Câu lạc bộ An Toàn
Thông Tin - ĐHBKHN

Major Heap Changes in Linux's glibc

Mitigations and techniques (not really)

Presenter: Nguyen Duc Kien
@Lieu

Nội dung

01 Heap bugs

Xem xét lại các lỗi heap và phương pháp khai thác

02 Major changes

Patches và bypass qua các phiên bản

03 Conclusion

Resources, further reading, QnA

Heap bugs

- Heap overflow là tràn bộ nhớ (giống BOF) nhưng xảy ra trên vùng heap.

```
4   void main() {
5       setbuf(stdin, NULL);
6       setbuf(stdout, NULL);
7
8       char *buffer = (char *)malloc(0x10);
9       malloc(0x10);
10      gets(buffer);
11      return;
12  }
```

- Dẫn đến khả năng ghi đè dữ liệu của các chunk khác.

```
0x56017404a280  0x0000000000000000  0x0000000000000000  ................
0x56017404a290  0x0000000000000000  0x0000000000000021  ........!.......
0x56017404a2a0  0x4141414141414141  0x4141414141414141  AAAAAAAAAAAAAAAA
0x56017404a2b0  0x4141414141414141  0x4141414141414141  AAAAAAAAAAAAAAAA
0x56017404a2c0  0x4141414141414141  0x4141414141414141  AAAAAAAAAAAAAAAA
0x56017404a2d0  0x4141414141414141  0x4141414141414141  AAAAAAAAAAAAAAAA
0x56017404a2e0  0x0000000000000000  0x0000000000000000  ................
0x56017404a2f0  0x0000000000000000  0x0000000000000000  ................
0x56017404a300  0x0000000000000000  0x0000000000000000  ................
```

Heap bugs

- Use after free xảy ra khi ta free 1 chunk nhưng không gán NULL cho pointer trỏ vào chunk đó.
- Nếu ta vẫn tương tác với pointer → vẫn tương tác được với chunk.
- Edit chunk sau khi free, arbitary read, arbitary write

```
void main() {
    setbuf(stdin, NULL);
    setbuf(stdout, NULL);

    char *buffer = (char *)malloc(0x500);
    malloc(0x10);
    free(buffer);
    puts(buffer);
    return;
}
```

```
0x0000000000000000  0x0000000000000000
0x0000000000000000  0x0000000000000000
0x0000000000000000  0x0000000000000511
0x00007f048a258ce0  0x00007f048a258ce0
0x0000000000000000  0x0000000000000000
0x0000000000000000  0x0000000000000000
0x0000000000000000  0x0000000000000000
0x0000000000000000  0x0000000000000000
0x0000000000000000  0x0000000000000000
0x0000000000000000  0x0000000000000000
0x0000000000000000  0x0000000000000000
```

```
[DEBUG] Received 0x7 bytes:
    00000000  e0 8c 25 8a  04 7f 0a
```

Heap bugs

- Double free: sau khi chương trình free một chunk nhưng không gán NULL cho pointer.
- Tiếp tục free pointer đó → (tùy trường hợp chương trình sẽ crash hoặc) chunk đó sẽ xuất hiện trong bins nhiều lần.
- Tận dụng bug đó để edit chunk (sau khi free).
- Double free và Use after free khác về cách thực hiện, hậu quả là y hệt nhau.

```
pwndbg> bins
fastbins
0x20: 0x17468000 —▸ 0x17468020 ◂— 0x17468000
unsortedbin
empty
```

Heap bugs

- Khi một chunk được free -> chúng sẽ được đưa vào bins (để chờ reuse)
- Bins là các linked list → bins quản lí các free chunk bằng con trỏ
- (tcache và fastbin là singly linked list, unsorted, small, large bin là doubly linked list)
- Nếu ta có thể control các con trỏ này → ép hàm malloc trả về các giá trị (địa chỉ) theo ý muốn.

DEMO

glibc 2.23

“no tcache”

- Fastbin dup: double free của fastbin
  - So sánh với chunk nằm đầu bins → free một padding chunk ở giữa
- __malloc_hook: allows a program to intercept all allocation/free calls that happen during execution (?)
  - Một target đáng để tâm (mà mình) thường dùng
- one_gadget: tool for easier exploit

DEMO

glibc 2.26

“goat tcache”

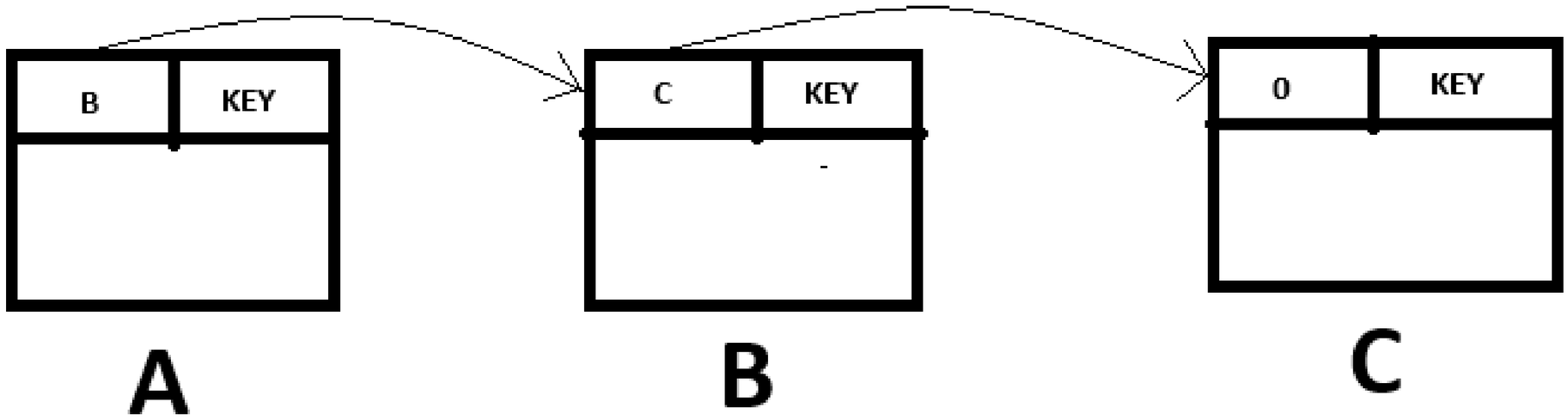
- Tcache:
  - Một loại bins mới được thêm vào để tối ưu hiệu suất
- Tcache double free
- Tcache key
  - Nỗ lực nhằm ngăn chặn DBF
- Tcache poisoning
  - kỹ thuật sửa fd_pointer để trick tcache trả về nơi mình muốn malloc lần tiếp theo

DEMO

glibc 2.32

“safe linking xuất hiện, trong nỗ lực ngăn chặn heap exploit”

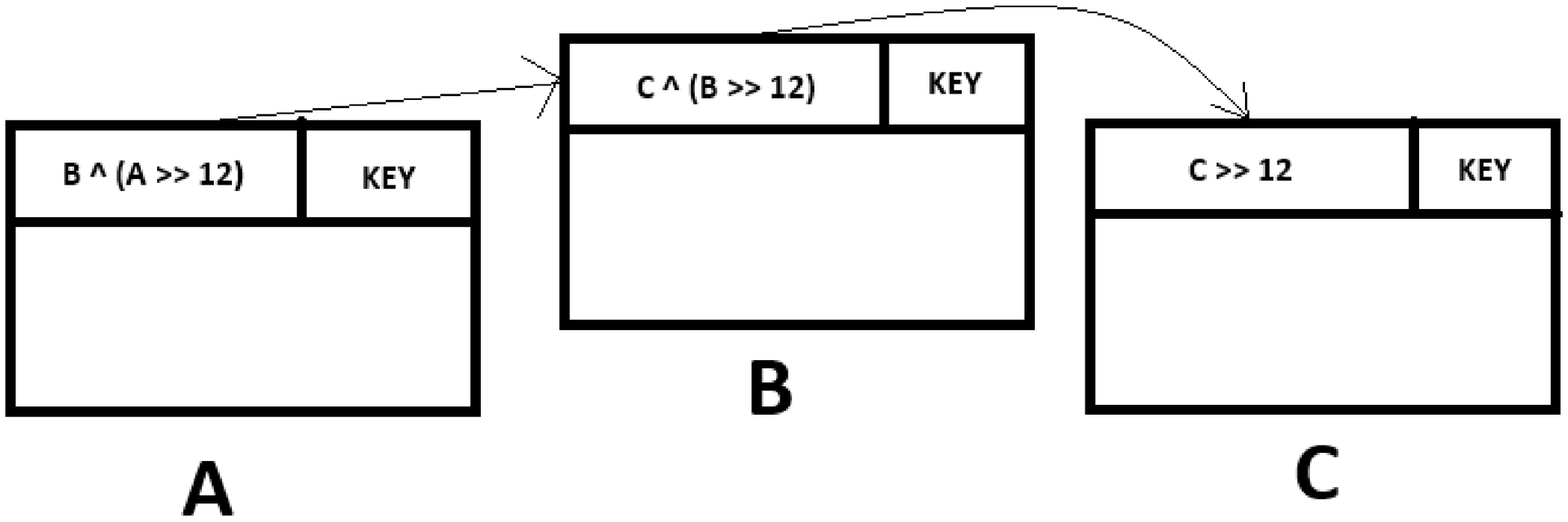
- Cơ chế bảo vệ fd_pointer của tcache và fastbin, “xáo trộn”, khó để “poison” hơn
- before:



glibc 2.32

“safe linking xuất hiện, trong nỗ lực ngăn chặn heap exploit”

- after:



- mangled_ptr = next_ptr ^ (current_ptr >> 12)

glibc 2.32

“safe linking xuất hiện, trong nỗ lực ngăn chặn heap exploit”

- Hiện giờ, muốn poison foward pointer của tcache hoặc fastbin, đều cần heap leak.
- Có 2 cách:
  - Leak mangled_ptr và dùng hàm deobfuscate (miễn là 2 pointer đều không khác nhau quá 12 bit cuối)
  - Leak mangled_ptr của chunk cuối trong bins và dịch trái 12 bit

```
def deobfuscate(val):
    mask = 0xfff << 52
    while mask:
        v = val & mask
        val ^= (v >> 12)
        mask >>= 12
    return val
```

glibc 2.34+

“hard time”

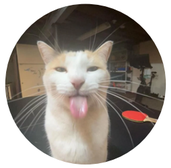
- Kể từ 2.34 trở đi, vì vấn đề bảo mật mà các _hook đã bị xóa
- RCE đã trở nên khó khăn hơn, nhưng vẫn khả thi
- Khi control được heap layout, thì khả năng ta có arbitary read, write rất cao
- Cân nhắc chọn những target:
  - GOT (nếu partial relro)
  - GOT libc (?)
  - Leak stack address qua environ sau đó viết ROP (my favourite)
  - File struct !!!
- Tùy cơ ứng biến

Reference

maybe this will help

- https://github.com/shellphish/how2heap
- https://github.com/guyinatuxedo/Shogun
- https://github.com/limitedeternity/HeapLAB
- https://0x434b.dev/overview-of-glibc-heap-exploitation-techniques/
- https://hackmd.io/@trhoanglan04/heap_exploit
- https://github.com/johnathanhuutri/CTFNote/tree/master/Heap-Exploitation
- ...

Thank You!



Presenter: Nguyen Duc Kien

@Lieu