


**Kuality** 신입생 교육 1주차

1. 자신의 노트북 혹은 컴퓨터가 **arm** 과 **intel** 인지  
확인해 볼 필요가 있음 특히 **2021** 맥북 부터는 **arm**  
임 실습이 불가함 **2021** 맥북 이상은 리눅스 서버를  
따로 제공 예정

모든 설명은 **64비트** 기준으로 설명

환경 구성

 \*제목 없음 - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

---

Ubuntu Linux : <https://releases.ubuntu.com/18.04/ubuntu-18.04.6-desktop-amd64.iso>

virtualbox : <https://download.virtualbox.org/virtualbox/6.1.32/VirtualBox-6.1.32-149290-Win.exe>

ida : <https://drive.google.com/file/d/19ax4v08iBpIplx2NYWPwT-t48mY0wDsr/view?usp=sharing> [외부 유출 금지]

개발자 즉 프로그래머가 코드를 제공하지 않고  
프로그램(.exe, elf, etc..) 만 제공할때 사용자가  
직접 프로그램을 분석할때 쓰는게 ida

리눅스 환경을 윈도우 안에서 쉽게 구성할 수  
있는데 **virtualbox** 이다. **vmware** 라는 좋은  
프로그램도 존재하지만 유료이다.

리눅스 환경에서 진행하는 이유는 윈도우 시스템  
해킹은 리눅스에 비해 많이 어려운 부분이 많음  
그래서 시스템해킹에 입문하는 대부분  
리눅스환경에서 진행함

C Lang Vuln



배역

```
#include <stdio.h>
```

```
int main() {
```

```
    char buf[4] = "ABCD"; //type name[size]
```

```
    printf("%c\n", buf[0]); //%c = 문자
```

```
    printf("%c\n", buf[1]);
```

```
    printf("%c\n", buf[2]);
```

```
    printf("%c", buf[3]);
```

```
}
```

Apple > ~/Univ WorkStaion/C Lang/**station**

gcc -o test test.c ; ./test

A

B

C

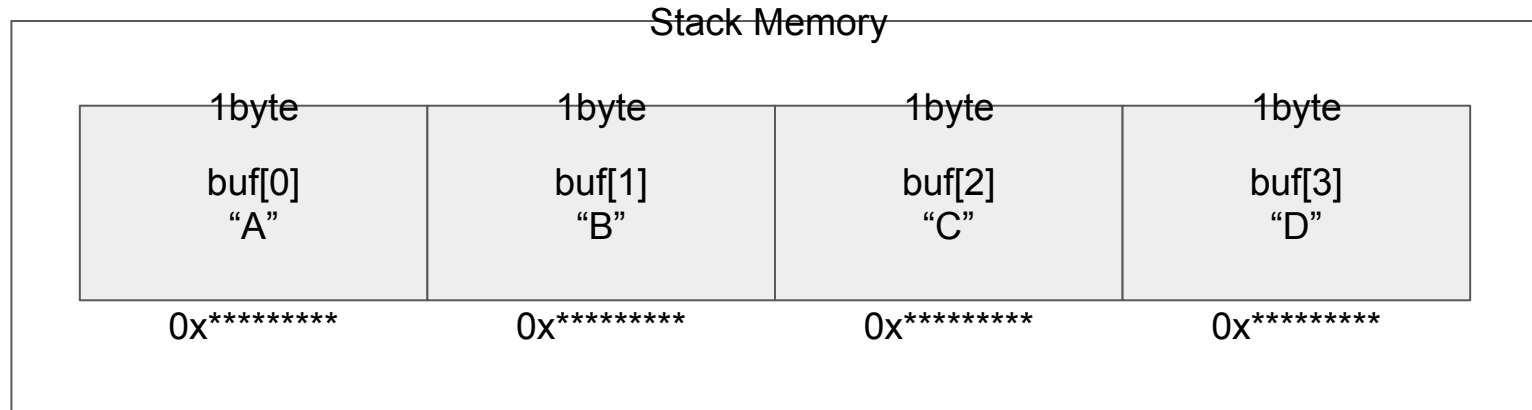
D %

Apple > ~/Univ WorkStaion/C Lang/**station**

char = 1byte

배열은 0번째부터.

```
char buf[4] = "ABCD";
```



메모리 주소는 계속 바뀐다.

```
#include <stdio.h>
```

```
int main() {
```

```
    char buf[4] = "ABCD"; //type name[size]
```

```
    printf("%s\n", buf);    //%s = 문자
```

```
    printf("%p\n", &buf);  //& 쓰면 메모리 주소 출력
```

```
}
```

```
leedohyun@DESKTOP-QMUCB7T: /mnt/c/Users/leedohyun/Desktop/hwang$ ./a
ABCD
0x7fff3c870584
leedohyun@DESKTOP-QMUCB7T: /mnt/c/Users/leedohyun/Desktop/hwang$ ./a
ABCD
0x7ffd09abe9e4
leedohyun@DESKTOP-QMUCB7T: /mnt/c/Users/leedohyun/Desktop/hwang$ ./a
ABCD
0x7ffd2a2d67d4
leedohyun@DESKTOP-QMUCB7T: /mnt/c/Users/leedohyun/Desktop/hwang$ |
```

64bit 메모리 주소



배열을 쓰면 나오는 취약점

uninitialized pointer  
uninitialized value  
buffer-overflow  
out-of-bound  
etc..

4개 취약점 모두 배열에서만  
발생하지 않음 다양한  
측면에서 발생할 수 있다.  
하지만 여기서는 배열을  
사용할 예정

uninitialized pointer

초기화 되지 않은 주소

uninitialized value

초기화 되지 않은 값

-> 이 값을 출력 예정

uninitialized value

예제

```
#include <stdio.h>
```

```
int main() {
```

```
    char buf[4] = "ABCD"; //type name[size]
```

```
    printf("%s\n", buf);    //%s = 문자
```

```
}
```

```
[+] Starting local process './test': pid 58544
[*] Switching to interactive mode
[DEBUG] Received 0xa bytes:
00000000 41 42 43 44 60 37 54 6f 01 0a |ABCD|`7To|..|
0000000a
ABCD`7To
[*] Process './test' stopped with exit code 0 (pid 58544)
[*] Got EOF while reading in interactive
$
[DEBUG] Sent 0x1 bytes:
10 * 0x1
[*] Got EOF while sending in interactive
Traceback (most recent call last):
  File "/opt/homebrew/lib/python3.9/site-packages/pwnlib/tubes/process.py", line 746, in close
    fd.close()
BrokenPipeError: [Errno 32] Broken pipe
🍎 > ~/Univ WorkStaion/C Lang/station > ./exploit.py
[+] Starting local process './test': pid 58547
[*] Switching to interactive mode
[DEBUG] Received 0xa bytes:
00000000 41 42 43 44 60 77 69 6d 01 0a |ABCD|`wim|..|
0000000a
ABCD`wim
[*] Process './test' stopped with exit code 0 (pid 58547)
[*] Got EOF while reading in interactive
$
[DEBUG] Sent 0x1 bytes:
10 * 0x1
[*] Got EOF while sending in interactive
Traceback (most recent call last):
  File "/opt/homebrew/lib/python3.9/site-packages/pwnlib/tubes/process.py", line 746, in close
    fd.close()
BrokenPipeError: [Errno 32] Broken pipe
🍎 > ~/Univ WorkStaion/C Lang/station > 
```



임외의 값이 출력되면서  
위협으로 다가올 수 있음!

# Stack based Buffer-Overflow

```
#include <stdio.h>
#include <string.h>

int main() {
    char buf[4]; //type name[size]
    scanf("%s", &buf);
}
```

# Trigger

🔍 🏠 ~ > ./test

ABCD A

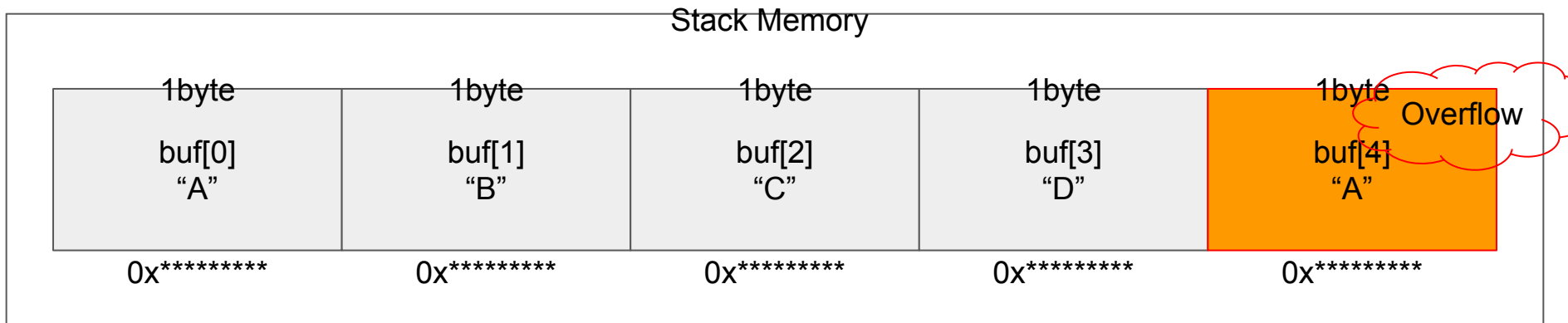
\*\*\* stack smashing detected \*\*\*: <unknown> terminated

[1] 2656 abort (core dumped) ./test

🔍 🏠 ~ > █

1 byte overflow

```
char buf[4] = "ABCD";
```



0x41 = A, 0x42 = B, 0x43 = C, 0x44 = D

```
pwndbg> x/24gx 0x7fffffffffe3c4
```

```
0x7fffffffffe3c4: 0x56e6004144434241
```

```
0x55554730922f9328
```

```
0x7fffffffffe3d4: 0xf7a03c8700005555
```

```
0x0000000100007fff
```

```
0x7fffffffffe3e4: 0xffffe4b800000000
```

```
0x0000800000007fff
```

```
0x7fffffffffe3f4: 0x555546da00000001
```

```
0x0000000000005555
```

```
0x7fffffffffe404: 0x1e0de00b00000000
```

```
0x555545d01d0564b0
```



# Stack based Buffer-Overflow

상황

buf[0] "A"	1 byte
buf[1] "B"	1 byte
buf[2] "C"	1 byte
buf[3] "D"	1 byte
<b>Canary</b>	8 byte
SFP	8 byte
RET	8 byte

Canary 는 상황에 따라 있을 수  
있고 없을 수 있다. (개발자  
마음)

빨간색 : 인풋값  
주황색 : Canary  
노란색 : SFP  
초록색 : RET

```
pwndbg> x/24gx 0x7fffffffffe3d0
```

```
0x7fffffffffe3d0: 0x44434241ffffe4c0
```

```
0x49a2d90447347700
```

```
0x7fffffffffe3e0: 0x0000555555554730
```

```
0x00007ffff7a03c87
```

Stack-based Buffer Overflow 를 활용하여 RET 영역에 있는 값을 바꾸면 프로그램에 실행 흐름을 제어할 수 있다.

# 예제 1

```
#include <stdio.h>
```

```
int main() {
```

```
    char buf[4];
```

```
    char d[8] = "";
```

```
    scanf("%s", &buf);
```

```
    if(strcmp(d, "BBBBBBBB") == 0) {
```

```
        printf("hack!");
```

```
    }
```

```
    else {
```

```
        printf("no hack!");
```

```
    }
```

```
}
```

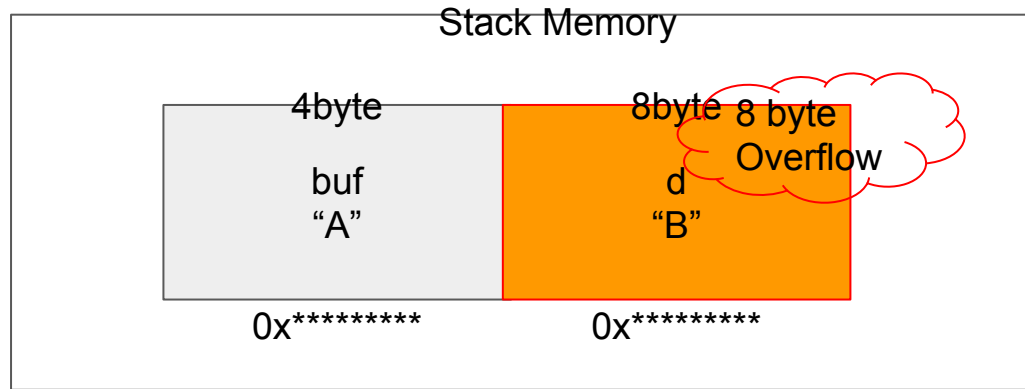
strcmp 함수



문자열 비교 함수로 문자열이  
서로 같으면 0을 반환함

즉 이 코드는 d배열이 변조  
되었나 안되었나 확인해주는  
코드다 변조가 되면 “hack”  
이라고 출력되고 다르면 “no  
hack”이라고 출력됨

```
./test  
AAAABBBBBBBBBB  
hack!%  
./test  
AAAAAAAAAAAAAA  
no hack!%  
.
```



OOB (Out-Of-Bounds)

```
#include <stdio.h>

int main()
{
    char array[6] = "ABCDE";
    int idx;

    printf("Input array index : ");
    scanf("%d",&idx);

    puts(&array[idx]);

    return 0;
}
```

```
~/junha > ./oob
Input array index : 0
ABCDE

~/junha > ./oob
Input array index : 1
BCDE

~/junha > ./oob
Input array index : 2
CDE

~/junha > ./oob
Input array index : 3
DE

~/junha > ./oob
Input array index : 4
E

~/junha > █
```

# Trigger



```
#include <stdio.h>

int main()
{
    char array[6] = "ABCDE";
    int idx;

    printf("Input array index : ");
    scanf("%d",&idx);

    puts(&array[idx]);

    return 0;
}
```

배열 번지수 체크를 하지 않음

```
~/junha > ./oob
Input array index : 7
o\4w±@
~/junha > ./oob
Input array index : 8
"@j ¢@
~/junha > █
```

배열의 범위를 체크하지 않기 때문에 배열의 범위를  
벗어난 메모리에 접근 가능

gef> x/10gx 0x7ffffffffde90

0x7ffffffffde90: 0x004544434241df80

0x1eed1748cc374c00

0x7ffffffffdea0: 0x00000000000400790

0x00007ffff7a03bf7

0x7ffffffffdeb0: 0x00000000000000001

0x00007ffffffffffdf88

0x7ffffffffdec0: 0x0000000010000c000

0x000000000004006d7

0x7ffffffffded0: 0x00000000000000000

0xa702fd8861161b8b

배열 array의 영역

0x7fffffffde90: 0x004544434241df80

```
char array[6] = "ABCDE";
```

Canary!!

```
gef> x/10gx 0x7fffffffde90
0x7fffffffde90: 0x004544434241df80      0x1eed1748cc374c00
0x7fffffffdea0: 0x00000000000400790    0x00007ffff7a03bf7
0x7fffffffdeb0: 0x00000000000000001    0x00007ffffffffffdf88
0x7fffffffdec0: 0x0000000010000c000    0x000000000004006d7
0x7fffffffded0: 0x00000000000000000    0xa702fd8861161b8b
```

해당 부분을 출력하여 **Canary** 값을 알아낼 수 있다!

```
aimha@aimha:~/junha$ ./oob  
Input array index : 0  
ABCDE
```

```
gef> x/10gx 0x7ffffffffde90  
0x7ffffffffde90: 0x004544434241df80      0x1eed1748cc374c00  
0x7ffffffffdea0: 0x00000000000400790      0x00007ffff7a03bf7  
0x7ffffffffdeb0: 0x00000000000000001      0x00007ffffffffffdf88  
0x7ffffffffdec0: 0x0000000010000c000      0x000000000004006d7  
0x7ffffffffded0: 0x00000000000000000      0xa702fd8861161b8b
```



```
aimha@aimha:~/junha$ ./oob
Input array index : 1
BCDE
```

```
gef> x/10gx 0x7fffffffde90
0x7fffffffde90: 0x004544434241df80      0x1eed1748cc374c00
0x7fffffffdea0: 0x00000000000400790      0x00007ffff7a03bf7
0x7fffffffdeb0: 0x00000000000000001      0x00007ffffffffffdf88
0x7fffffffdec0: 0x0000000010000c000      0x000000000004006d7
0x7fffffffded0: 0x00000000000000000      0xa702fd8861161b8b
```

```
aimha@aimha:~/junha$ ./oob
Input array index : 2
CDE
```

```
gef> x/10gx 0x7fffffffde90
0x7fffffffde90: 0x004544434241df80      0x1eed1748cc374c00
0x7fffffffdea0: 0x00000000000400790      0x00007ffff7a03bf7
0x7fffffffdeb0: 0x00000000000000001      0x00007ffffffffffdf88
0x7fffffffdec0: 0x0000000010000c000      0x000000000004006d7
0x7fffffffded0: 0x00000000000000000      0xa702fd8861161b8b
```

```
aimha@aimha:~/junha$ ./oob  
Input array index : 3  
DE
```

```
gef> x/10gx 0x7fffffffde90  
0x7fffffffde90: 0x004544434241df80      0x1eed1748cc374c00  
0x7fffffffdea0: 0x00000000000400790      0x00007ffff7a03bf7  
0x7fffffffdeb0: 0x00000000000000001      0x00007ffffffffffdf88  
0x7fffffffdec0: 0x0000000010000c000      0x000000000004006d7  
0x7fffffffded0: 0x00000000000000000      0xa702fd8861161b8b
```

```
aimha@aimha:~/junha$ ./oob  
Input array index : 4  
E
```

```
gef> x/10gx 0x7fffffffde90  
0x7fffffffde90: 0x004544434241df80      0x1eed1748cc374c00  
0x7fffffffdea0: 0x00000000000400790      0x00007ffff7a03bf7  
0x7fffffffdeb0: 0x00000000000000001      0x00007ffffffffffdf88  
0x7fffffffdec0: 0x0000000010000c000      0x000000000004006d7  
0x7fffffffded0: 0x00000000000000000      0xa702fd8861161b8b
```

```
aimha@aimha:~/junha$ ./oob  
Input array index : 5
```

```
gef> x/10gx 0x7ffffffffde90  
0x7ffffffffde90: 0x004544434241df80      0x1eed1748cc374c00  
0x7ffffffffdea0: 0x000000000000400790      0x00007ffff7a03bf7  
0x7ffffffffdeb0: 0x000000000000000001      0x00007ffffffffffdf88  
0x7ffffffffdec0: 0x0000000010000c000       0x0000000000004006d7  
0x7ffffffffded0: 0x000000000000000000      0xa702fd8861161b8b
```



```
aimha@aimha:~/junha$ ./oob
Input array index : 6
```

```
gef> x/10gx 0x7fffffffde90
0x7fffffffde90: 0x004544434241df80      0x1eed1748cc374c00
0x7fffffffdea0: 0x000000000000400790      0x00007ffff7a03bf7
0x7fffffffdeb0: 0x000000000000000001      0x00007ffffffffffdf88
0x7fffffffdec0: 0x0000000010000c000        0x0000000000004006d7
0x7fffffffded0: 0x000000000000000000      0xa702fd8861161b8b
```

```
aimha@aimha:~/junha$ ./oob
Input array index : 7
;
@@@
```

```
gef> x/10gx 0x7fffffffde90
0x7fffffffde90: 0x004544434241df80      0x1eed1748cc374c00
0x7fffffffdea0: 0x000000000000400790      0x00007ffff7a03bf7
0x7fffffffdeb0: 0x000000000000000001      0x00007ffffffffffdf88
0x7fffffffdec0: 0x0000000010000c000        0x0000000000004006d7
0x7fffffffded0: 0x000000000000000000      0xa702fd8861161b8b
```

```
[DEBUG] Received 0xb bytes:
00000000 d0 45 7c 92 c6 02 1a 90 07 40 0a
0000000b
♦E|\x92♦x1a\x07
$
```

Canary SFP "\n"

```
gef> x/10gx 0x7fff0b0c5e80
0x7fff0b0c5e80: 0x0045444342415f70
0x7fff0b0c5e90: 0x000000000000400790
0x7fff0b0c5ea0: 0x000000000000000001
```

Canary SFP

0x1a02c6927c45d000

0x00007fa074121bf7

0x00007fff0b0c5f78

출력된 값을 16진수로 변경해보면 Canary 값이 성공적으로 출력된 것을 알 수 있다!



# IDA 사용방법

# 따로 공부하면 좋은거

1. 리눅스 기본 사용법과 명령어
2. 구글에 버퍼오버플로우 자료가 많음 찾아보면 좋다.

질문