malloc 实现 & 访存 bug
Malloc Advanced & Memory Reference Bugs

课 程 名 ：计算机系统
主 讲 人 ：孟文龙

本课内容

- 动态内存分配
  - 基本概念
  - 隐式空闲链表
  - 显式空闲链表
  - 分离的空闲链表
- **内存相关的风险和陷阱**
  - **对坏指针的解引用**
  - **读未初始化的内存**
  - **内存覆盖错误**
  - **访问不存在的变量**
  - **对一个块反复释放多次**
  - **访问已释放的块**
  - **未能释放块**

C 操作符

操作符	结合性
() [] -> .	left to right
! ~ ++ -- + - * & (type) sizeof	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	right to left
= += -= *= /= %= &= ^= != <<= >>=	right to left
,	left to right

- ->, (), [] 优先级最高，* 和 & 仅次之
- 一元操作符 +, –, * 较其各自的二元形式优先级更高

Source: *K&R* p.53

对坏指针的解引用

- 典型的 `scanf` bug

```
int val;

...

scanf("%d", val);
```

读未初始化的内存

- 错误地假设堆区的数据被初始化为 0

```
/* return y = Ax */
int *matvec(int **A, int *x)
{
    int *y = malloc(N * sizeof(int));
    int i, j;

    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            y[i] += A[i][j] * x[j];
    return y;
}
```

malloc分配的内存应视为未初始化！

- 堆区虽然是请求二进制零的，但空闲块中含有头（脚）部、指针
- 因此应视为未初始化

内存覆盖错误

- 分配（可能）错误的大小
  - 如欲构造一 N x M 的矩阵：

```
int **p;

p = malloc(N * sizeof(int));

for (i = 0; i < N; i++) {
    p[i] = malloc(M * sizeof(int));
}
```

```
int **p;

p = malloc(N * sizeof(int*));

for (i = 0; i < N; i++) {
    p[i] = malloc(M * sizeof(int));
}
```

内存覆盖错误

- 错位off-by-one错误

```
int **p;

p = malloc(N * sizeof(int *));

for (i = 0; i <= N; i++) {
    p[i] = malloc(M * sizeof(int));
}
```

内存覆盖错误

- 未检测字符串是否越界

```
char s[8];
int i;

gets(s);  // reads "123456789" from stdin
```

- 典型的缓冲区溢出攻击即是基于此漏洞

访问不存在的变量

- 忘记局部变量在函数返回之后就不复存在

```
int *foo()
{
    int val;

    return &val;
}
```

对一个块反复释放多次

```
x = malloc(N * sizeof(int));
        <对 x 进行操作>
free(x);

y = malloc(M * sizeof(int));
        <对 y 进行操作>
free(x);
```

访问已释放的块

- **Evil!**

```
x = malloc(N * sizeof(int));
        <对 x 进行操作>
free(x);
    ...
y = malloc(M * sizeof(int));
for (i = 0; i < M; i++)
   y[i] = x[i]++;
```

未能释放已使用完毕的块（内存泄漏）

- 造成缓慢而长期的破坏！

```
foo()
{
    int *x = malloc(N * sizeof(int));
    ...
    return;
}
```

未能释放已使用完毕的块（内存泄漏，续）

- 仅释放了数据结构的一部分

```
struct list {
    int val;
    struct list *next;
};

foo()
{
    struct list *head = malloc(sizeof(struct list));
    head->val = 0;
    head->next = NULL;
    <创建、处理链表的剩余部分>
      ...
    free(head);
    return;
}
```