



Houston, We Have a Problem

Bite-Size Secure Coding

해설서

Instruction

- 타겟 : C 언어를 어느 정도 경험한 0 ~ 2 서클 카뎃
- 목표 : C 언어를 사용함에 있어서 흔히 만나는 에러 케이스를 학습 (메모리, 에러 핸들 -> 시큐어 코딩)
- 맨다토리에서는 직접 에러 케이스를 만들어서 어떤 경우에 발생하는지 학습한다.
- 비슷한 에러 케이스를 묶고, 학습할 수 있는 작은 사이즈로 나눈 후 문제를 배치하였다.
- 맨다토리에서는 에러를 만드는 경우였다면, 보너스에서는 에러를 해결하는 경우이다.

Exercise 00: Compile Error

- 중복된 헤더의 인클루드, 상호 의존적인 헤더 인클루드 케이스를 만들어서, 인클루드에서 발생 가능한 문제 학습

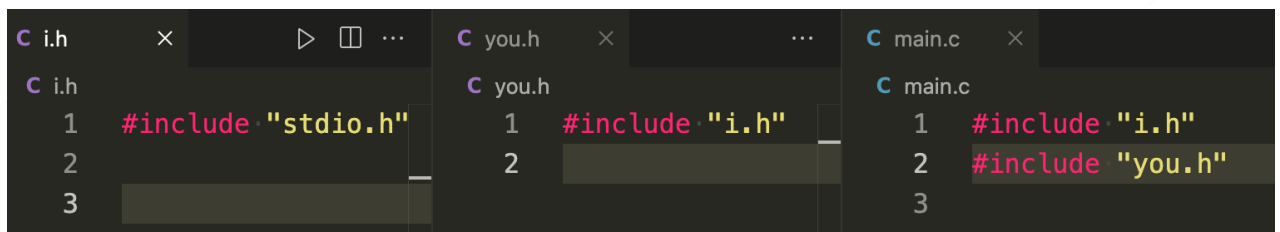


The image shows two code editors side-by-side. The left editor, titled 'i.h', contains the following code:

```
C i.h
1 #include "you.h"
2
```

The right editor, titled 'you.h', contains the following code:

```
C you.h
1 #include "i.h"
2
```



The image shows three code editors side-by-side. The left editor, titled 'i.h', contains the following code:

```
C i.h
1 #include "stdio.h"
2
3
```

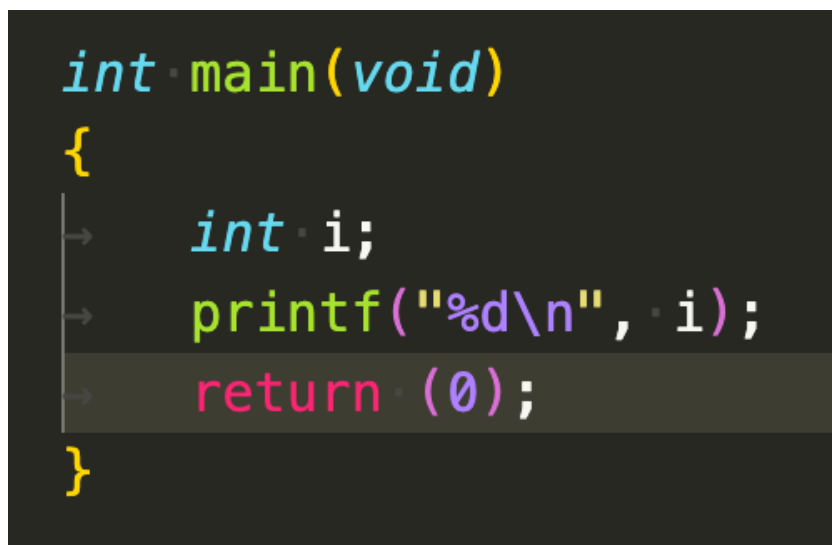
The middle editor, titled 'you.h', contains the following code:

```
C you.h
1 #include "i.h"
2
```

The right editor, titled 'main.c', contains the following code:

```
C main.c
1 #include "i.h"
2 #include "you.h"
3
```

- 초기화 되지 않은 변수를 사용하는 것이 왜 문제가 되는지 학습



```
int main(void)
{
    int i;
    printf("%d\n", i);
    return (0);
}
```

- 함수 프로토타입의 불일치나, 전방 선언 실수로 인한 함수 에러

```
C main.c ×
C main.c > ...
1  #include "stdio.h"
2
3  int main(void)
4  {
5      int i;
6
7      i = 42;
8      ft_print_int(i);
9      return (0);
10 }
11
12 void ft_print_int(int nbr)
13 {
14     printf("%d\n", nbr);
15 }
```

- 캐스팅에 의해서, 변수 값이 의도치 않게 변경되는 문제 학습

```
C main.c ×
C main.c > ...
1  #include "stdio.h"
2
3  int sum(int a, short b)
4  {
5
6      return ((short)a + b);
7  }
8
9  int main(void)
10 {
11     printf("%d\n", sum(65536, 2));
12     return (0);
13 }
```

Exercise 01: Runtime Error

- 무한 루프 반복문의 라이프 사이클 학습

```
C main.c ×
C main.c > ...
1  #include "stdio.h"
2
3  int main(void)
4  {
5      while (1)
6      {
7          printf("Hello World!\n");
8      }
9      return (0);
10 }
```

- 재귀함수의 종료조건 필요성 학습

```
C main.c ×
C main.c > ...
1  #include "stdio.h"
2
3  int main(void)
4  {
5      static int a = 0;
6
7      if (a > 0)
8      {
9          a++;
10         printf("%d\n", a);
11         main();
12     }
13     else
14     {
15         a++;
16         printf("%d\n", a);
17         main();
18     }
19     return (0);
20 }
```

- 스택 메모리 크기의 한계 학습 (무한 재귀 함수, 함수 개수 초과)

```
C main.c ×
C main.c > ...
1  #include "stdio.h"
2
3  int main(void)
4  {
5      static int a = 0;
6
7      if (a > 0)
8      {
9          a++;
10         printf("%d\n", a);
11         main();
12     }
13     else
14     {
15         a++;
16         printf("%d\n", a);
17         main();
18     }
19     return (0);
20 }
```

- 힙 메모리 크기의 한계 학습 (메모리 할당 초과)

```
C main.c ×
C main.c > ...
1  #include <stdlib.h>
2  #include <limits.h>
3
4  int main(void)
5  {
6      char *temp;
7
8      while (1)
9      {
10         temp = malloc(SIZE_MAX);
11     }
12     return (0);
13 }
```

Exercise 02: Exploit

- 메모리 접근 관련 에러에 대한 학습.
- 메모리를 다루는 C언어의 특성상 스택 영역의 메모리 점프도 가능하다는 아래의 예제 코드를 참고하여 각종 메모리 관련 에러에 대한 학습.

```
char *function(char c) {
    char *str;

    str = &c;
    return (str + 59);
}

int main(void) {
    char a = 41;
    char b = 42;
    char *res;

    res = function(a);
    printf("function addr: %lu\n", (unsigned long)&a);
    printf("function addr: %lu\n", (unsigned long)&b);
    printf("function addr: %lu\n", (unsigned long)res);
    printf("function addr diff: %lu\n", (unsigned long)(&a - res));
    printf("function addr diff: %lu\n", (unsigned long)(&b - res));
    printf("function addr val: %d\n", *res);

    return (0);
}
```

- 주어진 메모리 범위를 초과한 접근에서 발생 가능한 문제점 학습 (배열 인덱스 초과, 널 종료)
- 실제로 널 터미네이팅을 하지 않았을때 크래시가 날수도 / 안날수도 있다는 점을 예시로 들어서 왜 이런 현상이 발생하는지 학습.

```
int main(void) {
    char *str;
    char *errstr;

    errno = ENOENT;
    errstr = strerror(errno);
    str = malloc(sizeof(char) * strlen(errstr));
    strcpy(str, errstr);
    printf("%s\n", str);
    printf("dereferencing index 1000000000: %c\n", str[1000000000]);
    return (0);
}
```

- 동적 할당된 메모리 뿐만 아니라 스택 영역에서도 잘못된 주소의 참조가 가능하고, 에러가 발생하지 않을 수 있다는 부분에 대한 학습

```
int *stack_function(void) {
    int a;

    a = 10;
    return &a;
}

int *stack_function2(void) {
    int b = 30;
    int a = 20;
    int c = 40;
    return &a;
}
```

```
int main(void) {
    int *a;

    a = stack_function();
    printf("stack_function called: %d\n", *a);
    stack_function2();
    printf("stack_function2 called: %d\n", *a);
    return (0);
}
```

● seseo@seseoui-MacBookPro ex00 % ./a.out
stack_function called: 10
stack_function2 called: 30

Exercise 03: Dobby is free!!!

- 메모리 할당 해제된 지역을 참조할 때 발생하는 문제 학습, 더블 프리 케이스.

```
void free_strs(char **strs){
    while (*strs){
        free(*strs++);
    }
}
```

```
int main(void) {
    char **strs;
    char **strs2;
    int i;

    strs = malloc(sizeof(char *) * 10);
    strs2 = malloc(sizeof(char *) * 10);
    strs[9] = NULL;
    i = 0;
    while (i < 9){
        strs[i] = malloc(sizeof(char) * 20);
        strs2[i] = strs[i];
        i++;
    }
    free_strs(strs);
    free_strs(strs2);
    return 0;
}
```

- 스택 메모리와, 힙 메모리의 차이점 학습

```
int main(void) {
    {
        int *a;
        int b;

        // a = malloc(sizeof(int) * 2);
        a = &b;
        free(a);
        // free(a);
    }
    return 0;
}
```

Exercise 04: Why doesn't it work

```
int main(int argc, char **argv)
{
    unsigned int    n;

    n = ft_atoi(argv[1]);
    if (argc != 2)
    {
        printf("%s\n", "Give me Number\n");
        return (0);
    }
    printf("%u \n", n);
    return (0);
}
```

- 의도치 않은 부호 비트의 변경에 의한 오류 학습 (MSB)

```
int main(void)
{
    float  change;
    int    i;

    i = 0;
    change = 0.1;
    while (i < 9)
    {
        change += 0.1;
        i++;
    }
    printf("[ %f ] == COMPARE WITH == [ %f ]\n", change, 0.1 * 10);
    if (change == (0.1 * 10))
        printf("SAME\n");
    else
        printf("It's Not SAME!\n");
    return (0);
}
```

- 부동소수 오차 학습 / EPSILON의 필요성 학습
소수점의 실제 저장 방식 학습

```
int main(void)
{
    int temp;
    int count;

    srand(100);
    count = 0;
    while (count < 6)
    {
        temp = rand() % 45;
        printf("%5d", temp);
        count++;
    }
    printf("\n");
    return (0);
}
```

- 편향치없는 난수 생성 학습
rand 함수를 사용하지만 시드값의 중요성에 대한 학습

Exercise 05: Zombie

- 다중 프로세스 환경에서 발생가능한 좀비 프로세스 학습

```
void    *routine(void *s)
{
    while (1)
    {
        write(1, "aa\n", 3);
        sleep(1);
    }
    return (NULL);
}

int main(void)
{
    pthread_t    *thread;

    pthread_create(&thread, NULL, routine, NULL);
    return (0);
}
```

- waitpid()의 사용을 필수로 함으로서, WNOHANG 옵션의 사용방법 학습 및 어떤 경우에 사용할지 고찰
- 메인 함수가 종료 된 후 좀비프로세스가 남아서 프린팅이 계속 되고, pid 를 kill 하는 과정의 학습과 프로세스 누수 확인

Bonus part – Houston, We Have a Problem

- 실제 사용되는 라이브러리에 숨어있는 에러 요소 발견 및 수정
- 코드 리딩 능력 향상
- 문제 해결 능력 향상