

2025.11.05

Heap Basics

가디언 시스템 보안 & 취약점 분석 세미나 4.2

임준서

Disclaimer

Regarding source code.

glibc 2.27 기준 (ubuntu 18.04)

이후 2.35까지 다룰 예정

소스코드는 아래에서 확인할 수 있다.

<https://elixir.bootlin.com/glibc/glibc-2.27/source/>

<https://elixir.bootlin.com/glibc/glibc-2.27/source/malloc/malloc.c>

version이 높을수록 제약이 많아진다.

임준서

Disclaimer

Useful Resources

How2Heap

Lazenca.net

github.com/shellphish/how2heap

www.lazenca.net/display/TEC/02.Heap+Exploitation

Disclaimer

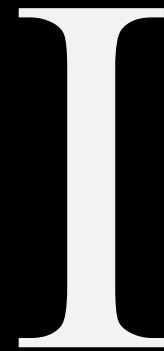
Important!

glibc 소스코드를 참조할 것

직접 코드를 읽으며 이해해야 한다

Basic Concepts

Glibc 2.27



Malloc

결국 syscall의 wrapper

malloc은 최종적으로 mmap, brk 실행

mmap = 큰 메모리 영역용, brk = 작은 메모리 영역용

너무 느리다

여러 cache 도입 → 속도 향상, 안정성

tcachebin, fastbin, unsortedbin, small/largebin

window에선 다른 syscall → 다른 동작

Memory MAP: 새로운 VMA(Virtual Memory Area) 생성. 4KB 단위

BReaK: 힙 영역에 남아있는 메모리 할당

Basic Heap

`glibc`

메모리 할당의 단위: Chunk

큰 Chunk를 잘라서 `malloc`으로 할당

정확한 정의는 곧 설명

임준서

Basic Heap

glibc

0x10 정렬

청크의 마지막 주소는 0으로 끝나야한다.

First-Fit Algorithm

[TMI: geeks for geeks / operating system](http://www.geeksforgeeks.org/operating-systems/first-fit-allocation-in-operating-systems/)

LIFO 구조

www.geeksforgeeks.org/operating-systems/first-fit-allocation-in-operating-systems/

임준서

Basic Heap

glibc



```
1 // fast_fit.c
2 // gcc -o fast_fit fast_fit.c
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 int main() {
7     void* ptr = malloc(0x80);
8     if (!ptr) {
9         perror("malloc");
10        return 1;
11    }
12    printf("Allocated 0x80 bytes at %p\n", ptr);
13    free(ptr);
14    void *ptr2 = malloc(0x80);
15    if (!ptr2) {
16        perror("malloc");
17        return 1;
18    }
19    printf("Allocated 0x80 bytes at %p\n", ptr2);
20    free(ptr2);
21    return 0;
22 }
```

Basic Heap

`glibc`

`Allocated 0x80 bytes at 0x55e46aff42a0`

`Allocated 0x80 bytes at 0x55e46aff42a0`

보통 0x55... 끝

임준서

Chunk

glibc 2.27



```
1  struct malloc_chunk {
2
3      INTERNAL_SIZE_T      mchunk_prev_size; /* Size of previous chunk (if free). */
4      INTERNAL_SIZE_T      mchunk_size;      /* Size in bytes, including overhead. */
5
6      struct malloc_chunk* fd;                /* double links -- used only if free. */
7      struct malloc_chunk* bk;
8
9      /* Only used for large blocks: pointer to next larger size. */
10     struct malloc_chunk* fd_nextsize; /* double links -- used only if free. */
11     struct malloc_chunk* bk_nextsize;
12 };
```

Chunk

glibc 2.27

0x00	mchunk_prev_size	mchunk_size
0x10	fd	bk
0x20	fd_nextsize	bk_nextsize

metadata 외 부분은 free 되었을 때만 존재

위 도식은 free된 청크를 나타냄

어느 bin에 위치하냐에 따라 각 항목의 쓰임 상이

임준서

Caching

Heap

매번 syscall로 메모리 할당은 무거움

tcachebin

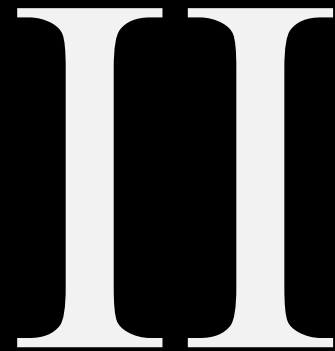
tcachebin > fastbin > small/largebin > unsortedbin 순으로 찾는다.

Thread CACHE

임준서

Tcachebins

Glibc 2.27



Tcachebin

Thread CACHE, heap

크기별로 저장

Single-linked list 기반

Default 크기는 7

Fastbin으로 넘어가기 위해선 7개의 entries를 채우면 된다.

malloc.c L318: #define TCACHE_FILL_COUNT 7

2.27는 tacahe에 대한 보안이 매우 약하다.

Chunk

glibc 2.27 – tcachebin

prev_size: 0x0
size: 0x31
fd: 0x0
bk: 0x8047730b86034d65



```
1  size_t* ptr = malloc(0x20);  
2  free(ptr);  
3  printf("prev_size: %llx\n", *(ptr - 2));  
4  printf("size: %llx\n", *(ptr - 1));  
5  printf("fd: %llx\n", *(ptr));  
6  printf("bk: %llx\n", *(ptr + 1));
```

Size가 0x30이 아닌 이유는 flag 비트 때문
여기서 bk는 garbage 값이다.

Chunk

glibc 2.27 – tcachebin

```
prev_size: 0x0  
size: 0x31  
fd: 0x0  
bk: 0x8047730b86034d65
```

tcache에서는 fd를
next entry를 저장하는데 사용

Size가 0x30이 아닌 이유는 flag 비트 때문
여기서 bk는 garbage 값이다.

Tcachebin

Free

tcachebin

0x20	0x0
0x30	0x0
...	...

heap_base

0xA0	user_data	
0x90		
0x80	mchunk_prev_size	0x30
0x70	user_data	
0x60		
0x50	user_data	
0x40		
0x30	mchunk_prev_size	0x30
0x20	user_data	
0x10		
0x00	mchunk_prev_size	0x30

편의상 size는 flag 없이 표기함

임준서

Tcachebin

Free

free(p1);

tcachebin

0x20	0x0
0x30	heap_base+0x10
...	...

heap_base

0xA0	user_data	
0x90		
0x80	mchunk_prev_size	0x30
0x70	user_data	
0x60	mchunk_prev_size	0x20
0x50	user_data	
0x40		
0x30	mchunk_prev_size	0x30
0x20		
0x10	0x0	-
0x00	mchunk_prev_size	0x31

Tcachebin

Free

```
free(p1);
free(p2);
```

tcachebin

0x20	0x0
0x30	heap_base+0x40
...	...

heap_base

0xA0	user_data	
0x90		
0x80	mchunk_prev_size	0x30
0x70	user_data	
0x60	mchunk_prev_size	0x20
0x50		
0x40	heap_base+0x10	-
0x30	mchunk_prev_size	0x30
0x20		
0x10	0x0	-
0x00	mchunk_prev_size	0x31

Tcachebin

Free

```
free(p1);  
free(p2);  
free(p3);
```

tcachebin

0x20	heap_base+0x70
0x30	heap_base+0x10
...	...

heap_base

0xA0	user_data	
0x90		
0x80	mchunk_prev_size	0x30
0x70	0x0	
0x60	mchunk_prev_size	0x20
0x50		
0x40	heap_base+0x10	-
0x30	mchunk_prev_size	0x30
0x20		
0x10	0x0	-
0x00	mchunk_prev_size	0x31

Tcachebin

Free

```
free(p1);  
free(p2);  
free(p3);  
free(p4);
```

tcachebin

0x20	heap_base+0x70
0x30	heap_base+0x90
...	...

heap_base

0xA0		
0x90	heap_base+0x40	-
0x80	mchunk_prev_size	0x30
0x70	0x0	-
0x60	mchunk_prev_size	0x20
0x50		
0x40	heap_base+0x10	-
0x30	mchunk_prev_size	0x30
0x20		
0x10	0x0	-
0x00	mchunk_prev_size	0x31

Tcachebin

Reallocation

Free한 후 동일한 크기 할당

tcachebin을 찾는다

size를 tcachebin idx로 변환

Tcachebin

Reallocation



```
1  if (tc_idx < mp_.tcache_bins
2      /*&& tc_idx < TCACHE_MAX_BINS*/ /* to appease gcc */
3      && tcache
4      && tcache->entries[tc_idx] != NULL)
5      {
6          return tcache_get (tc_idx);
7      }
```

Tcachebin의 존재와 해당 엔트리의 존재만 확인

libc 2.35부터는 entry 개수도 확인

Tcachebin

Malloc

tcachebin

0x20	heap_base+0x70
0x30	heap_base+0x90
...	...

heap_base

libc 2.35부터는 entry 개수도 확인

0xA0		
0x90	heap_base+0x40	-
0x80	mchunk_prev_size	0x30
0x70	0x0	-
0x60	mchunk_prev_size	0x20
0x50		
0x40	heap_base+0x10	-
0x30	mchunk_prev_size	0x30
0x20		
0x10	0x0	-
0x00	mchunk_prev_size	0x31

임준서

Tcachebin

Malloc

```
malloc(0x30);  
→ heap_base+0x90
```

tcachebin

0x20	heap_base+0x70
0x30	heap_base+0x40
...	...

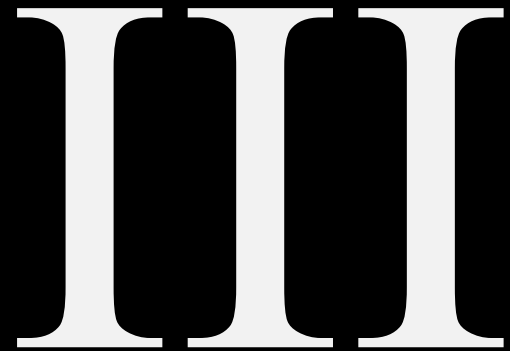
heap_base

heap_base+0x40은 여전히 남아있다.

0xA0		
0x90	heap_base+0x40	-
0x80	mchunk_prev_size	0x30
0x70	0x0	-
0x60	mchunk_prev_size	0x20
0x50		
0x40	heap_base+0x10	-
0x30	mchunk_prev_size	0x30
0x20		
0x10	0x0	-
0x00	mchunk_prev_size	0x31

Tcache posioning

Glibc 2.27



Tcache poisoning

concept

Tcachebin은 성능이 우선인 bin

보안 관련 체크가 적다

Tcache poisoning

concept

그러려면 2개의 Concept을 숙지해야 함!

Use After Free

Double Free

Use After Free

concept

Free 한 pointer을 재사용하는 것

당연히 의도하지 않은 상황

어떤 일이 벌어질까?

Tcachebin

원래의 heap bin 구조

tcachebin

0x20	heap_base+0x70
0x30	heap_base+0x90
...	...

heap_base

0xA0		
0x90	UAF!	-
0x80	mchunk_prev_size	0x30
0x70	0x0	-
0x60	mchunk_prev_size	0x20
0x50		
0x40	heap_base+0x10	-
0x30	mchunk_prev_size	0x30
0x20		
0x10	0x0	-
0x00	mchunk_prev_size	0x31

Tcachebin

원래의 heap bin 구조
이후 malloc을 하면

tcachebin

0x20	heap_base+0x70
0x30	heap_base+0x90
...	...

heap_base

0xA0	User allocated	
0x90		
0x80	mchunk_prev_size	0x30
0x70	0x0	-
0x60	mchunk_prev_size	0x20
0x50		
0x40	heap_base+0x10	-
0x30	mchunk_prev_size	0x30
0x20		
0x10	0x0	-
0x00	mchunk_prev_size	0x31

Tcachebin

원래의 heap bin 구조
이후 malloc을 하면
bin이 corrupted 되어 있다

tcachebin

0x20	heap_base+0x70
0x30	UAF!
...	...

heap_base

??	??	??
??	??	??

0xA0	User allocated	
0x90		
0x80	mchunk_prev_size	0x30
0x70	0x0	-
0x60	mchunk_prev_size	0x20
0x50		
0x40	heap_base+0x10	-
0x30	mchunk_prev_size	0x30
0x20		
0x10	0x0	-
0x00	mchunk_prev_size	0x31

Tcachebin

한번 더 free?
해당 청크를 반환하며,
파란 부분이 다음 next가 됨

tcachebin

0x20	heap_base+0x70
0x30	???
...	...

heap_base

User allocated

0xA0	User allocated	
0x90		
0x80	mchunk_prev_size	0x30
0x70	0x0	-
0x60	mchunk_prev_size	0x20
0x50		
0x40	heap_base+0x10	-
0x30	mchunk_prev_size	0x30
0x20		
0x10	0x0	-
0x00	mchunk_prev_size	0x31

Double Free

concept

UAF와 비슷한 상황을 연출

Tcachebin

아직 free전

tcachebin

0x20	-
0x30	-
...	...

heap_base

0xA0	-	-
0x90	-	-
0x80	mchunk_prev_size	0x30
0x70	-	-
0x60	mchunk_prev_size	0x20
0x50	-	-
0x40	-	-
0x30	mchunk_prev_size	0x30
0x20	-	-
0x10	-	-
0x00	mchunk_prev_size	0x31

임준서

Tcachebin

아직 free전 그렇다면
double free를 봅시다

tcachebin

0x20	-
0x30	heap_base+0x10
...	...

heap_base

0xA0	-	-
0x90	-	-
0x80	mchunk_prev_size	0x30
0x70	-	-
0x60	mchunk_prev_size	0x20
0x50	-	-
0x40	-	-
0x30	mchunk_prev_size	0x30
0x20	FREED CHUNK Next = 0x0	
0x10		
0x00	mchunk_prev_size	0x31

Tcachebin

아직 free전 그렇다면
double free를 봅시다

tcachebin

0x20	-
0x30	heap_base+0x10
...	...

heap_base

0xA0	-	-
0x90	-	-
0x80	mchunk_prev_size	0x30
0x70	-	-
0x60	mchunk_prev_size	0x20
0x50	-	-
0x40	-	-
0x30	mchunk_prev_size	0x30
0x20	FREED CHUNK Next = heap_base+0x10	
0x10		
0x00	mchunk_prev_size	0x31

Tcachebin

여기서 malloc을 하면
free면서 allocated인 상태

결국 UAF처럼 임의 청크를
할당할 수 있다!

tcachebin

0x20	-
0x30	heap_base+0x10
...	...

heap_base

0xA0	-	-
0x90	-	-
0x80	mchunk_prev_size	0x30
0x70	-	-
0x60	mchunk_prev_size	0x20
0x50	-	-
0x40	-	-
0x30	mchunk_prev_size	0x30
0x20	FREED CHUNK Allocated Next = heap_base+0x10	
0x10		
0x00	mchunk_prev_size	0x31

Tcache poisoning

concept

"힙 조작" 한다는 것은...

AAW / AAR 이 결국 목표

UAF / Double Free 가 전제됨

Tcache poisoning

concept

UAF, Double Free등으로 next ptr 조작

glibc 2.27은 tcache가 처음으로 도입된 버전
관련 보안 완화 조치가 없다.
앞서 배운 UAF, Double free에 취약

glibc 2.35+ 에서 tcache 또한 추후에 다룰 예정

임준서

실습 [heap helper v2.27]

free hook이 아직 삭제되기 전입니다 :)

2025.11.05

Q&A

질문이 있다면 하십시오

임준서

2.5cm-3.5cm 떨어진 제목 36px

제목 하단의 부제목 18px

3.5cm 떨어진 내용 1 32px

Git init

Git status

Git add text.txt

Git add .

Git commit

Ctrl+C

Git commit -m "genesis"

Git log

Git log --oneline

Git add .

Git reset .

Git commit -m "add README"

Git log --oneline -n 3

Git commit -a -m "hello"

1cm-1cm 떨어진 주석 12px

1cm-1cm 떨어진 주석 12px

