Use After Free

What is UAF？

悬垂指针：指向曾经存在的对象，但该对象已经不再存在了，
此类指针称为垂悬指针。结果未定义，往往导致程序错误，而且难以检测。[1]

UAF 漏洞：当悬挂指针在释放后没有为其分配新的内存块时使用悬挂指针时，
这就被称为“use after free”漏洞[2]。由于指针悬摆不定，所以当指针用于写入
存储器时，则一些其他的数据结构可能被破坏。即使只读一次内存，也可能会导致
信息泄露（如果下一个位置申请的位置存放了敏感信息），或者进行权限提升（利用
已被释放的内存块进行shellcode执行）

UAF成因：dlmalloc是C标准库提供的分配器, 也是应用程序默认使用的malloc/free等函数。
该内存分配器进行程序的内存管理。当使用free释放内存块的时候，大小小于256kb的块
将被置于空闲状态。原因一是，不一定能立刻被释放（如当前内存块可能不在堆顶）
二是供应用程序下次申请用。这就导致当出现悬垂指针的时候，只要申请大小与上一次
相同的块，即可利用到先前被释放的内存块。

“hello world”


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(){
    char *p1;
    char *p2;
    p1 = (char *)malloc(sizeof(char)*10);
    memcpy(p1, "hello", 10);
    printf("p1 addr:%x ,%s\n", p1, p1);
    free(p1);
    p2 = (char *)malloc(sizeof(char)*10);
    memcpy(p2, "world", 10);
    printf("p2 addr:%x ,%s\n", p2, p1);
    return 0;
}
```

一个关于uaf的简易例子

该程序分别申请了两个字符指针，并分别被赋值“hello”和“world”。p1被释放之后，p2申请大小与p1相同大小的内存块。这时，p2申请的内存块为被释放的p1内存块。由于p1指针未被置为NULL，导致p1与p2指向同一个内存块。所以当程序输出内存信息的时候，出现p1的内容为p2的内存里的内容。

基于UAF的简易密码破解

```
int main () {
    int * p_index ,* p_pass ;
    int mode = 0;
    scanf("%d", &mode);
    if(mode != 1 ) {
        printf ("test stop\n");
        return 0;
    }
    p_global =( int *) malloc ( sizeof ( int ));
    * p_global = SECRET_PASS ;
    if( mode == 1 ) {
        p_index =( int *) malloc ( sizeof ( int ));
        printf (" Give a number between 0 and 1\n");
        scanf ("%d",p_index );
        index_user ( p_index );
        free ( p_index );
    }else {
        printf (" Good luck ! \n");
    }
        p_pass =( int *) malloc ( sizeof ( int ));
        printf (" Give the secret \n");
        scanf ("%d",p_pass );
    if (* p_pass ==* p_global ) {
        printf (" Congrats ! \n");
    }else {
        printf (" Sorry ...\n");
    }
    getchar();
    return 0;

}
```

为全局变量p_global申请内存空间，并赋值为密钥

这里申请的p_index则是使用人为制造的悬垂指针，然后利用index_user函数对p_global进行操作

该指针指向用户输入密码

对比两指针内容，进行密码验证

```c
#include <stdlib.h>
#include <stdio.h>
#define SECRET_PASS 123

int * p_global ;

void index_user ( int *p) {
    int * p_global_save ;
    p_global_save = p_global ;
    p_global =p;
    if( * p_global == 1) {
        printf (" UAF mode on\n");
        return ;
    }else {
        printf (" UAF mode off \n");
        p_global = p_global_save ;
        return ;
    }
}
```

原始密码为 ”123”

人为制造一个uaf的漏洞，设置一个全局变量p_global，利用该指针进行密钥匹配。

使用p_global_save的临时变量存储p_global的内容。把传入参数p_index的内容赋给p_global

当uaf模式关闭时，p_global的内容被还原

IDA 静态分析

1.内存模型和值分析

1.1 抽象内存表示

假设在栈中的地址是基于基址寄存器EBP，那么每一个栈桢将被描述为(EBP$_0$,offset(偏移量))，EBP$_0$ 为初始化的EBP，例如局部变量p_index表示为(EBP$_0$,-4)，或全局变量p_global表示为(p_global)。相对于堆，定义HE(base, size)为申请的堆块，其也可表示为chunk。我们定义两个函数，HA和HF，分别用来表示所有申请和释放的堆块[3]

1.2 对应值分析

此分析的目的在于静态分析程序申请与释放堆块，同时追踪内存地址和其大小。因此根据抽象内存表示所产生的抽象环境，我们可以表示每个内存块的地址和对应可能值

1.3字符化uaf

定义公式为：UafSet = { (pc, chunk) | chunk ∈ AbsEnv(ad) ∩ HE

ad表示 malloc或者free call中的参数和块大小

例如, AbsEnv(38) ∩ HE = {chunk0,chunk1} 即 UafSet = { (38, chunk1) }

意思是当前chunk1是悬摆的且在第38行

2.流程图分析

根据上面的VSA分析结果，对应流程图中的语句进行分析

VSA分析表

Code	AbsEnv	Heap
31 p_global =(int *) malloc (..)	{(**p global**,(**chunk$_0$**))}	HA = {**chunk$_0$**} HF = ∅
34 p_index =(int *) malloc (..)	{((**EBP$_0$, -4**),(**chunk$_1$**))}	HA ={chunk$_0$, **chunk$_1$**} HF = ∅
13 p_global_save = p_global	{(p global,(chunk$_0$)), ((**EBP$_0$, -44**),(**chunk$_0$**))}	HA ={chunk$_0$, chunk$_1$} HF = ∅
14 p_global = p	{(**p global**,(**chunk$_1$**)), ((EBP$_0$ - 4),(chunk$_1$))}	HA ={chunk$_0$, chunk$_1$} HF = ∅
19 p_global = p_global_save	((EBP$_0$, -44),(chunk$_0$))}	HA ={chunk$_0$, chunk$_1$} HF = ∅
37 index_user (p_index)	{(**p global**,(**chunk$_0$, chunk$_1$**)), ((EBP$_0$, -4),(chunk$_1$)), ((EBP$_0$, -44),(chunk$_0$))}	HA ={chunk$_0$, chunk$_1$} HF = ∅
38 free (p_index)	{((EBP$_0$, -4),(chunk$_1$))}	HA = {chunk$_0$} **HF = {chunk$_1$}**
42 p_pass =(int *) malloc (..)	((EBP$_0$ - 8), (chunk$_2$))}	HA = {chunk$_0$, **chunk$_2$**} HF = {chunk$_1$}
45 if (* p_pass ==* p_global)	{(p global,(chunk0, chunk$_1$)), ((EBP$_0$ - 8), (chunk$_2$))}	HA = {chunk$_0$, chunk$_2$} HF = {chunk$_1$}

该部分为index_user 的两个分支，p_global的两个可能值

由于AbsEnv和heap中出现了同一堆块chunk1，该chunk悬挂

分配p_global的内存空间，
并赋值为密钥

```
push    4                   ; size_t
call    _malloc
add     esp, 4
mov     dword_42AC5C, eax
mov     ecx, dword_42AC5C
mov     dword ptr [ecx], 7Bh
mov     edx, dword_42AC5C
mov     eax, [edx]
push    eax
push    offset aP_globalPass_w; "p_global = pass_word = %d\n"
call    _printf
add     esp, 8
cmp     [ebp+var_C], 1
jnz     short loc_40FCA6
```

```
push
call
add
xor
jmp
```

false

true

```
push    4                   ; size_t
call    _malloc
add     esp, 4
mov     [ebp+var_4], eax
push    offset aGiveANumberBet; "Give a number between 0 and 1\n"
call    _printf
add     esp, 4
mov     ecx, [ebp+var_4]
push    ecx
push    offset aD           ; "%d"
call    _scanf
add     esp, 8
mov     edx, [ebp+var_4]
push    edx
call    sub_401005
add     esp, 4
mov     eax, [ebp+var_4]
push    eax
call    sub_402160
add     esp, 4
jmp     short loc_40FCB3
```

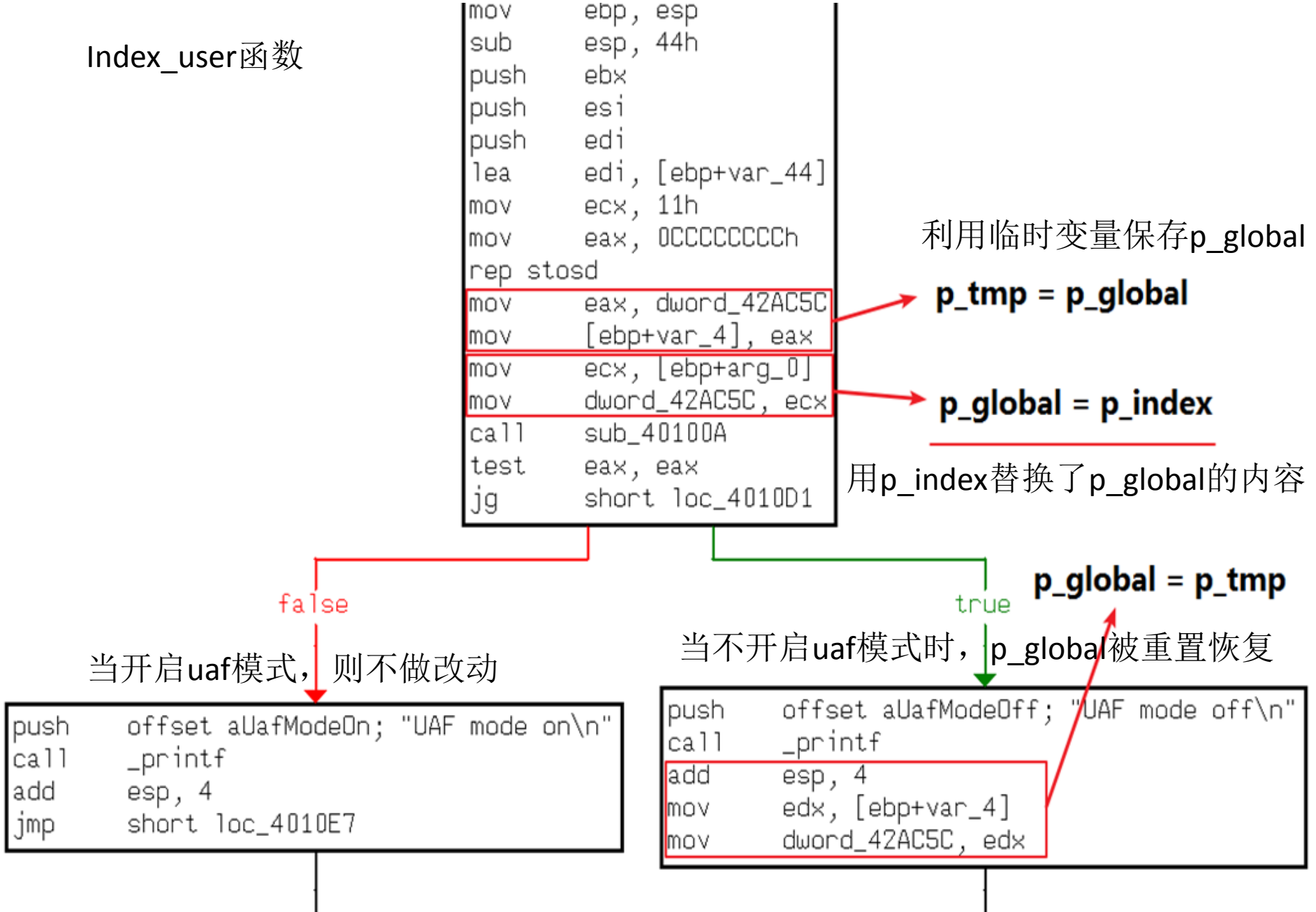
分配p_index的内存空间

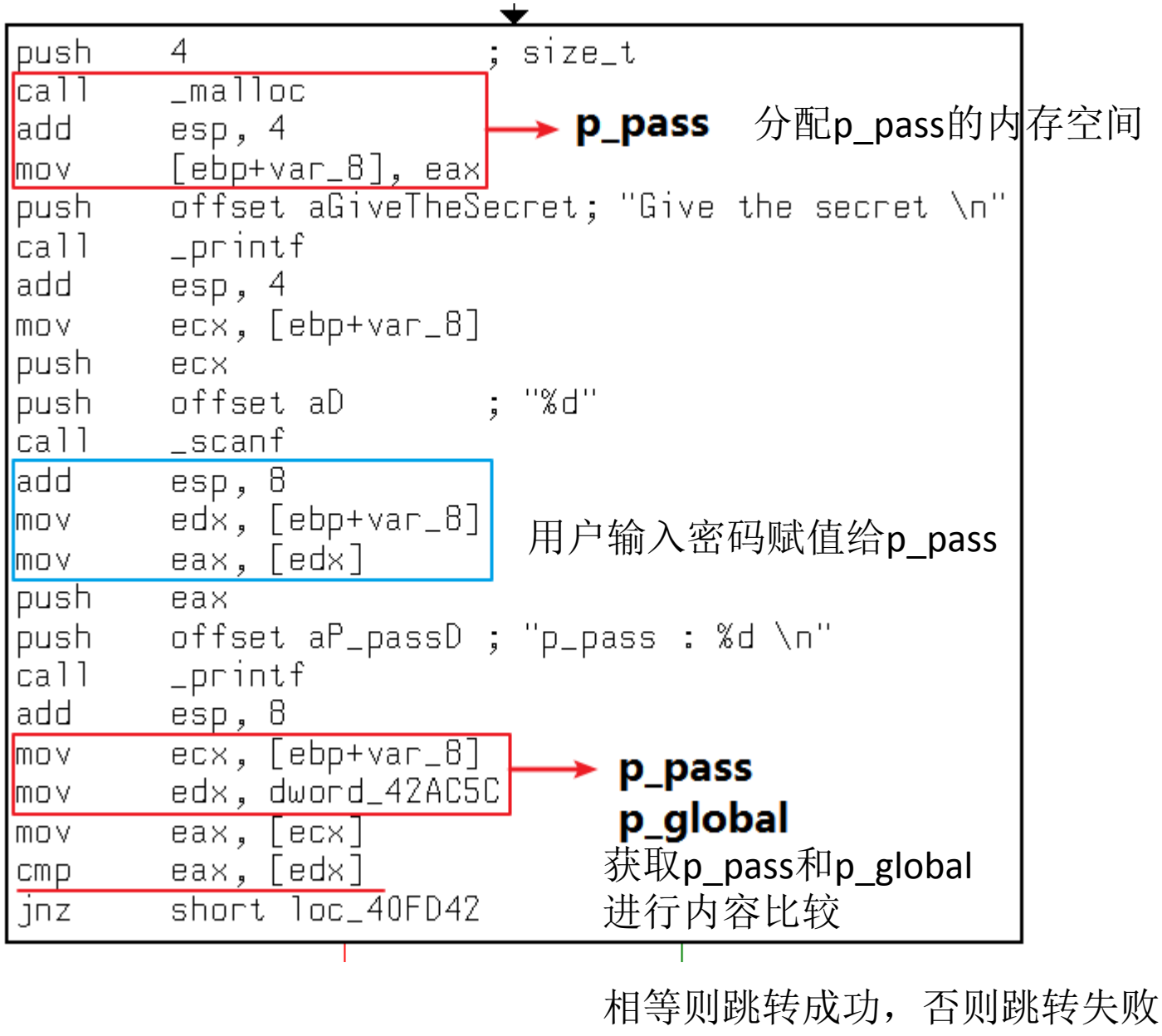
调用index_user函数，打开
Uaf模式，制造漏洞

free
释放p_index的内存空间

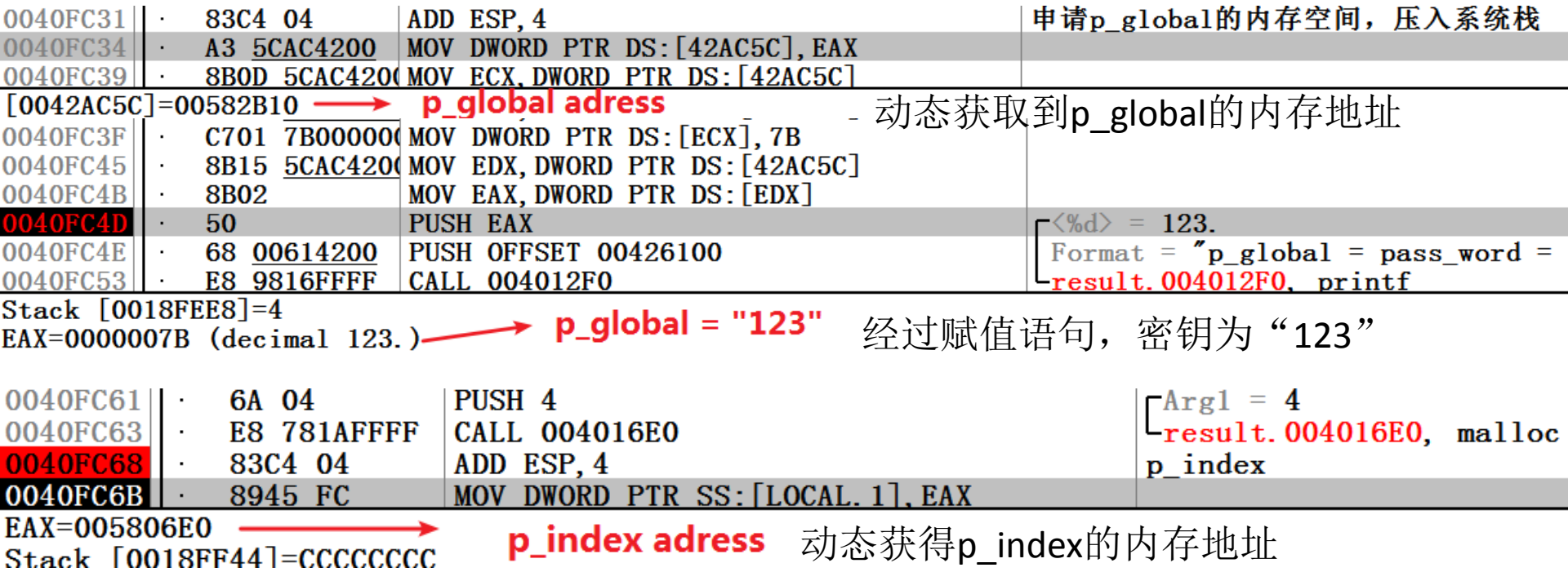
```
push    offset aGoodLuck; "Good luck ! \n"
call    _printf
add     esp, 4
```

Index_user函数




push 4 ; size_t
call _malloc
add esp, 4
mov [ebp+var_8], eax
push offset aGiveTheSecret; "Give the secret \n"
call _printf
add esp, 4
mov ecx, [ebp+var_8]
push ecx
push offset aD ; "%d"
call _scanf
add esp, 8
mov edx, [ebp+var_8]
mov eax, [edx]
push eax
push offset aP_passD ; "p_pass : %d \n"
call _printf
add esp, 8
mov ecx, [ebp+var_8]
mov edx, dword_42AC5C
mov eax, [ecx]
cmp eax, [edx]
jnz short loc_40FD42
p_pass
分配p_pass的内存空间
用户输入密码赋值给p_pass
p_pass
p_global
获取p_pass和p_global
进行内容比较
相等则跳转成功，否则跳转失败

ollydbg 动态分析程序



```
004010A8  ·  A1 5CAC4200   MOV EAX,DWORD PTR DS:[42AC5C]
004010AD  ·  8945 FC       MOV DWORD PTR SS:[LOCAL.1],EAX
004010B0  ·  8B4D 08       MOV ECX,DWORD PTR SS:[ARG.1]
004010B3  ·  890D 5CAC420( MOV DWORD PTR DS:[42AC5C],ECX
004010B9  ·  E8 4CFFFFFF   CALL 0040100A
[0042AC5C]=00582B10
```

p_global adress 当前p_global的内存地址

```
004010A8  ·  A1 5CAC4200   MOV EAX,DWORD PTR DS:[42AC5C]
004010AD  ·  8945 FC       MOV DWORD PTR SS:[LOCAL.1],EAX
004010B0  ·  8B4D 08       MOV ECX,DWORD PTR SS:[ARG.1]
004010B3  ·  890D 5CAC420( MOV DWORD PTR DS:[42AC5C],ECX
004010B9  ·  E8 4CFFFFFF   CALL 0040100A
ECX=005806E0
[0042AC5C]=00582B10
```

p_global = p_index 在index_user函数中做的内容交换

```
0040FCB5  ·  E8 261AFFFF   CALL 004016E0          result.004016E0, malloc
0040FCBA  ·  83C4 04       ADD ESP,4               p_pass
0040FCBD  ·  8945 F8       MOV DWORD PTR SS:[LOCAL.2],EAX
EAX=005806E0
Stack [0018FF40]=CCCCCCCC
```

p_pass adress 动态获取p_pass 的内存地址

通过对比发现p_global和p_pass指向同一内存地址

```
0040FCD1  ·  68 2C504200   PUSH OFFSET 0042502C    Arg1 = ASCII "%d"
0040FCD6  ·  E8 8539FFFF   CALL 00403660           result.00403660, scanf
0040FCDB  ·  83C4 08       ADD ESP,8
0040FCDE  ·  8B55 F8       MOV EDX,DWORD PTR SS:[LOCAL.2]
0040FCE1  ·  8B02          MOV EAX,DWORD PTR DS:[EDX]
[005806E0]=0000014D (decimal 333.)
EAX=1
```

p_pass = "333" 用户输入的密码为“333”

```
0040FCF1  ·  8B4D F8        MOV ECX,DWORD PTR SS:[LOCAL.2]
0040FCF4  ·  8B15 5CAC420(  MOV EDX,DWORD PTR DS:[42AC5C]
0040FCFA  ·  8B01           MOV EAX,DWORD PTR DS:[ECX]
0040FCFC  ·  3B02           CMP EAX,DWORD PTR DS:[EDX]
0040FCFE  ·  75 42          JNE SHORT 0040FD42
0040FD00  ·  68 A0504200    PUSH OFFSET 004250A0
[0042AC5C]=005806E0
```

p_global adress

动态获取到p_global的内存地址

```
0040FCFC  ·  3B02           CMP EAX,DWORD PTR DS:[EDX]
0040FCFE  ·  75 42          JNE SHORT 0040FD42
0040FD00  ·  68 A0504200    PUSH OFFSET 004250A0        Format = "Congrats ! "
0040FD05  ·  E8 E615FFFF    CALL 004012F0               result.004012F0
0040FD0A  ·  83C4 04        ADD ESP,4
[005806E0]=0000014D (decimal 333.)
EAX=0000014D (decimal 333.)
```

compare value

进行内容匹配

```
0040FCFE  ·  75 42          JNE SHORT 0040FD42
0040FD00  ·  68 A0504200    PUSH OFFSET 004250A0        Format = "Congrats ! "
0040FD05  ·  E8 E615FFFF    CALL 004012F0               result.004012F0
0040FD0A  ·  83C4 04        ADD ESP,4
Jump is not taken
Dest=result.0040FD42
```

开启uaf模式的结果

```
1
p_global = pass_word = 123
Give a number between 0 and 1
1
UAF mode on
Give the secret
333
p_pass : 333
Congrats !
p_global :5806e0, 333
p_pass :5806e0, 333
```

未开启uaf模式的结果

```
1
p_global = pass_word = 123
Give a number between 0 and 1
0
UAF mode off
Give the secret
345
p_pass : 345
Sorry ...
p_global :2c2b10, 123
p_pass :2c06e0, 345
```

Reference

[1]百度百科，悬垂指针的定义
[2]维基百科，Dangling pointer ， section introduction
[3]Laurent Mounier, Marie-Laure Potet, Josselin Feist,
Statically detecting use after free on binary code, GreHack 2013, Grenoble, France