015D2     90            nop
```

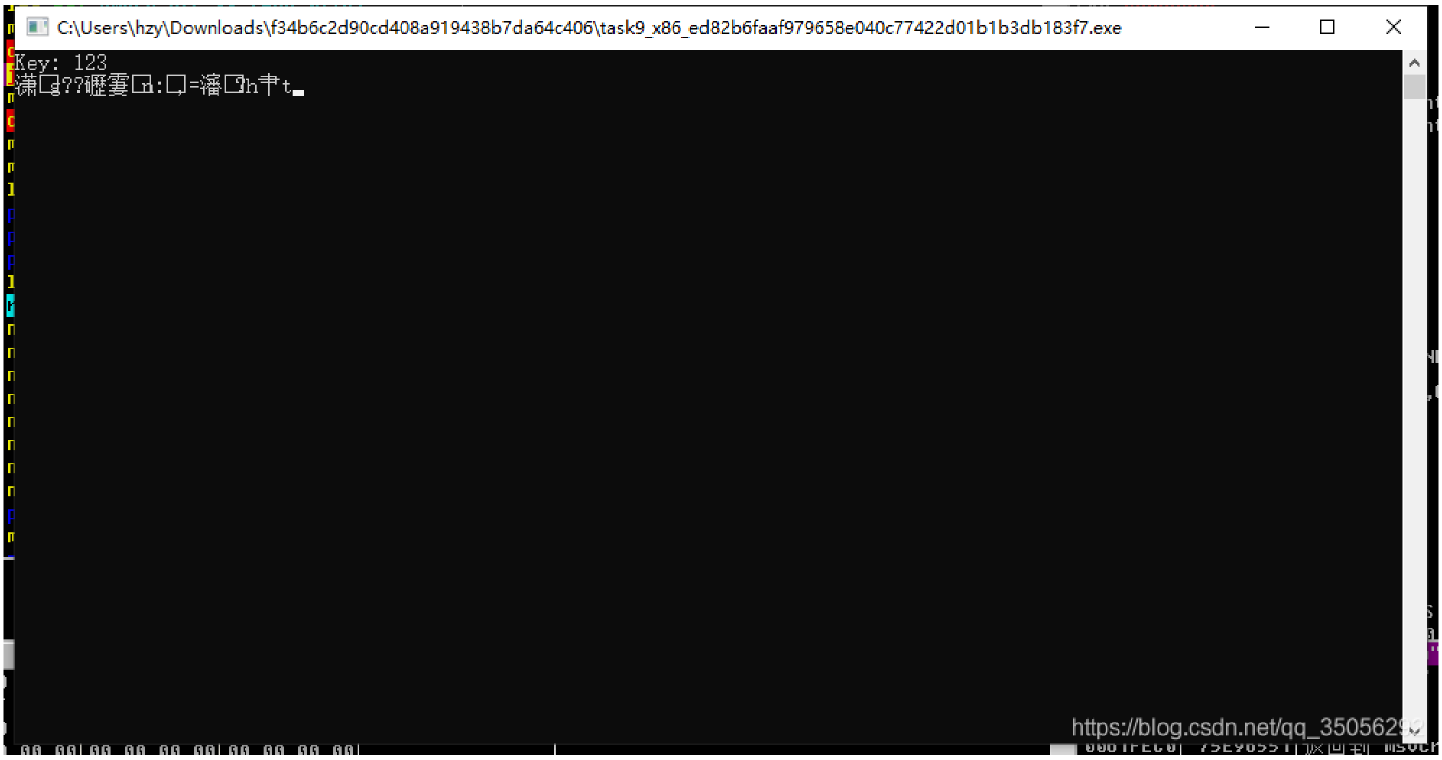
执行完check_key函数后，修改EAX寄存器的值为1。



```
0040159C     890424        mov dword ptr ss:[esp],eax
0040159F     E8 3CFEFFFF   call task9_x8.004013E0                       check_key()函数
004015A4     85C0          test eax,eax
004015A6  ⌄  74 0D         je short task9_x8.004015B5
004015A8     8D45 D3       lea eax,dword ptr ss:[ebp-0x2D]
004015AB     890424        mov dword ptr ss:[esp],eax
004015AE     E8 7FFEFFFF   call task9_x8.00401432                       interesting_function()函数
004015B3  ⌄  EB 0C         jmp short task9_x8.004015C1
004015B5     C70424 A5A0400 mov dword ptr ss:[esp],task9_x8.0040A0A! ASCII "Wrong"
004015BC     E8 676B0000   call <jmp.&msvcrt.puts>
004015C1     B8 00000000   mov eax,0x0
004015C6     89DC          mov esp,ebx
004015C8     8D65 F8       lea esp,dword ptr ss:[ebp-0x8]
004015CB     59            pop ecx                                      0061FEB8
004015CC     5B            pop ebx                                      0061FEB8
004015CD     5D            pop ebp                                      0061FEB8
004015CE     8D61 FC       lea esp,dword ptr ds:[ecx-0x4]
004015D1     C3            retn
004015D2     90            nop
004015D3     90            nop
004015D4     66:90         nop
004015D6     66:90         nop
004015D8     66:90         nop
004015DA     66:90         nop
004015DC     66:90         nop
004015DE     66:90         nop
004015E0     55            push ebp
```

```
寄存器 (FPU)
EAX 00000001     双击，将值修改为1
ECX E60895CE
EDX 00000010
EBX 0061FED0
ESP 0061FEB0 ASCII "羹a"
EBP 0061FF18
ESI 004012A0 task9_x8.<ModuleEntryPoint>
EDI 004012A0 task9_x8.<ModuleEntryPoint>

EIP 004015A4 task9_x8.004015A4

C 1  ES 002B 32位 0(FFFFFFFF)
P 0  CS 0023 32位 0(FFFFFFFF)
A 1  SS 002B 32位 0(FFFFFFFF)
Z 0  DS 002B 32位 0(FFFFFFFF)
S 1  FS 0053 32位 217000(FFF)
T 0  GS 002B 32位 0(FFFFFFFF)
D 0
O 1  LastErr ERROR_MOD_NOT_FOUND (0000007E)

EFL 00000A93 (O,B,NE,BE,S,PO,GE,G)

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
```

但是最后输出了一堆乱码。。。(▼へ▼メ)



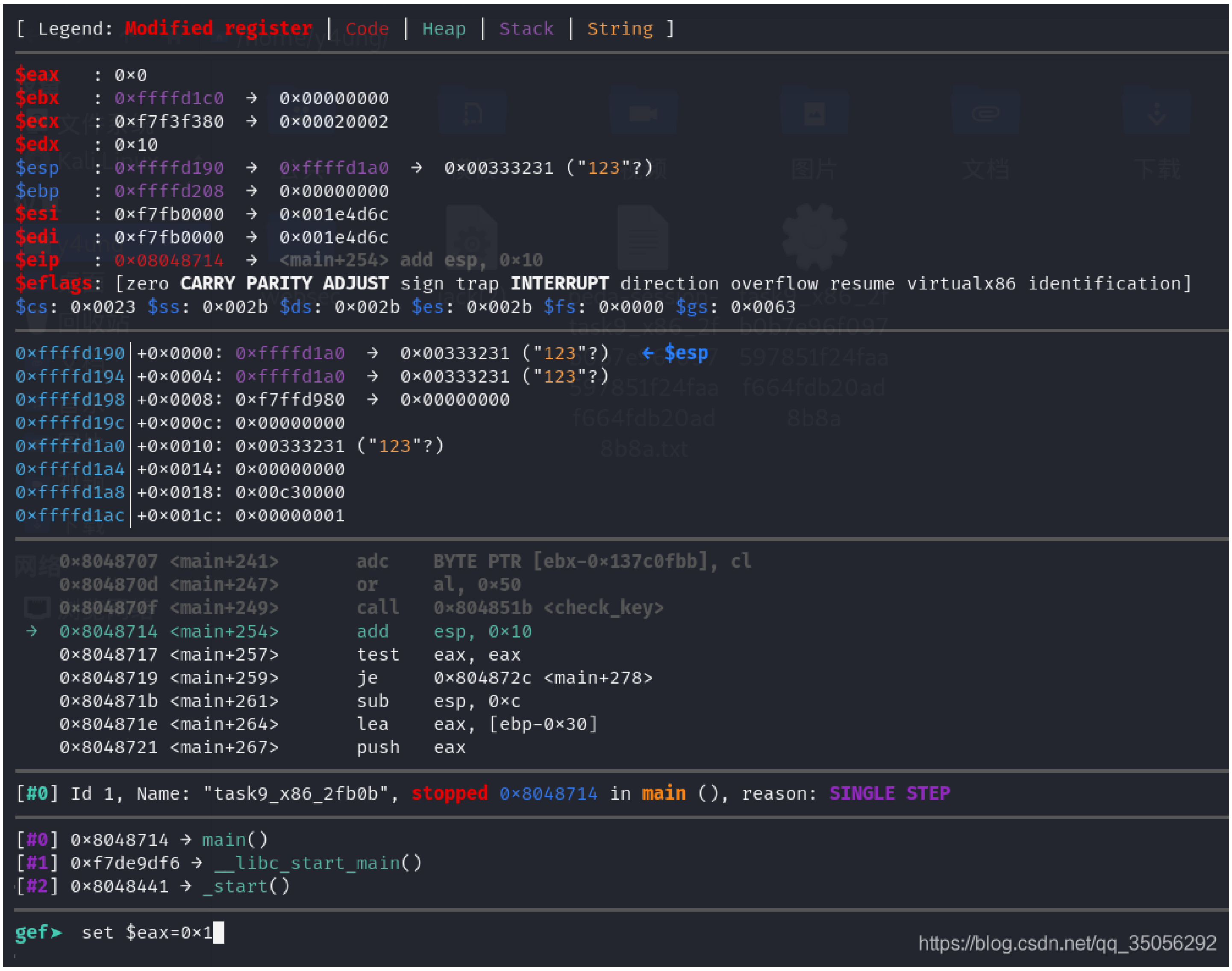
```
C:\Users\hzy\Downloads\f34b6c2d90cd408a919438b7da64c406\task9_x86_ed82b6faaf979658e040c77422d01b1b3db183f7.exe
Key: 123
潇□g??礰霎□n:□=濤□h苷t
```

试试Linux下的文件看看。

```
$ file task9_x86_2fb0b7e96f097597851f24faaf664fdb20ad8b8a
task9_x86_2fb0b7e96f097597851f24faaf664fdb20ad8b8a: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dy
namically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=4c398ee319674018cb4c10b048
42bbb7c46fd9de, not stripped
$ chmod +x task9_x86_2fb0b7e96f097597851f24faaf664fdb20ad8b8a
```

然后用gdb调试，思路仍然是刚刚那个修改check_key()返回值的思路。

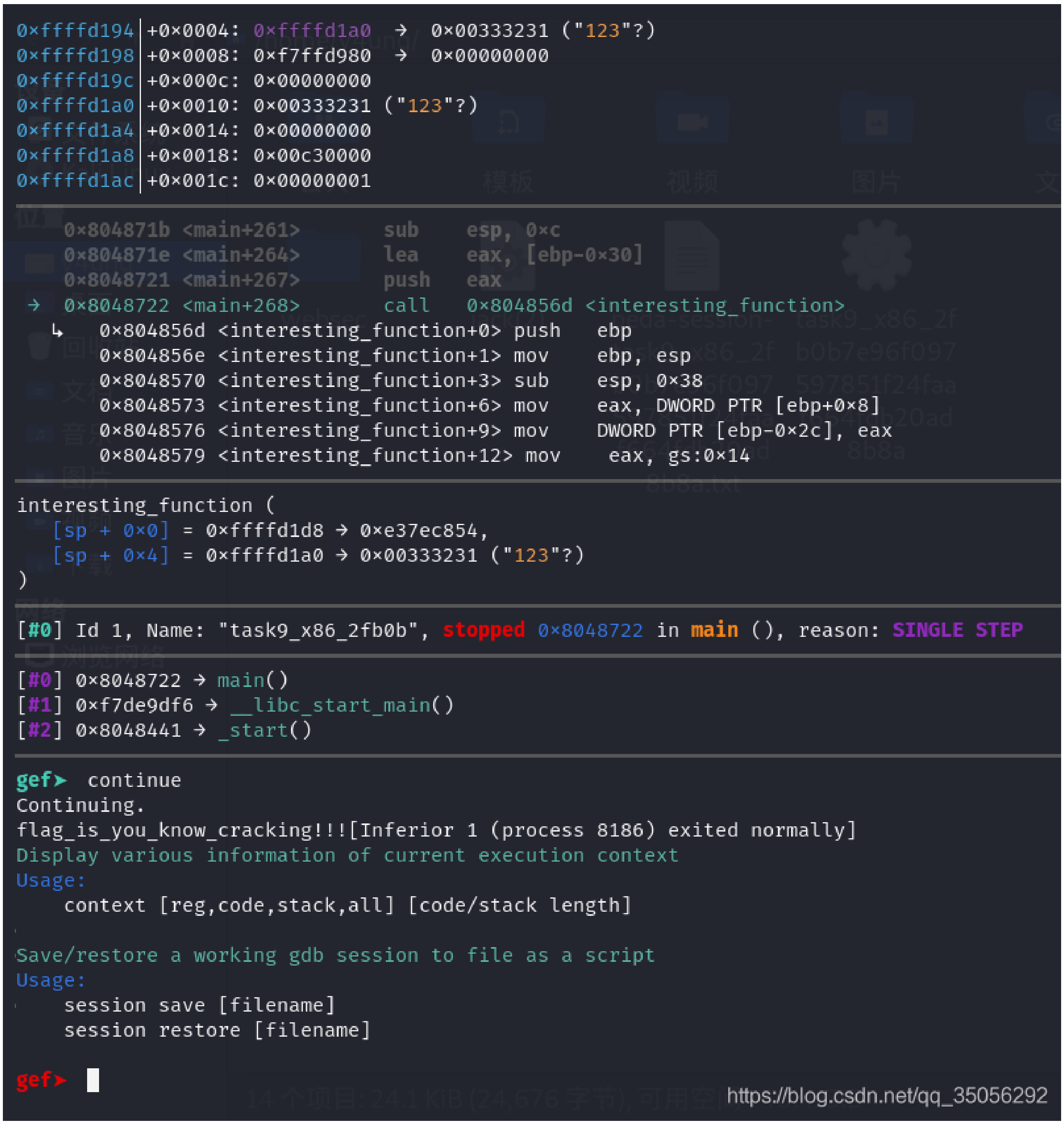
运行完check_key函数，修改返回值为1



```
[ Legend: Modified register | Code | Heap | Stack | String ]
$eax   : 0x0
$ebx   : 0xffffd1c0  →  0x00000000
$ecx   : 0xf7f3f380  →  0x00020002
$edx   : 0x10
$esp   : 0xffffd190  →  0xffffd1a0  →  0x00333231 ("123"?)
$ebp   : 0xffffd208  →  0x00000000
$esi   : 0xf7fb0000  →  0x001e4d6c
$edi   : 0xf7fb0000  →  0x001e4d6c
$eip   : 0x08048714  →  <main+254> add esp, 0x10
$eflags: [zero CARRY PARITY ADJUST sign trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0x0023 $ss: 0x002b $ds: 0x002b $es: 0x002b $fs: 0x0000 $gs: 0x0063
0xffffd190│+0x0000: 0xffffd1a0  →  0x00333231 ("123"?)   ← $esp
0xffffd194│+0x0004: 0xffffd1a0  →  0x00333231 ("123"?)
0xffffd198│+0x0008: 0xf7ffd980  →  0x00000000
0xffffd19c│+0x000c: 0x00000000
0xffffd1a0│+0x0010: 0x00333231 ("123"?)
0xffffd1a4│+0x0014: 0x00000000
0xffffd1a8│+0x0018: 0x00c30000
0xffffd1ac│+0x001c: 0x00000001
    0x8048707 <main+241>       adc    BYTE PTR [ebx-0x137c0fbb], cl
    0x804870d <main+247>       or     al, 0x50
    0x804870f <main+249>       call   0x804851b <check_key>
 →  0x8048714 <main+254>       add    esp, 0x10
    0x8048717 <main+257>       test   eax, eax
    0x8048719 <main+259>       je     0x804872c <main+278>
    0x804871b <main+261>       sub    esp, 0xc
    0x804871e <main+264>       lea    eax, [ebp-0x30]
    0x8048721 <main+267>       push   eax
[#0] Id 1, Name: "task9_x86_2fb0b", stopped 0x8048714 in main (), reason: SINGLE STEP
[#0] 0x8048714 → main()
[#1] 0xf7de9df6 → __libc_start_main()
[#2] 0x8048441 → _start()
gef➤  set $eax=0x1
```

最后直接continue，运行到程序结束即可。

可以看到，flag应该就是：`flag_is_you_know_cracking!!!`



```
0xffffd194│+0x0004: 0xffffd1a0  →  0x00333231 ("123"?)
0xffffd198│+0x0008: 0xf7ffd980  →  0x00000000
0xffffd19c│+0x000c: 0x00000000
0xffffd1a0│+0x0010: 0x00333231 ("123"?)
0xffffd1a4│+0x0014: 0x00000000
0xffffd1a8│+0x0018: 0x00c30000
0xffffd1ac│+0x001c: 0x00000001
───────────────────────────────────────────────────────────────────────
    0x804871b <main+261>       sub    esp, 0xc
    0x804871e <main+264>       lea    eax, [ebp-0x30]
    0x8048721 <main+267>       push   eax
 →  0x8048722 <main+268>       call   0x804856d <interesting_function>
   ↳   0x804856d <interesting_function+0> push   ebp
       0x804856e <interesting_function+1> mov    ebp, esp
       0x8048570 <interesting_function+3> sub    esp, 0x38
       0x8048573 <interesting_function+6> mov    eax, DWORD PTR [ebp+0x8]
       0x8048576 <interesting_function+9> mov    DWORD PTR [ebp-0x2c], eax
       0x8048579 <interesting_function+12> mov    eax, gs:0x14
───────────────────────────────────────────────────────────────────────
interesting_function (
   [sp + 0x0] = 0xffffd1d8 → 0xe37ec854,
   [sp + 0x4] = 0xffffd1a0 → 0x00333231 ("123"?)
)
───────────────────────────────────────────────────────────────────────
[#0] Id 1, Name: "task9_x86_2fb0b", stopped 0x8048722 in main (), reason: SINGLE STEP
───────────────────────────────────────────────────────────────────────
[#0] 0x8048722 → main()
[#1] 0xf7de9df6 → __libc_start_main()
[#2] 0x8048441 → _start()
───────────────────────────────────────────────────────────────────────
gef➤  continue
Continuing.
flag_is_you_know_cracking!!![Inferior 1 (process 8186) exited normally]
Display various information of current execution context
Usage:
    context [reg,code,stack,all] [code/stack length]

Save/restore a working gdb session to file as a script
Usage:
    session save [filename]
    session restore [filename]

gef➤  █
```

3. 总结

1. 对于直接运行就能出flag的题，可以用动态调试直接过掉。通过修改寄存器的值，让程序往我们期望的方向运行。
2. 当题目里提供了windows、Linux下的可执行文件时，如果感觉自己思路没错，windows的结果是乱码，那么不妨试一试Linux的文件。