first-fit(Use-After-Free)[English]

Unknown macro: 'html'

List

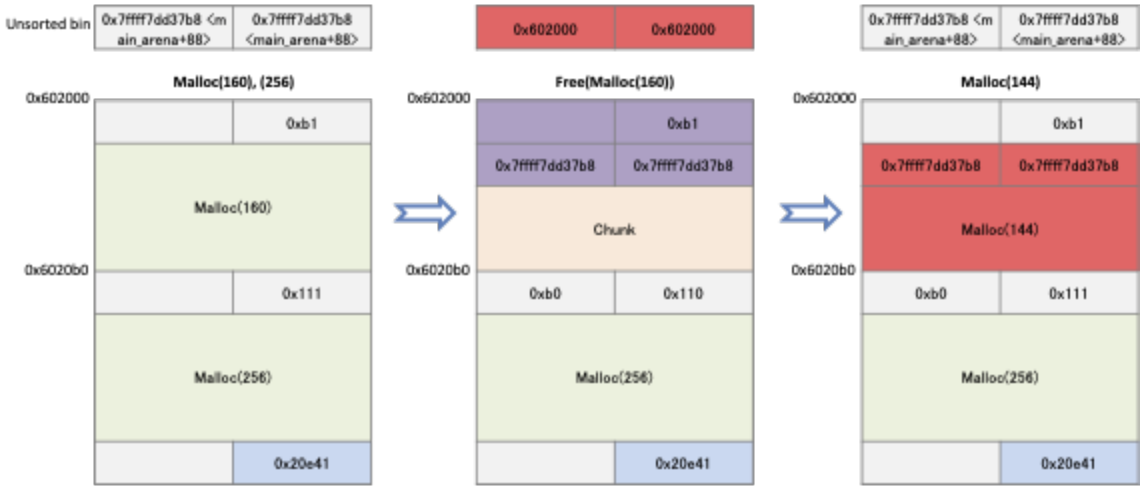
UAF(Use-After-Free)

- UAF uses previously freed memory, which can cause a variety of negative consequences, from corrupting valid data to executing arbitrary code.
- For example, if the program allocates two memories, frees the first memory, and then requests a similar amount of memory allocation, the previously freed memory is allocated.
  - And a has a pointer to the first allocated memory, and c also has a pointer to the same memory.
  - In other words, data can be changed and printed using the pointer of a.
  - This causes undefined behavior in the process.

UAF.c

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

int main()
{
    char* a = malloc(160);
    char* b = malloc(256);
    char* c;

    free(a);

    c = malloc(144);

    strcpy(a, "Secret message");
}
```

UAF-1 flow



- Saved the data after allocating memory as follows:
  - The memory is released, but the data stored in the memory is not initialized.
  - Therefore, Can be output the data in the memory pointed to by an after freeing the memory.

UAF-2.c

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

int main()
{
    char* a = malloc(160);
    strcpy(a, "Secret message");
    free(a);
    printf("%s\n",a);
}
```

Example

Example1

- The following code asks malloc () to allocate memory of 160 bytes and 256 bytes in size.
  - The returned pointer is stored in "a" and "b".
  - Free the first memory, request an allocation of 144byte memory, and the returned pointer is stored in "c".
  - Copies a string into the corresponding memory and prints out the data in the memory pointed to by "a".

UAF-1.c

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

int main()
{
        char* a = malloc(160);
        char* b = malloc(256);
        char* c;

        free(a);

        c = malloc(144);

        strcpy(c, "Secret message");
        printf("%s\n",a);
}
```

- Check the address of the memory stored in "a" at 0x4005c8, and confirm the release of the memory at 0x4005e6.
    - Checks the address of the memory stored in "c" at 0x4005f0 and checks the data stored in that memory at 0x400616.
    - Check for UAF at 0x40061d.

Breakpoints

```
lazenca0x0@ubuntu:~/Book/4.UAF$ gcc -o UAF UAF.c
lazenca0x0@ubuntu:~/Book/4.UAF$ gdb -q UAF
Reading symbols from UAF...(no debugging symbols found)...done.
gdb-peda$ disassemble main
Dump of assembler code for function main:
   0x00000000004005b6 <+0>:        push   rbp
   0x00000000004005b7 <+1>:        mov    rbp,rsp
   0x00000000004005ba <+4>:        sub    rsp,0x20
   0x00000000004005be <+8>:        mov    edi,0xa0
   0x00000000004005c3 <+13>:        call   0x4004a0 <malloc@plt>
   0x00000000004005c8 <+18>:        mov    QWORD PTR [rbp-0x18],rax
   0x00000000004005cc <+22>:        mov    edi,0x100
   0x00000000004005d1 <+27>:        call   0x4004a0 <malloc@plt>
   0x00000000004005d6 <+32>:        mov    QWORD PTR [rbp-0x10],rax
   0x00000000004005da <+36>:        mov    rax,QWORD PTR [rbp-0x18]
   0x00000000004005de <+40>:        mov    rdi,rax
   0x00000000004005e1 <+43>:        call   0x400470 <free@plt>
   0x00000000004005e6 <+48>:        mov    edi,0x90
   0x00000000004005eb <+53>:        call   0x4004a0 <malloc@plt>
   0x00000000004005f0 <+58>:        mov    QWORD PTR [rbp-0x8],rax
   0x00000000004005f4 <+62>:        mov    rax,QWORD PTR [rbp-0x8]
   0x00000000004005f8 <+66>:        movabs rdx,0x6d20746572636553
   0x0000000000400602 <+76>:        mov    QWORD PTR [rax],rdx
   0x0000000000400605 <+79>:        mov    DWORD PTR [rax+0x8],0x61737365
   0x000000000040060c <+86>:        mov    WORD PTR [rax+0xc],0x6567
   0x0000000000400612 <+92>:        mov    BYTE PTR [rax+0xe],0x0
   0x0000000000400616 <+96>:        mov    rax,QWORD PTR [rbp-0x18]
   0x000000000040061a <+100>:        mov    rdi,rax
   0x000000000040061d <+103>:        call   0x400480 <puts@plt>
   0x0000000000400622 <+108>:        mov    eax,0x0
   0x0000000000400627 <+113>:        leave
   0x0000000000400628 <+114>:        ret
End of assembler dump.
gdb-peda$ b *0x00000000004005c8
Breakpoint 1 at 0x4005c8
gdb-peda$  b *0x00000000004005e6
Breakpoint 2 at 0x4005e6
gdb-peda$ b *0x00000000004005f0
Breakpoint 3 at 0x4005f0
gdb-peda$ b *0x0000000000400616
Breakpoint 4 at 0x400616
gdb-peda$ b *0x000000000040061d
Breakpoint 5 at 0x40061d
gdb-peda$
```

- When you ask malloc() for memory allocation of size 160 bytes, the allocator returns 0x602010, whose address is stored in "a".
    - The memory is registered in the Unsorted bin if it is freed after allocating a different size of memory.

Free memory

```
gdb-peda$ r
Starting program: /home/lazenca0x0/Book/4.UAF/UAF

Breakpoint 1, 0x00000000004005c8 in main ()
gdb-peda$ i r rax
rax            0x602010        0x602010
gdb-peda$ c
Continuing.

Breakpoint 2, 0x00000000004005e6 in main ()
gdb-peda$ p main_arena.bins[0]
$1 = (mchunkptr) 0x602000
gdb-peda$ p main_arena.bins[1]
$2 = (mchunkptr) 0x602000
gdb-peda$ p/d main_arena.bins[0].size - 1
$3 = 176
gdb-peda$
```

- If request malloc() for memory allocation of 144 bytes in size, the memory that was registered in the Unsorted bin is reallocated.
- A request was made to allocate a memory whose size differs from the amount of memory placed in the unsorted bin, but the memory registered in the unsorted bin is returned.
    - In order for the allocator to use memory efficiently, it will preferentially use that memory if the size of the requested memory is less than or equal to the size of the memory placed in the Unsorted bin.
    - If the memory placed in the unsorted bin is large, the allocator allocates memory as much as the requested memory, and places the remaining memory in main_arena.last_remainder.
- In this example, the memory placed in the unsorted bin is larger than the newly requested memory, so the memory of the unsorted bin is used to return the memory (0x602010).
    - After allocating the newly requested memory, the remaining size of memory is 16 bytes.
    - In this case, malloc will not be able to reuse the remaining memory, so it will return the amount of memory placed in the Unsorted bin (176) without partitioning the memory.
    - The address of this allocated memory is stored in variable "c".
    - The address is the same address stored in variable "a".

Reallocate memory

```
gdb-peda$ c
Continuing.

Breakpoint 3, 0x00000000004005f0 in main ()
gdb-peda$ i r rax
rax            0x602010        0x602010
gdb-peda$ p main_arena.last_remainder
$4 = (mchunkptr) 0x0
gdb-peda$ p/d 176 - 160
$5 = 16
gdb-peda$ x/4gx 0x602010 - 0x10
0x602000:        0x0000000000000000        0x00000000000000b1
0x602010:        0x00007ffff7dd1b78        0x00007ffff7dd1b78
gdb-peda$ p/d 0xb0
$7 = 176
gdb-peda$
```

- Enter the string "Secret message" in the reallocated area.
    - The data stored in "a"(0x602010) is output.
- If you request the output of the data stored in the variable "a", the string "Secret message" is output.
    - The reason why the data is displayed is that the memory pointed to by the variable "a" is released but the value stored in the variable "a" is not initialized.
    - That is, the variable "a" still has the address of the first allocated memory (0x602010).
    - This allows you to check the data stored in variable "c" via variable "a".

Output the data stored in the pointer of 'a'

```
gdb-peda$ c
Continuing.

Breakpoint 4, 0x0000000000400616 in main ()
gdb-peda$ i r rax
rax            0x602010        0x602010
gdb-peda$ x/s 0x602010
0x602010:        "Secret message"
gdb-peda$ c
Continuing.

Breakpoint 5, 0x000000000040061d in main ()
gdb-peda$ i r rdi
rdi            0x602010        0x602010
gdb-peda$ ni
Secret message

gdb-peda$
```

Example2

- This example uses UAF-2.c.
    - Check the allocated pointer at 0x4005c8 and check the data in the freed memory at 0x4005f5.
    - Check the data output at 0x400601.

Breakpoints

```
lazenca0x0@ubuntu:~/Book/4.UAF$ gdb -q ./UAF-2
Reading symbols from ./UAF-2...(no debugging symbols found)...done.
gdb-peda$ disassemble main
Dump of assembler code for function main:
   0x00000000004005b6 <+0>:        push   rbp
   0x00000000004005b7 <+1>:        mov    rbp,rsp
   0x00000000004005ba <+4>:        sub    rsp,0x10
   0x00000000004005be <+8>:        mov    edi,0xa0
   0x00000000004005c3 <+13>:        call   0x4004a0 <malloc@plt>
   0x00000000004005c8 <+18>:        mov    QWORD PTR [rbp-0x8],rax
   0x00000000004005cc <+22>:        mov    rax,QWORD PTR [rbp-0x8]
   0x00000000004005d0 <+26>:        movabs rdx,0x6d20746572636553
   0x00000000004005da <+36>:        mov    QWORD PTR [rax],rdx
   0x00000000004005dd <+39>:        mov    DWORD PTR [rax+0x8],0x61737365
   0x00000000004005e4 <+46>:        mov    WORD PTR [rax+0xc],0x6567
   0x00000000004005ea <+52>:        mov    BYTE PTR [rax+0xe],0x0
   0x00000000004005ee <+56>:        mov    rax,QWORD PTR [rbp-0x8]
   0x00000000004005f2 <+60>:        mov    rdi,rax
   0x00000000004005f5 <+63>:        call   0x400470 <free@plt>
   0x00000000004005fa <+68>:        mov    rax,QWORD PTR [rbp-0x8]
   0x00000000004005fe <+72>:        mov    rdi,rax
   0x0000000000400601 <+75>:        call   0x400480 <puts@plt>
   0x0000000000400606 <+80>:        mov    eax,0x0
   0x000000000040060b <+85>:        leave
   0x000000000040060c <+86>:        ret
End of assembler dump.
gdb-peda$ b *0x00000000004005c8
Breakpoint 1 at 0x4005c8
gdb-peda$ b *0x00000000004005f5
Breakpoint 2 at 0x4005f5
gdb-peda$ b *0x0000000000400601
Breakpoint 3 at 0x400601
gdb-peda$
```

- The address of allocated memory is 0x602010, and the memory is released after saving the data.
    - The stored data is not initialized even after the memory is released.

Allocate memory and Free the memory after stored data in memory.

```
gdb-peda$ r
Starting program: /home/lazenca0x0/Book/4.UAF/UAF-2

Breakpoint 1, 0x00000000004005c8 in main ()
gdb-peda$ i r rax
rax            0x602010        0x602010
gdb-peda$ c
Continuing.

Breakpoint 2, 0x00000000004005f5 in main ()
gdb-peda$ i r rdi
rdi            0x602010        0x602010
gdb-peda$ x/i $rip
=> 0x4005f5 <main+63>:        call   0x400470 <free@plt>
gdb-peda$ x/s 0x602010
0x602010:        "Secret message"
gdb-peda$ ni

0x00000000004005fa in main ()
gdb-peda$ x/30gx 0x602010 - 0x10
0x602000:        0x0000000000000000        0x0000000000021001
0x602010:        0x6d20746572636553        0x0000656761737365
0x602020:        0x0000000000000000        0x0000000000000000
0x602030:        0x0000000000000000        0x0000000000000000
0x602040:        0x0000000000000000        0x0000000000000000
0x602050:        0x0000000000000000        0x0000000000000000
0x602060:        0x0000000000000000        0x0000000000000000
0x602070:        0x0000000000000000        0x0000000000000000
0x602080:        0x0000000000000000        0x0000000000000000
0x602090:        0x0000000000000000        0x0000000000000000
0x6020a0:        0x0000000000000000        0x0000000000000000
0x6020b0:        0x0000000000000000        0x0000000000020f51
0x6020c0:        0x0000000000000000        0x0000000000000000
0x6020d0:        0x0000000000000000        0x0000000000000000
0x6020e0:        0x0000000000000000        0x0000000000000000
gdb-peda$ x/s 0x602010
0x602010:        "Secret message"
gdb-peda$ p main_arena.top
$1 = (mchunkptr) 0x602000
gdb-peda$
```

- The variable "a" has an address that points to the freed memory and can output the data of that area.

Output the data stored in the freed memory.

```
gdb-peda$ c
Continuing.

Breakpoint 3, 0x0000000000400601 in main ()
gdb-peda$ i r rdi
rdi            0x602010        0x602010
gdb-peda$ x/i $rip
=> 0x400601 <main+75>:        call   0x400480 <puts@plt>
gdb-peda$ ni
Secret message
gdb-peda$
```

Related information

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

Unknown macro: 'html'