**Homework Assignment #4**

**Question 1)**

**① 어떠한 취약점인지 적고 이유를 설명하세요**

1. argv[1]에 값을 잘 못 넣은 경우
   1. unsigned short의 범위를 살짝 넘은 숫자를 입력한 경우 (65536 ~ 65615)

* 세그멘테이션(Overflow) 오류가 발생
  1. 숫자 (INT)가 아닌 값을 입력한 경우
* atoi함수가 적용되지 않아서 s가 0이 됨
  1. 음수 값을 입력한 경우
* 세그멘테이션(Underflow) 오류가 발생

1. argv[1]에서 지정한 사이즈보다 argv[2]의 사이즈가 더 큰 경우

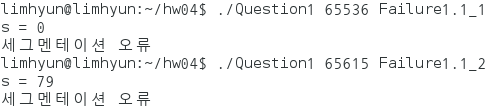
* argv[1] 크기 만큼 복사하기 때문에 argv[2]가 완전히 복사되지 않음

**② 정상적으로 동작할 경우와 취약점으로 인해 비정상적으로 동작할 경우를 각각 실행하고 실행화면을 제시하세요. 비정상적인 경우만 있을 때에는 그 경우만 제시하세요.**

1. 정상적으로 동작할 경우



1. argv[1]에 값을 잘 못 넣은 경우
   1. unsigned short의 범위를 살짝 넘은 숫자를 입력한 경우 (65536 ~ 65615)



* 1. 숫자 (INT)가 아닌 값을 입력한 경우



* 1. 음수 값을 입력한 경우



1. argv[1]에서 지정한 사이즈보다 argv[2]의 사이즈가 더 큰 경우



**③ 발견된 취약점을 완화할 수 있는 전략을 설명하고 코드를 고치세요**

#include <stdio.h>

#include <string.h>

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

unsigned short s;

int i;

char buf[80];

if (argc < 3) {

return -1;

}

i = atoi(argv[1]);

/\* ======== 1.2 취약점 완화 ======== \*/

if (i == 0) { // 0 혹은 string이 들어오면 예외 처리

printf("Error(1.2) : argv[1] Input Positive Integer\n");

return -1;

}

/\* ======== 1.3 취약점 완화 ======== \*/

if (i < 0) { // minus 값이 들어오면 예외 처리

printf("Error(1.3) : argv[1] Input Positive Integer\n");

return -1;

}

/\* ======== 1.1 취약점 완화 ======== \*/

if (i >= 80) { // short에 넣기 전에 예외 처리

printf("Error(1.1) : Too big memory allocation\n");

return -1;

}

s = i;

printf("s = %d\n", s);

/\* ======== 2 취약점 완화 ======== \*/

if (i < strlen(argv[2])) { // 복사가 완전히 되지 않았다는 경고

printf("Waring(2) : Not Completely Copied\n");

}

memcpy(buf, argv[2], i);

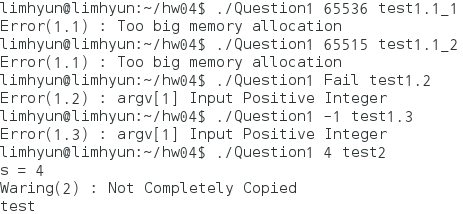
buf[i] = '\0';

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

return 0;

}

**④ ②에서 비정상적으로 동작한 경우에 대해 실행하고 정상적으로 동작하는 모습을 캡처화면으로 제시하세요.**



**Question 2)**

**① 어떠한 취약점인지 적고 이유를 설명하세요**

~ 연산 시 int Promotion 되어 연산이 되어서 ff가 000000ff가 되어 발생하는 취약점

**② 정상적으로 동작할 경우와 취약점으로 인해 비정상적으로 동작할 경우를 각각 실행하고 실행화면을 제시하세요. 비정상적인 경우만 있을 때에는 그 경우만 제시하세요.**



**③ 발견된 취약점을 완화할 수 있는 전략을 설명하고 코드를 고치세요**

int main(void) {

assert(sizeof(unsigned char) == 1);

unsigned char uc1 = 0xff;

unsigned char uc2 = 0;

/\* ======== ~연산시 char로 강제 형 변환 ======== \*/

if (~((char)uc1) == uc2) {

printf("%08x == %08x\n", ~((char)uc1), uc2);

}

else {

printf("%08x != %08x\n", ~uc1, uc2);

}

return 0;

}

**④ ②에서 비정상적으로 동작한 경우에 대해 실행하고 정상적으로 동작하는 모습을 캡처화면으로 제시하세요.**



**Question 3)**

*\* Debian 8에서는 취약점이 발생하지 않아서 Ubuntu 16.04.4에서 진행하였습니다.*

**① 어떠한 취약점인지 적고 이유를 설명하세요**

1. int 범위 이상으로 입력할 경우

* overflow가 발생하여 음수가 됨

1. unsigned int 범위 이상으로 입력할 경우

* overflow가 발생하여 다시 0부터 시작이 됨

**② 정상적으로 동작할 경우와 취약점으로 인해 비정상적으로 동작할 경우를 각각 실행하고 실행화면을 제시하세요. 비정상적인 경우만 있을 때에는 그 경우만 제시하세요.**

1. 정상적으로 동작할 경우



1. Int 범위 이상으로 입력할 경우



1. unsigned int 범위 이상으로 입력할 경우



**③ 발견된 취약점을 완화할 수 있는 전략을 설명하고 코드를 고치세요**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

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

{

int val, i;

char \*mem;

if (argc < 2)

exit(1);

/\* ======== 10의 자리가 넘을 경우 예외 처리 ======== \*/

if (strlen(argv[1]) >= 10) {

printf("Error : Input less than 1,000,000,000\n");

return -1;

}

val = atoi(argv[1]);

if (val > 0) {

mem = malloc(val \* sizeof(char \*));

if (mem == NULL) {

printf("Failure\n");

exit(2);

}

}

for (i = 0; i < val; i++) {

mem[i] = 'A';

printf("%c", mem[i]);

}

printf("\n");

return 0;

}

**④ ②에서 비정상적으로 동작한 경우에 대해 실행하고 정상적으로 동작하는 모습을 캡처화면으로 제시하세요.**

