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

List

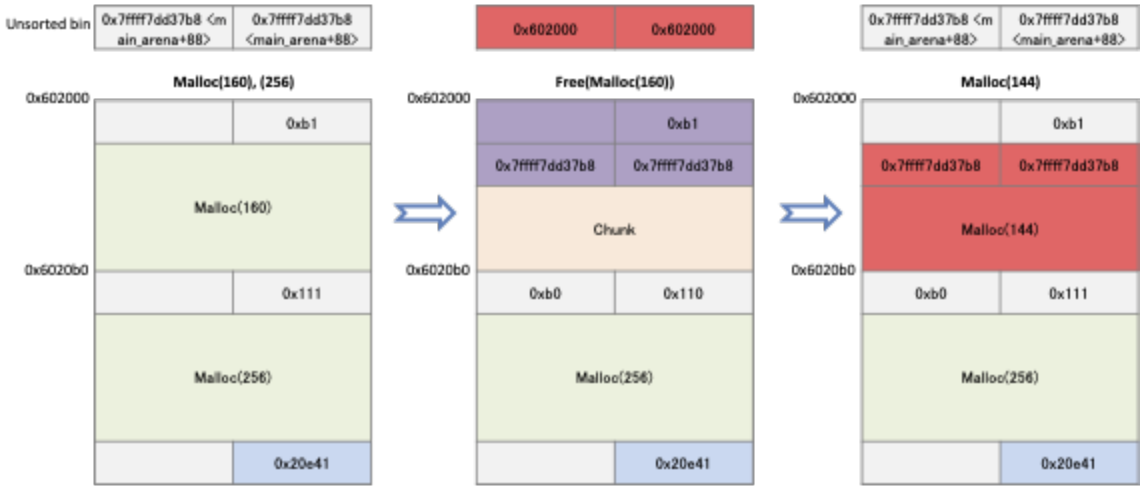
UAF(Use-After-Free)

- UAF는 이전에 비워진 메모리를 사용하여 유효한 데이터의 손상에서 임의의 코드 실행에 이르기까지 여러 가지 부정적인 결과가 발생할 수 있습니다.
- 예를 들어 다음 코드와 같이 프로그램이 2개의 메모리를 할당하고, 첫번째 메모리를 해제한 후 비슷한 크기의 메모리 할당을 요청하면 이전에 해제된 메모리가 할당됩니다.
    - 그리고 a에는 처음 할당받은 메모리의 포인터가 가지고 있으며, c도 같은 메모리의 포인터를 가지게됩니다.
    - 즉, a가 가지고 있는 포인터를 이용하여 데이터 변경 및 출력이 가능합니다.
    - 이는 프로세스에서 정의되지 않은 동작을 유발합니다.

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



- 다음과 같이 메모리를 할당한 후에 데이터를 보관합니다.
    - 그리고 해당 메모리를 해제하지만, 메모리에 보관된 데이터는 초기화 되지 않습니다.
    - 그렇기 때문에 메모리를 해제한 후에도 a가 가리키는 메모리의 데이터를 출력할 수 있습니다.

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

- 다음 코드는 크기가 160byte, 256byte인 메모리의 할당을 malloc()에 요청합니다.
    - 반환된 pointer는 a, b에 저장됩니다.
    - 첫번째 메모리가 해제되고, 144byte 메모리를 할당을 요청하고 반환된 pointer는 변수 c에 저장됩니다.
    - 해당 메모리에 문자열을 복사되고, a가 가리키는 메모리의 데이터를 출력합니다.

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);
}
```

- 0x4005c8에서 "a"에 저장되는 메모리의 주소를 확인하고, 0x4005e6에서 해당 메모리의 해제를 확인합니다.
    - 0x4005f0에서 "c"에 저장되는 메모리의 주소를 확인하고, 0x400616에서 해당 메모리에 저장된 데이터를 확인합니다.
    - 0x40061d에서 UAF를 확인합니다.

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$
```

- 크기가 160byte인 메모리 할당을 malloc()에 요청하면 할당자는 0x602010를 반환하며, 해당 주소는 "a"에 저장됩니다.
    - 해당 메모리는 다른 크기의 메모리를 할당한 후에 해제하면Unsorted bin에 등록됩니다.

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
$3 = 176
gdb-peda$
```

- 크기가 144byte인 메모리 할당을 malloc()에 요청하면, Unsorted bin에 등록되었던 메모리가 재할당됩니다.
- Unsorted bin에 배치된 메모리의 크기와 다른 크기의 메모리 할당을 요청하였는데 Unsorted bin에 등록된 메모리가 반환되었습니다.
    - 이것은 할당자가 메모리를 효율적으로 사용하기 위해 요청된 메모리의 크기가 Unsorted bin에 배치된 메모리의 크기와 같거나 작을 경우 해당 메모리를 우선적으로 사용합니다.
    - Unsorted bin에 배치된 메모리의 크기가 클 경우 할당자는 요청된 메모리의 크기만큼 메모리를 할당되고, 남은 메모리는 main_arena.last_remainder 배치합니다.
- 이 예제에서는 Unsorted bin에 배치된 메모리의 크기가 새로 요청된 메모리의 크기보다 크기 때문에 Unsorted bin의 메모리를 사용하여 메모리(0x602010)를 반환하였습니다.
    - 새로 요청된 메모리를 할당한 후에 남은 메모리의 크기는 16byte입니다.
    - 이 경우 malloc은 남은 메모리를 재사용 할 수 없기 때문에 메모리를 분할하지 않고 Unsorted bin에 배치된 메모리의 크기(176)로 반환합니다.
    - 이렇게 할당받은 메모리의 주소는 c에 저장됩니다.
    - 해당 주소는 변수 "a"에 저장된 주소와 같은 주소입니다.

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
$7 = 176
gdb-peda$
```

- "Secret message"이라는 문자열을 재할당 받은 영역에 입력합니다.
    - 그리고 a(0x602010)에 저장된 데이터를 출력합니다.
- 변수"a"에 저장된 데이터의 출력을 요청하면 "Secret message"라는 문자열이 출력됩니다.
    - 이렇게 데이터가 출력되는 이유는 변수"a"가 가리키는 메모리를 해제하였으나 변수"a"에 저장된 값은 초기화하지 않았기 때문입니다.
    - 즉, 변수"a"에는 아직 처음에 할당받은 메모리의 주소(0x602010)가 남아있습니다.
    - 이로 인해 변수"a"를 통해 변수"c"에 저장한 데이터를 확인할 수 있습니다.

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

- 이 예제에서는 UAF-2.c를 사용합니다.
    - 0x4005c8에서 할당된 pointer를 확인하고, 0x4005f5에서는 해제된 메모리의 데이터를 확인합니다.
    - 0x400601에서는 데이터 출력을 확인합니다.

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$
```

- 할당된 메모리의 주소는 0x602010이며, 해당 메모리를 데이터를 저장한 후에 메모리를 해제하였습니다.
    - 저장된 데이터는 메모리가 해제 된 후에도 초기화되지 않습니다.

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
$1 = (mchunkptr) 0x602000
gdb-peda$
```

- 변수"a"는 해제된 메모리를 가리키는 주소를 가지고 있으며, 해당 영역의 데이터를 출력할 수 있습니다.

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