**Homework Assignment #3**

**1. Source Code**

#include <string.h>

#include <stdio.h>

void normal\_function(const char\* input)

{

char buf[10];

printf("Stack before input :\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n\n");

// strcpy 함수를 통해 input값을 buf에 복사

// strcpy 함수는 NTBS 보장을 안 하기 때문에 모두 복사

strcpy(buf, input);

printf("%s\n", buf);

printf("Stack after input :\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n\n");

}

// 해킹 성공을 Test 해볼 함수

// 해당 함수의 주소 값(0x8048498)으로 ret값을 덮어씌움

void hacked\_function(void)

{

printf("I am hacked!\n");

}

int main(int argc, char\* argv[])

{

// normal\_function과 hacked\_function의 시작 주소 출력

printf("The address of normal\_function = %p\n", normal\_function);

printf("The address of hacked\_function = %p\n", hacked\_function);

// 명령행 인자가 2개가 아니라면 비정상 종료

if (argc != 2)

{

printf("Input one argument\n");

return -1;

}

// 2번째 명령행 인자를 normal\_function 매개변수로 넣음

normal\_function(argv[1]);

return 0;

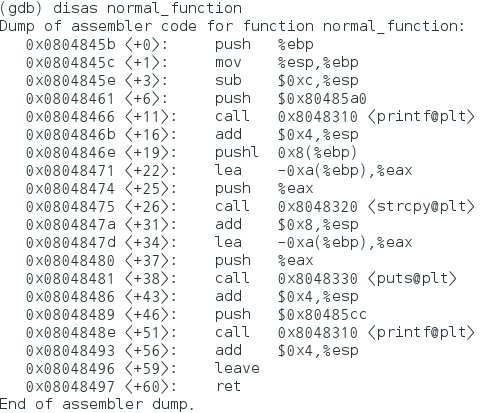
}

**2. 해킹에 성공한 문자열과 그 문자열이 만들어진 이유 자세히 설명**

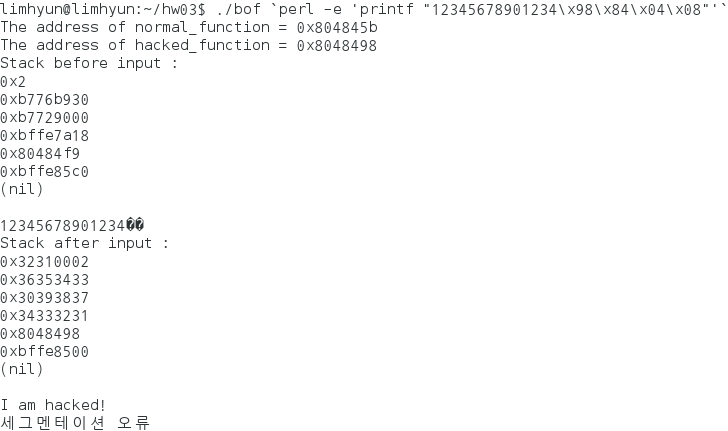
**./bof `perl -e 'printf "12345678901234\x98\x84\x04\x08”’`**

우선 **\x98\x84\x04\x08의 의미**를 먼저 설명하자면, **hacked\_function의 시작 주소를 의미**합니다. 사실 hacked\_function의 시작 주소는 0x8048498이지만 위처럼 써준 이유는 바로 ‘Endian’ 때문 입니다. 엔디언에는 크게 빅 엔디언과 리틀 엔디언으로 나눌 수 있는데, 현재 실습에 쓰이는 컴퓨터 구조(x86 아키텍처)는 리틀 엔디언(인텔 포맷)이기 때문에 작은 단위의 바이트가 앞에 오게 되었습니다.

그리고 앞에 있는 **12345678901234의 의미**는 문장에 의미가 있는 것이 아니라 총 **14글자인데 의미**가 있습니다. 14글자로 적어준 이유는 아래에 <strcpy@plt> 즉, strcpy함수가 호출 되기 전 strcpy에는 매개변수가 buf, input 총 2개가 들어가게 됩니다. 그 중 input에 해당하는 값은 %ebp-0xa에 해당되므로 0xa(hex) -> 10(integer)에 해당하는 byte만큼을 채우고 ebp의 값인 4byte만큼을 더 채우면 총 14byte가 되므로 14글자를 적어주었습니다.



**3. 해킹에 성공한 화면 내용 출력**



Stack after input 내용을 해석하자면 입력했던 12345678901234\x98\x84\x04\x08가 little endian으로 인해서 거꾸로 들어가 있는 모습을 볼 수 있습니다.

**4. (보너스) printf(“Stack before Input : \n%p\n%p\n%p\n%p\n%p\n%p\n\n”); 문장이 스택의 내용을 출력한 이유**

**printf 서식 문자열에 대한 인수가 충분하게 전달되지 않았기 때문**입니다. 그리고 인수가 주어지지 않은 상태에서 아무런 값이 들어있지 않은 push한 $0x80485a0과 $0x8048310의 주소에 있는 값이 출력 되므로 스택의 내용이 출력됩니다.

