Lec8: Heap Exploitation

Chengyu Song

Slides based on CS6265 taught by Prof. Taesoo Kim
Scoreboard
Administrative

• Last lab!
• Due: Lab08 is out and its due on Mar 16 at midnight
• 6 basic challenges + 4 bonus
Grading

- In the last lecture (Mar 18), we will let you know your grade
- If that's not the grade that you wanted, you have one more weeks for additional work (let's discuss in person)
- **Write-up matters!**
Lab07: Format String

- fmtstr-read, fmtstr-write, fmtstr-digging, fmt-fortity, fmtstr-heap: 8-9
- sprintf, profile, source-code, mini-sudo, fmtstr-relro: 6-7
Discussion: Lab07

- What's the most "interesting" bug or challenge?
- `printf(buf)` vs `printf("%s", buf)`
- `FORTIFY_SOURCE`, enabled by default from O1
  - Helps but not perfect
Discussion: Basic Challenges

- How do you find the offset / index of parameters
- How do you craft the format string, manually or with a function
Discussion: sprintf

- No '%n' but can still overflow
- Leak canary
- 0 byte in canary → two round overflow
Discussion: profile

- `edit_all`: copying the object on stack to the global one
  - One field is left uninitialized!
  - Spray the stack to control it
  - Ref: [UBI](#)
Discussion: source-code

• Like `sprintf`, but without a loop, how to do multiple stage attacks?
  • Return to main!
Discussion: mini-sudo

- Payload not on stack, how to inject an address?
- Can you reuse existing address on stack?
Lab08: Heap Exploitation

- various malloc implementation (e.g., dlmalloc, ptmalloc)
- use-after-free
- type confusion (not just heap, but more popular in heap)
- double-free techniques
Dynamic memory allocation

- `malloc/free`, `new/delete`
  - Same interface, lots of implementations

Challenges:
- Fragmentation
- Performance
- Security?
dlmalloc

• Doug Lea allocator
  • intro, source

• Used as the default heap allocator in Glibc before ptdmalloc

• Goals: compatibility, portability, space, time, locality, error detection
dlmalloc

- Binning: group allocation by size
  - 16, 24, 32, ... $2^n$
- Locality: best-fit; then next-fit
dlmalloc

- `malloc_chunk` (header)

```c
#define INTERNAL_SIZE_T size_t

struct malloc_chunk {
    INTERNAL_SIZE_T prev_size;
    INTERNAL_SIZE_T size;
    struct malloc_chunk * fd;
    struct malloc_chunk * bk;
};
```
dlmalloc

```c
#define Void_t void
#define SIZE_SZ sizeof(INTERNAL_SIZE_T)
typedef struct malloc_chunk * mchunkptr;

#define chunk2mem( p ) \n  ( (Void_t *)((char *)(p) + 2*SIZE_SZ) )
#define mem2chunk( mem ) \n  ( (mchunkptr)((char *)(mem) - 2*SIZE_SZ) )
```
dlmalloc

chunk -> | prev_size | |
+-----------------------------+-
| size | |
+-----------------------------+-

mem -> | data | |
: ... |
+-----------------------------+-

nextchunk -> | prev_size (if freed) | |
: |
+-----------------------------+-
dlmalloc

+----------------------------------+
| prev_size                        |
| size                             |
+----------------------------------+

| fd                               |
+----------------------------------+
| bk                               |
| (old memory, can be zero bytes)  |

+----------------------------------+
| prev_size ...                    |
+----------------------------------+

| nextchunk                        |
| prev_size ...                    |
+----------------------------------+
#define PREV_INUSE 0x1
#define IS_MMAPPED 0x2
#define SIZE_BITS ( PREV_INUSE | IS_MMAPPED )

#define chunksize( p ) \  
( (p)&#8594;size & ~(SIZE_BITS) )

#define inuse( p ) \  
((mchunkptr)((char*)(p)+((p)->size&~PREV_INUSE)))->size&PREV_INUSE
dlmalloc

- `free`

```c
#define insert_chunk(B, P) {
    mchunkptr F = B->fd; 
    B->fd = P; 
    F->bk = P; 
    P->fd = F; 
    P->bk = B; 
}
```
dlmalloc

- **malloc**

```c
#define unlink(P) {
    mchunkptr BK = P->bk; 
    mchunkptr FD = P->fd; 
    FD->bk = BK; 
    BK->fd = FD; 
    BK->fd = FD; 
}
```
Heap overflow

- What if the metadata of freed chunk is corrupted

```c
#define unlink(P) {
    mchunkptr BK = P->bk; \
    mchunkptr FD = P->fd; \
    FD->bk = BK; \ 
    BK->fd = FD; \ 
}
```
Use-after-free

- Dangling pointers
- Access through dangling pointers
  - Ref: [DangNull](#)
Type Confusion

- Type -> "visibility"
- Type error -> out-of-bound access
  - Ref: Caver
Double-free

• What if try to free already freed memory?
Tutorial

- In-class tutorial:
  - Your first heap exploitation
  - Exploring heap memory structure in GDB