Attacks

Direct **Exploits**

- Heap Overflow Attacks — **Overflow Vulnerabilities**
- Dangling Pointer Attacks — **Reuse Vulnerabilities**

Exploits **Support**

- Heap Spraying Attacks

Attacker Controlled Location

- Information Leakage
- Unauthorized Access
- Service Interruption

Heap Overflow Attacks
Attack Model & Specific Attacks

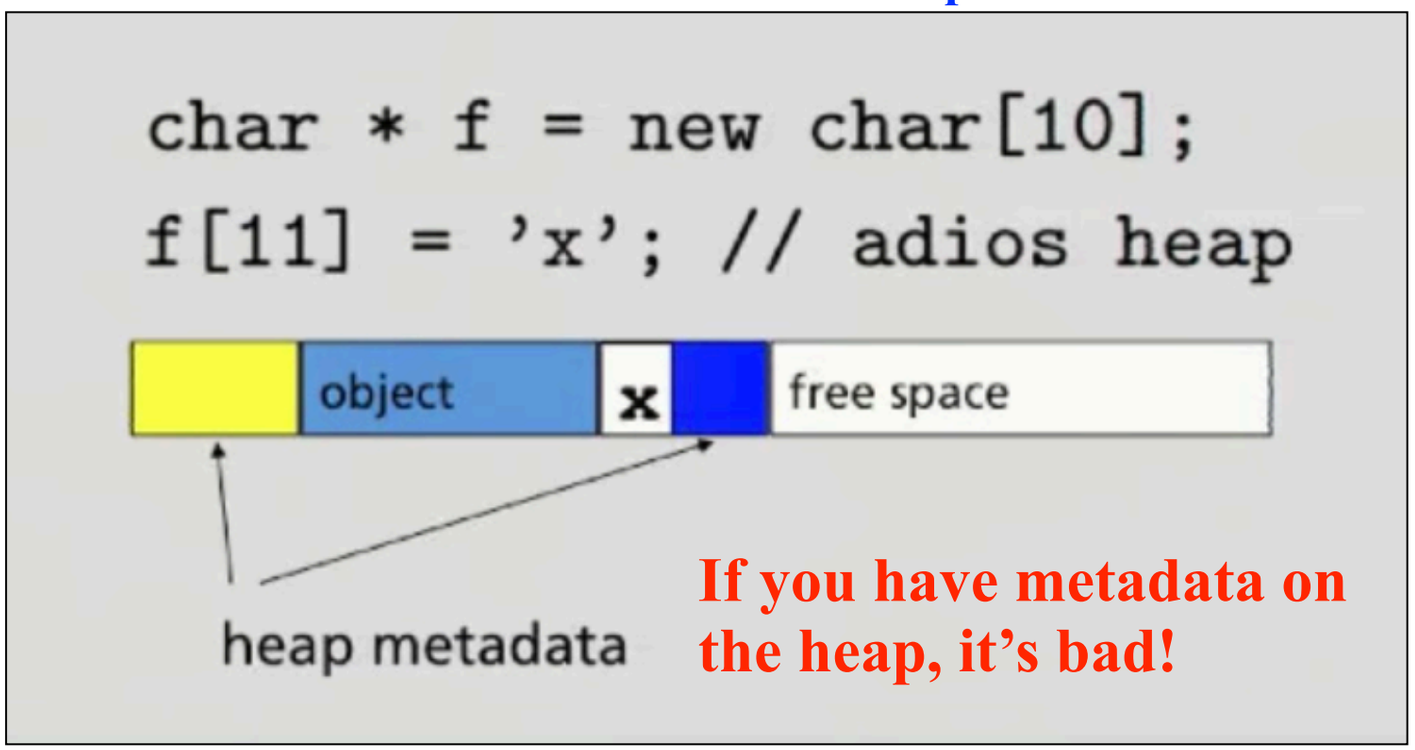
Attack Model

- Overflow attack memory regions
  - source chunk
  - target chunk(s) **application data/heap metadata (eg. free list pointers)**
- Attack succeeds whenever a target chunk is overwritten

Specific Attacks

- Attack application data
  - filename buffers
  - function pointer
- Attack freelist metadata
  - freelist pointer
- Other metadata

A Classical Overflow Example



Ref: Dieharder Presentation

Heap Overflow Attacks
Allocator Analysis

Inline Metadata
- Meta data in each allocated object
- Even vulnerable to small overflow

Page-resident Metadata
- No inline metadata
- Metadata adjacent to heap objects
- Lack of guard pages

Guard Pages

- Before each page with metadata
- Provide gaps in memory
- Against contiguous overrun

Canaries

- Overflow corrupts canary first
- Failed verification indicates overflow
- Runtime efficiency tradeoff

Randomized Placement

- Limited randomization
- Low entropy, not fully randomized

Heap Spraying Attacks
Attack Model & Allocator Analysis

Attack Model

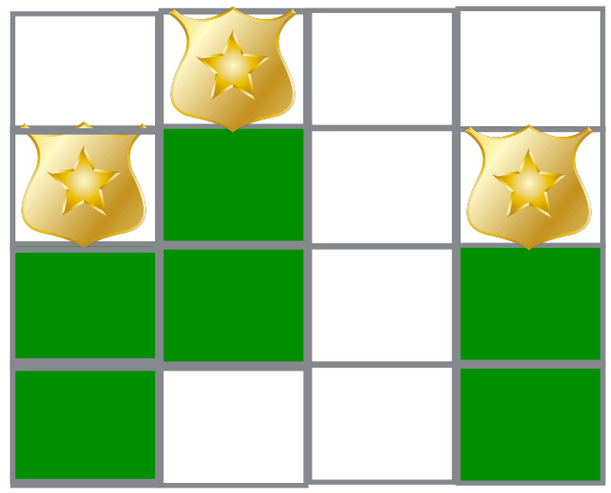
- Does not exploit by itself
- Predictable start location of heap allocations
- No a priori knowledge
- Known address attacks

} **Address of a valid heap object** → **Arbitrary Code Execution**

Allocator Analysis

- **No a priori knowledge**
  - Guess the address
  - Make |V| close to |H|

|V| - size of target objects
|H| - size of the heap



- **Known address attacks**
  - From known address to target address, **guessable?**
  - Contiguous allocated - **guessable**
  - Random allocated - **minimum correlation**

Dangling Pointer Attacks
Attack Model

Use of Free Chunk

- Write - dangling point error
- Subsequent Free - double-free error

Combat Strategies

- Delay reuse, eg. FIFO
- Randomized reuse

Reuse Vulnerabilities

- Overwrite the function pointer in a dangled object
- Reuse the pointer jumps to attacker-controlled location

Dangling Pointer Example

DieHard: Probabilistic Memory Safety for C/C++ Programs [PLDI 2005]

```
Foo * f = new Foo ("happy");
Foo * x = f;
delete f;
Foo * g = new Foo ("sad");

// dang, dangling pointer
cout << x->info << endl;
```

Dangling Pointer Attacks
Allocator Analysis

Freelist

- LIFO, perfect predictability of reuse
- Inline metadata, can be forced to write attacker-controlled data

BiBOP

- Different reuse policies
- No inline metadata

Coalescing

- Unpredictable of reuse
- Defragmented heap, difficult to coalesce