CSE467: Computer Security

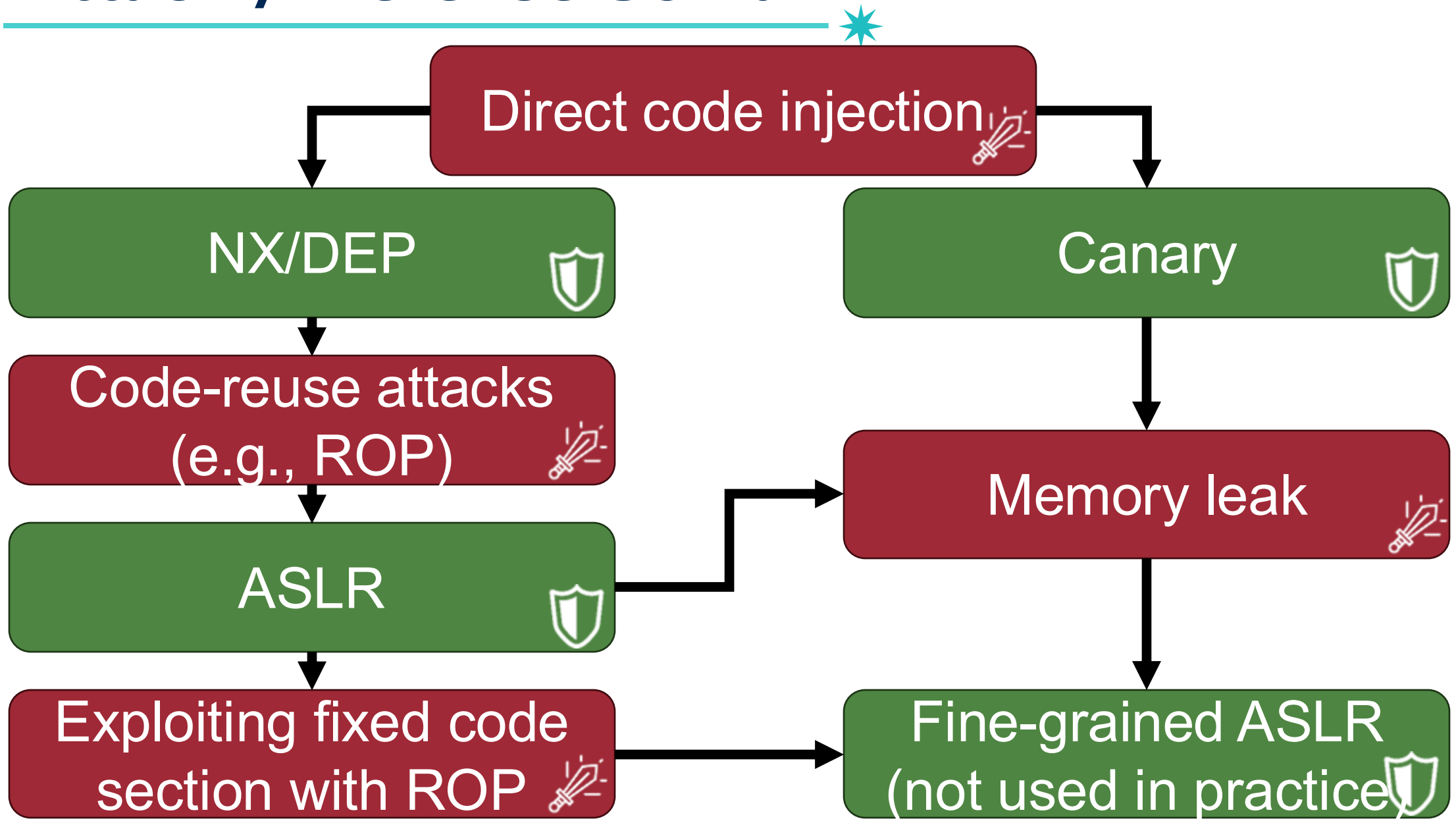
18. Type Confusion & Use After Free

Seongil Wi

Department of Computer Science and Engineering

The slide is based on Prof. Sang Kil Cha's lecture slide

Attack / Defense So Far



Memory Corruption So Far

- Buffer overflows
- Format string bugs

What is another major ***attack vector*** to corrupt memory?

Type Confusion 

Type

A classification of data which tells the compiler or interpreter how the programmer intends to use the data

Type Safety

Types prevent unintended errors

```
>>> 1 + "1"

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

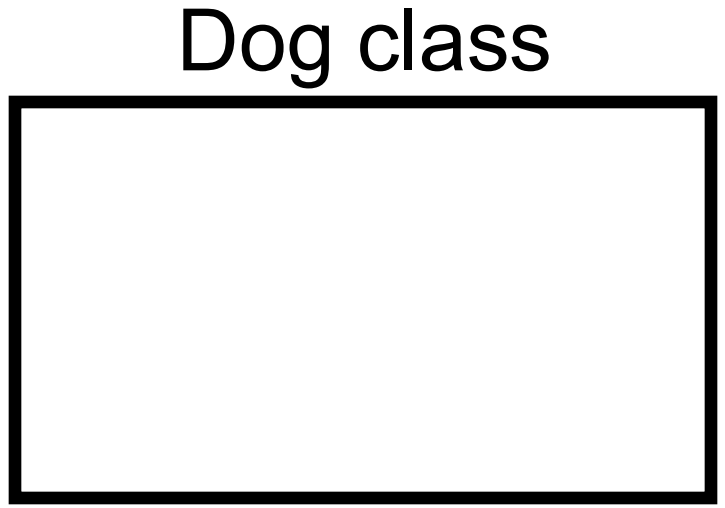
Type Confusion

- Mistaking a memory location for certain type as a memory for different type

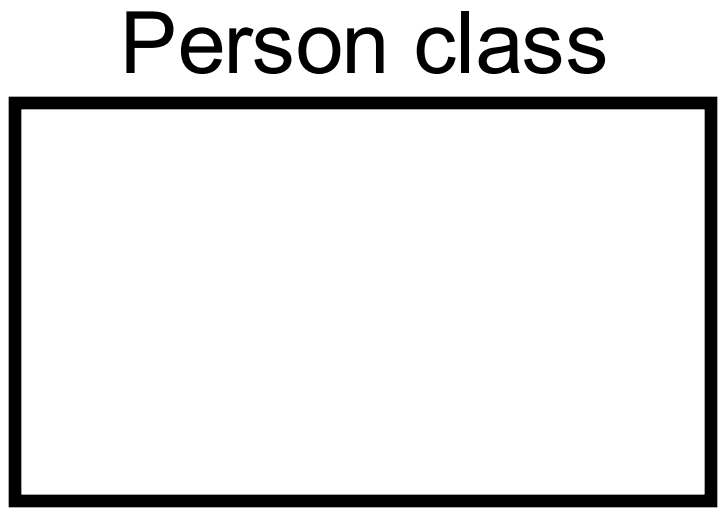
- Type confusion happens when the type-safety is violated

Type Confusion


```
Dog *d = (Dog*) some_ptr;
d->bark();
```

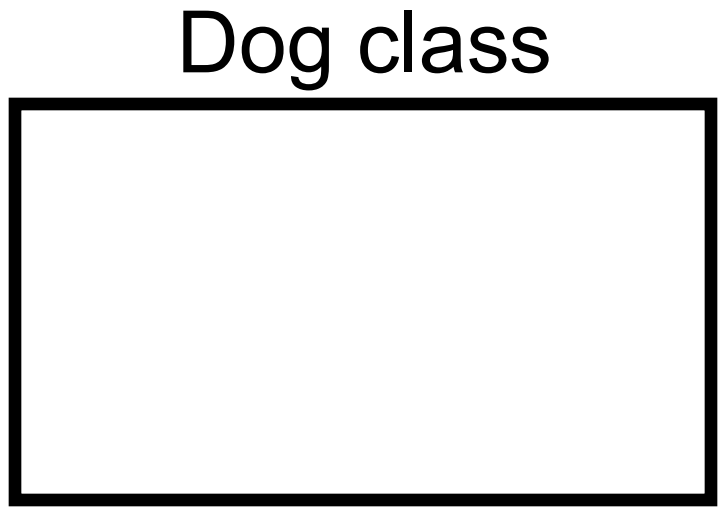



```
Dog *d = (Dog*) some_ptr;
d->bark();  //???
```



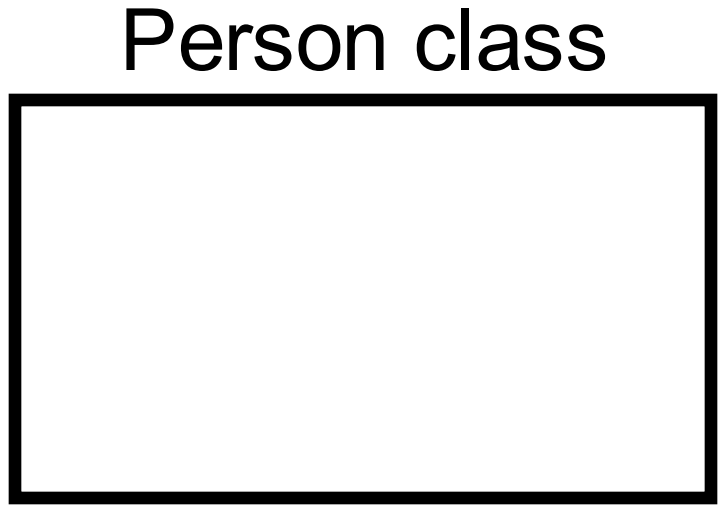
Type Confusion


```
Dog *d = (Dog*) some_ptr;
d->bark();
```

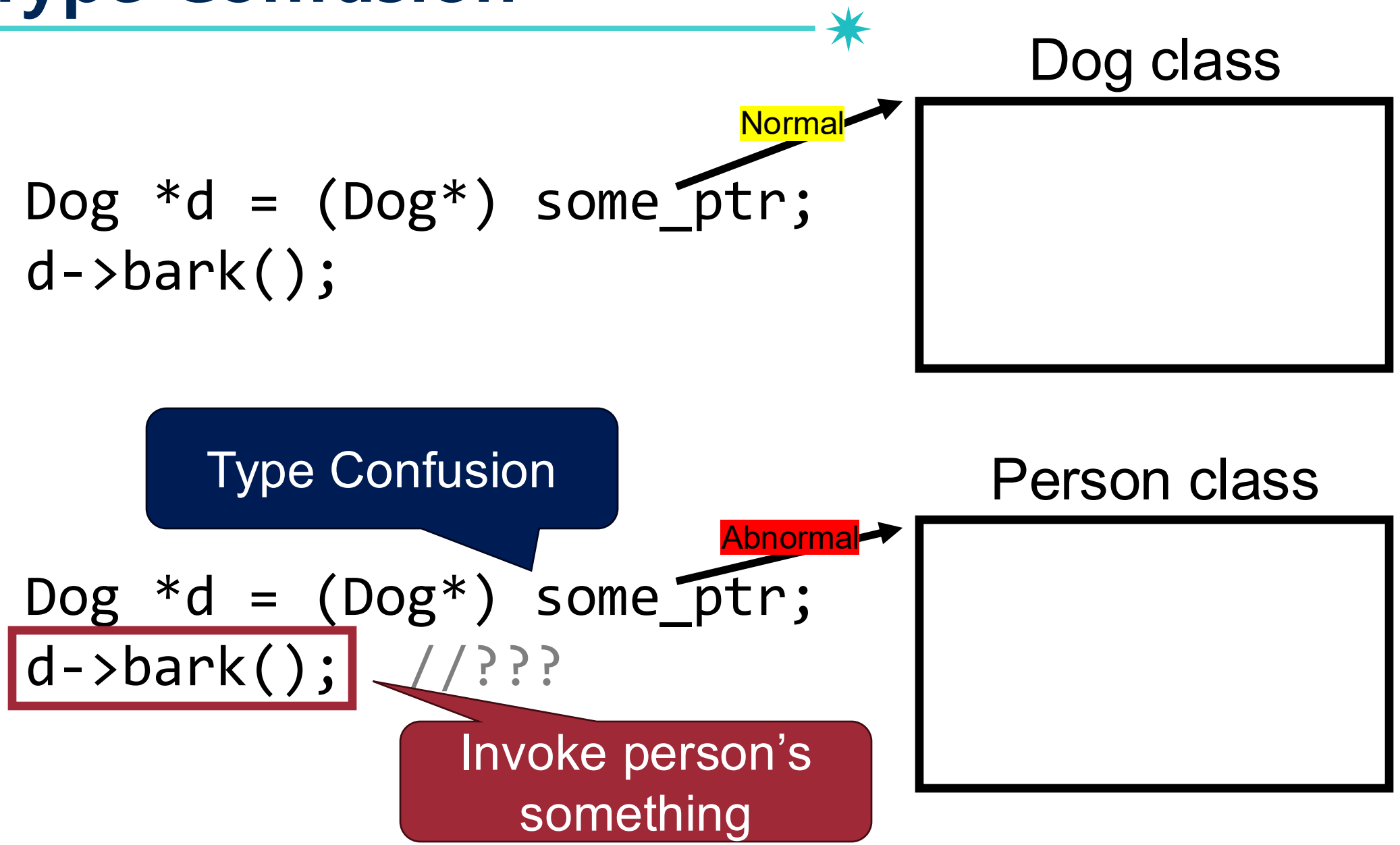


Type Confusion

```
Dog *d = (Dog*) some_ptr;
d->bark();  //???
```

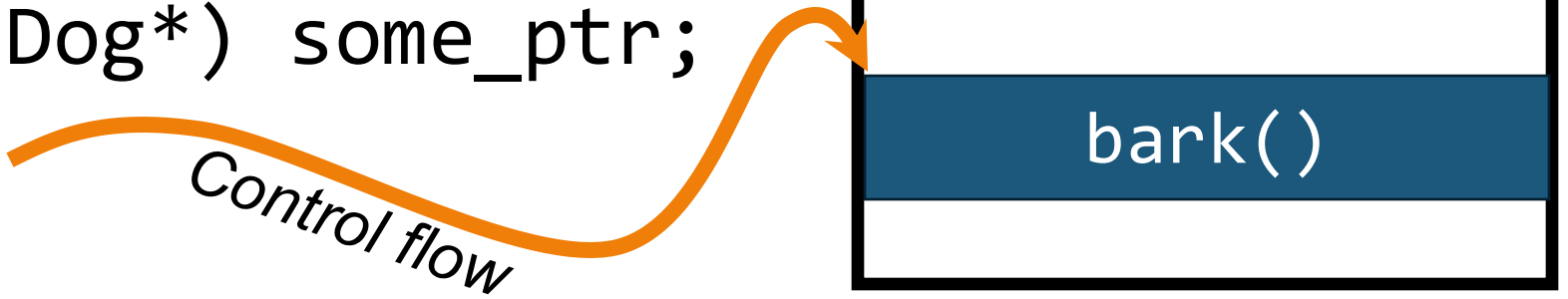


Type Confusion



Type Confusion Attack (Implication)

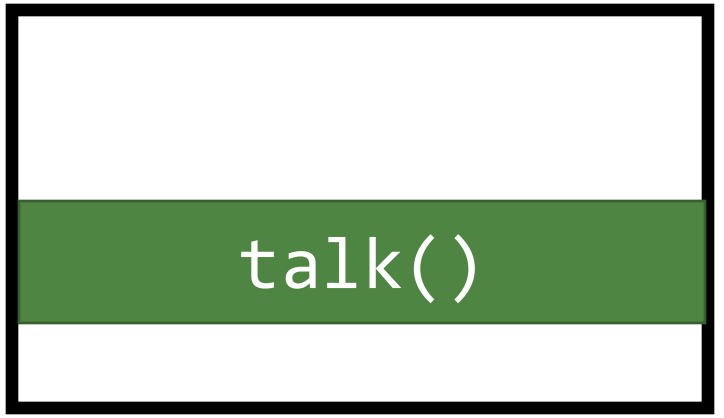

```
Dog *d = (Dog*) some_ptr;
d->bark();
```



Dog class

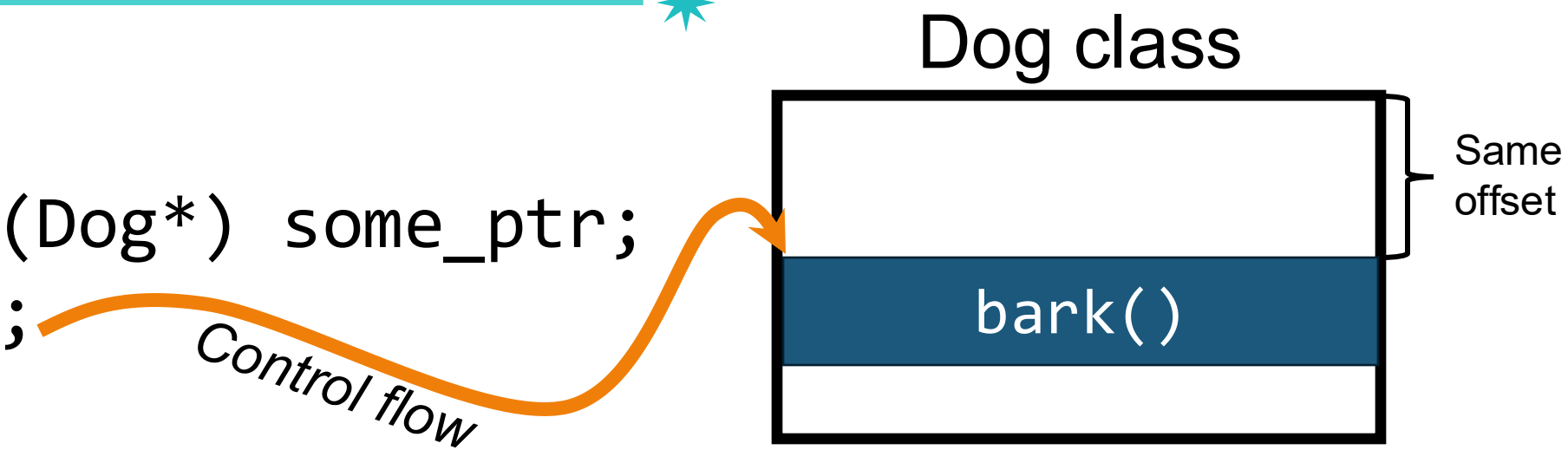

```
Dog *d = (Dog*) some_ptr;
d->bark();  //???
```

Person class

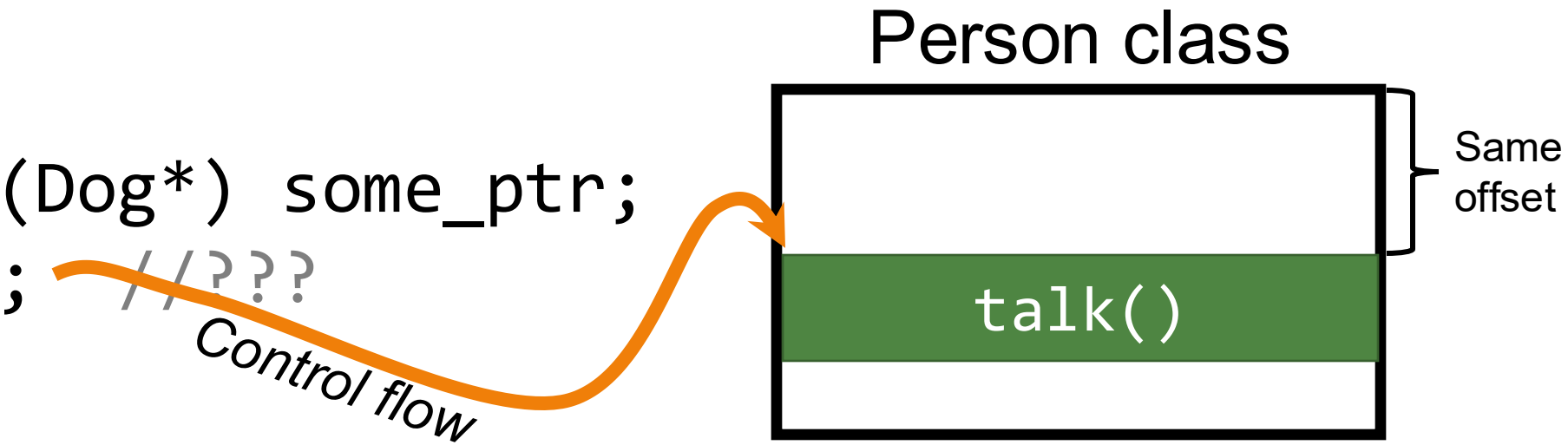


Type Confusion Attack (Implication)


```
Dog *d = (Dog*) some_ptr;
d->bark();
```

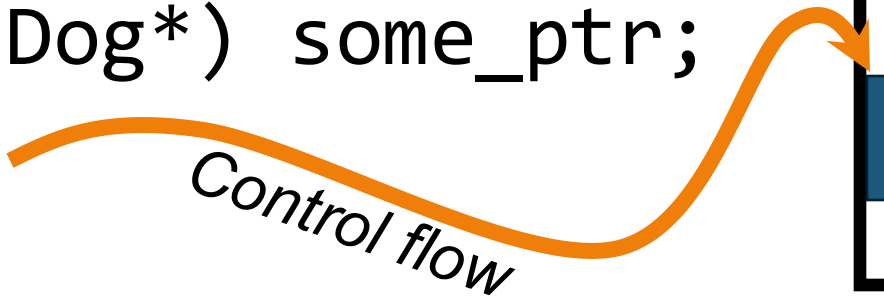


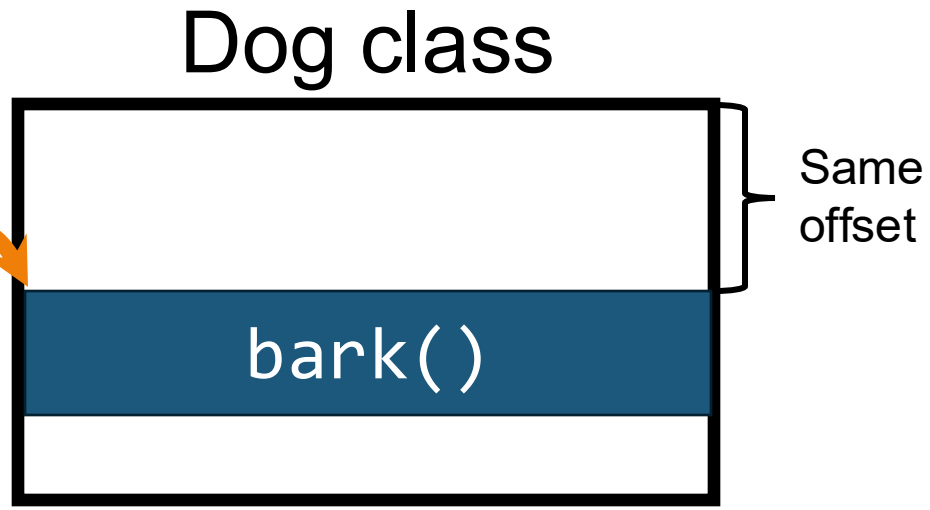

```
Dog *d = (Dog*) some_ptr;
d->bark(); //???
```



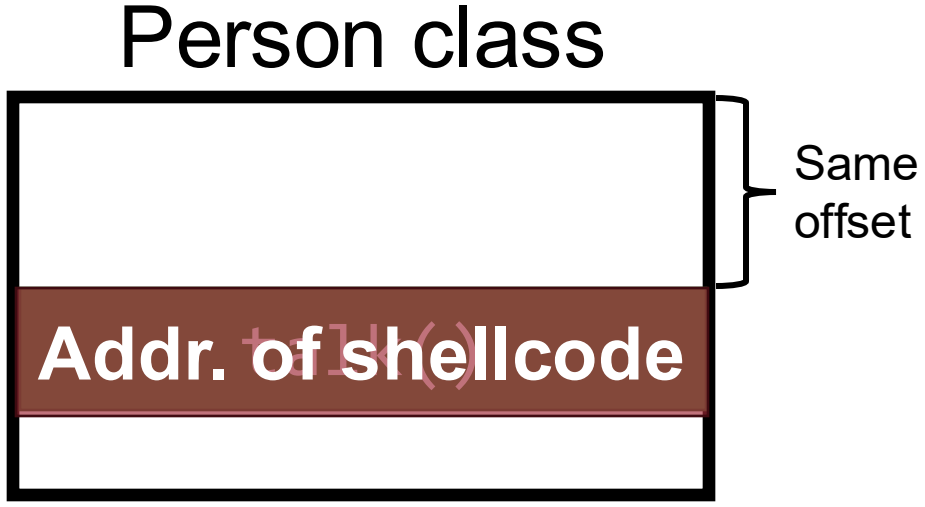
Type Confusion Attack (Implication)


```
Dog *d = (Dog*) some_ptr;
d->bark();
```






```
some_ptr->name="[shellcode]"
…
Dog *d = (Dog*) some_ptr;
d->bark();  //???
```



Type Confusion Attack (Implication)

```
Dog *d = (Dog*) some_ptr;
d->bark();
```

Control flow


```
some_ptr->name="[shellcode]"
…
Dog *d = (Dog*) some_ptr;
d->bark(); //???
```

Control flow

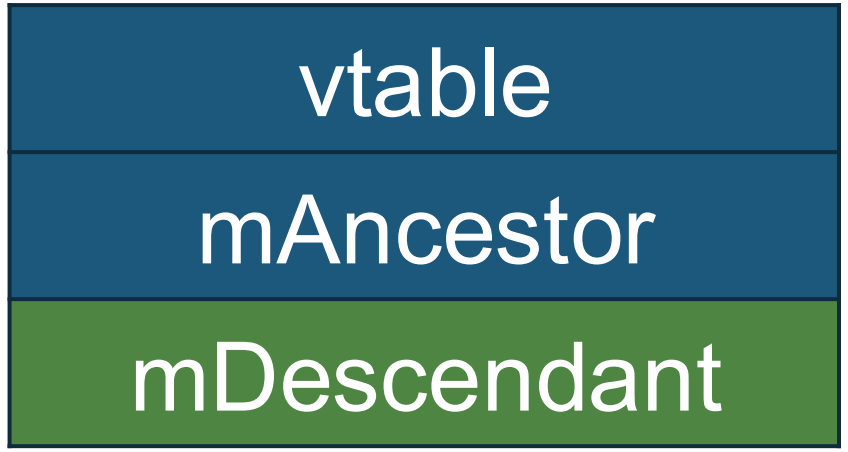
Type Confusion Example: Downcasting


```
class Ancestor {
    public:
        int mAncestor;
    ...
};

class Descendant: public Ancestor {
    public:
        int mDescendant;
    ...
};
```

Inherit Ancestor class

Type Confusion Example: Downcasting


```
class Ancestor {
    public:
        int mAncestor;
    ...
};

class Descendant: public Ancestor {
    public:
        int mDescendant;
    ...
};
```

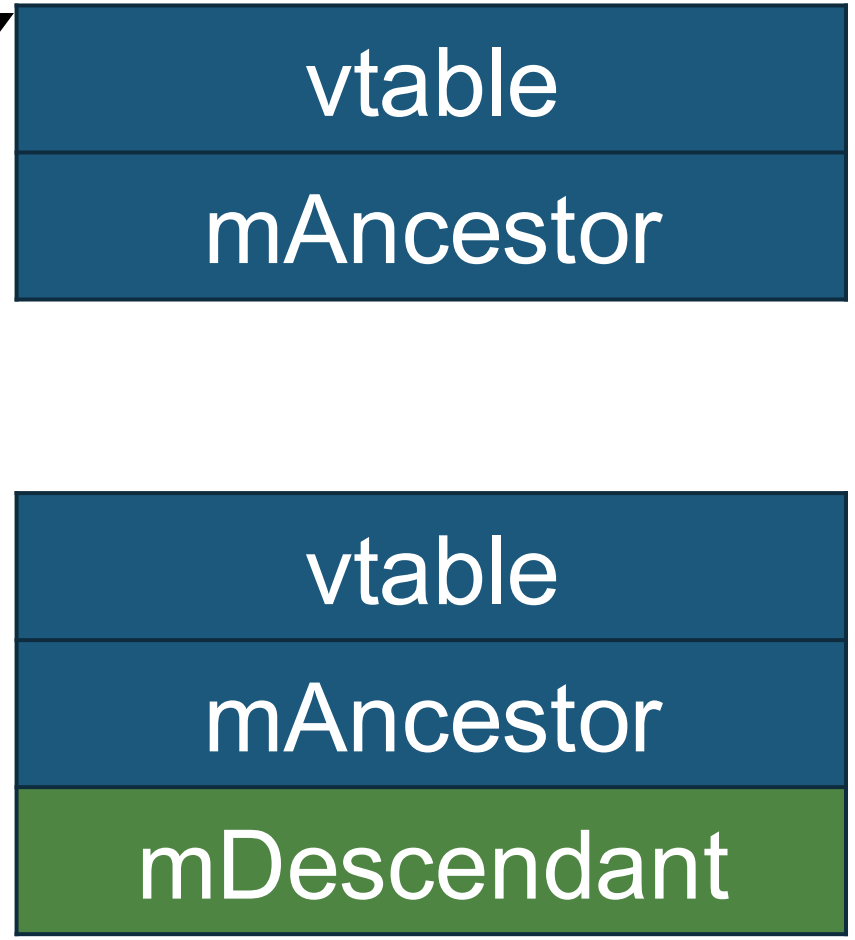


Vulnerable code

```
Ancestor* a = new Ancestor();
Descendant* d = static_cast<Descendant*>(a);
d->mDescendant = 42;
```

Type Confusion Example: Downcasting


```
class Ancestor {
    public:
        int mAncestor;
    ...
};

class Descendant: public Ancestor {
    public:
        int mDescendant;
    ...
};

Ancestor* a = new Ancestor();
Descendant* d = static_cast<Descendant*>(a);
d->mDescendant = 42;
```



Downcasted pointer

Vulnerable code

Type Confusion Example: Downcasting

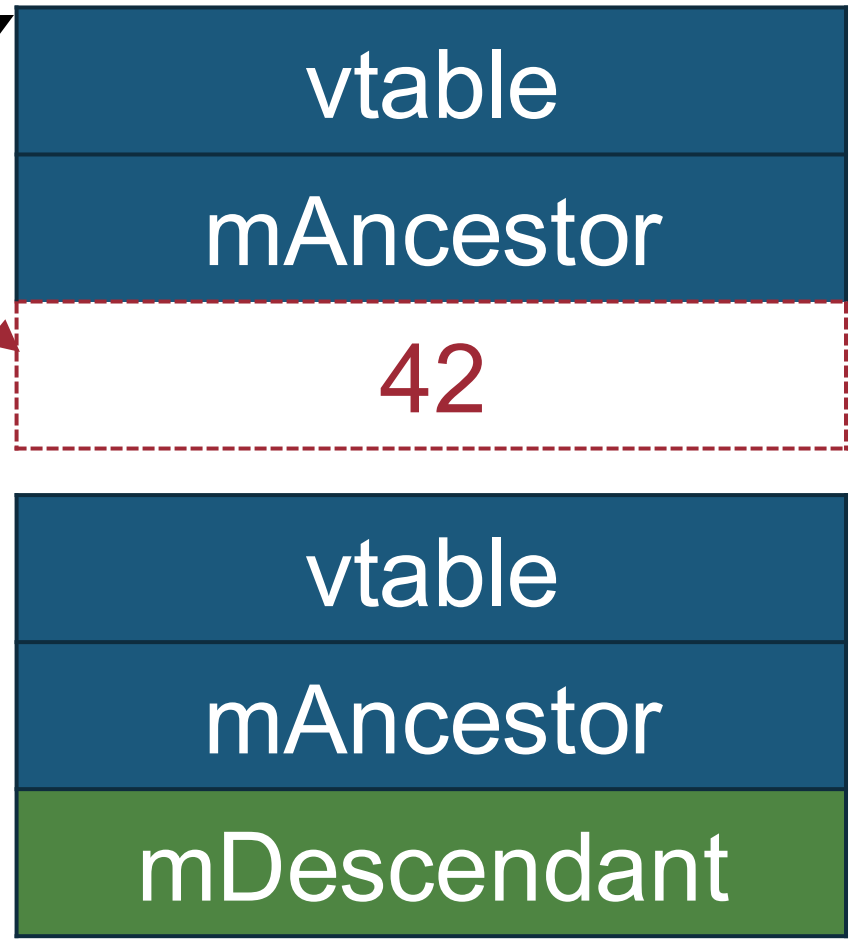

```
class Ancestor {
    public:

};

class Descendant: public Ancestor {
    public:
        Descendant;

};
```

Memory corruption:
It can now access a memory region that was not allocated!

Downcasted pointer

Vulnerable code

```
Ancestor* a = new Ancestor();
Descendant* d = static_cast<Descendant*>(a);
d->mDescendant = 42;
```



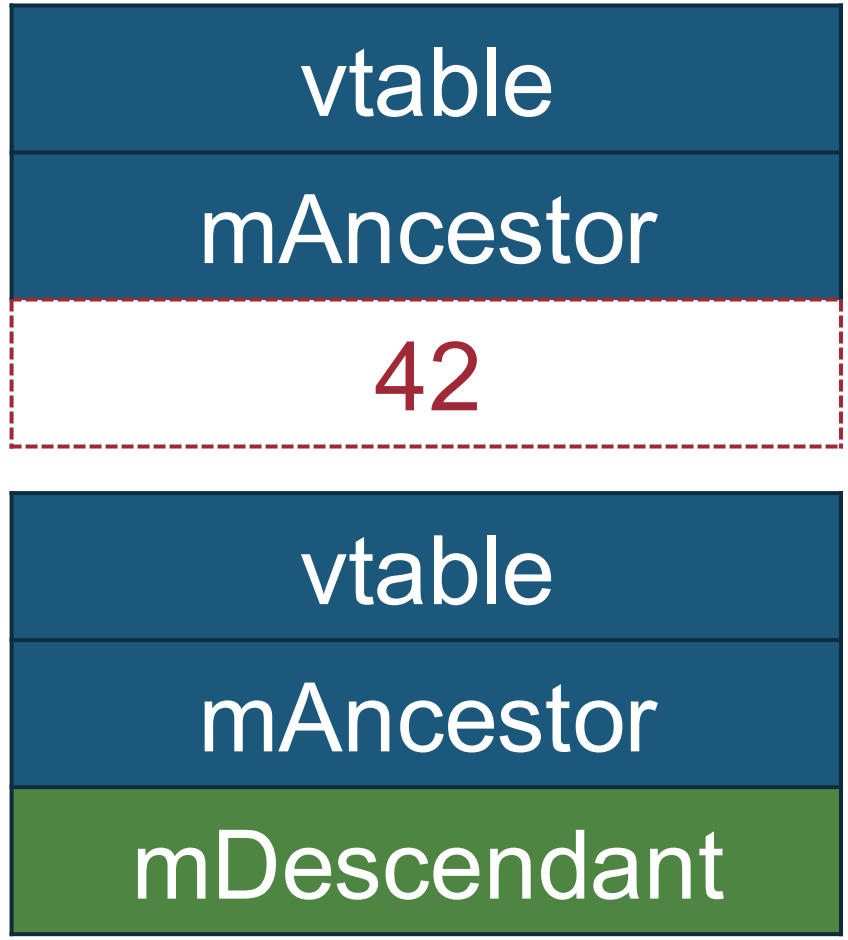
Question: But, Why Get Confused?

Suppose there is a huge gap between these lines
(e.g., separated in two different libraries)

Vulnerable code

```
Ancestor* a = new Ancestor();
Descendant* d = static_cast<Descendant*>(a);
d->mDescendant = 42;
```

Implication of the Downcasting

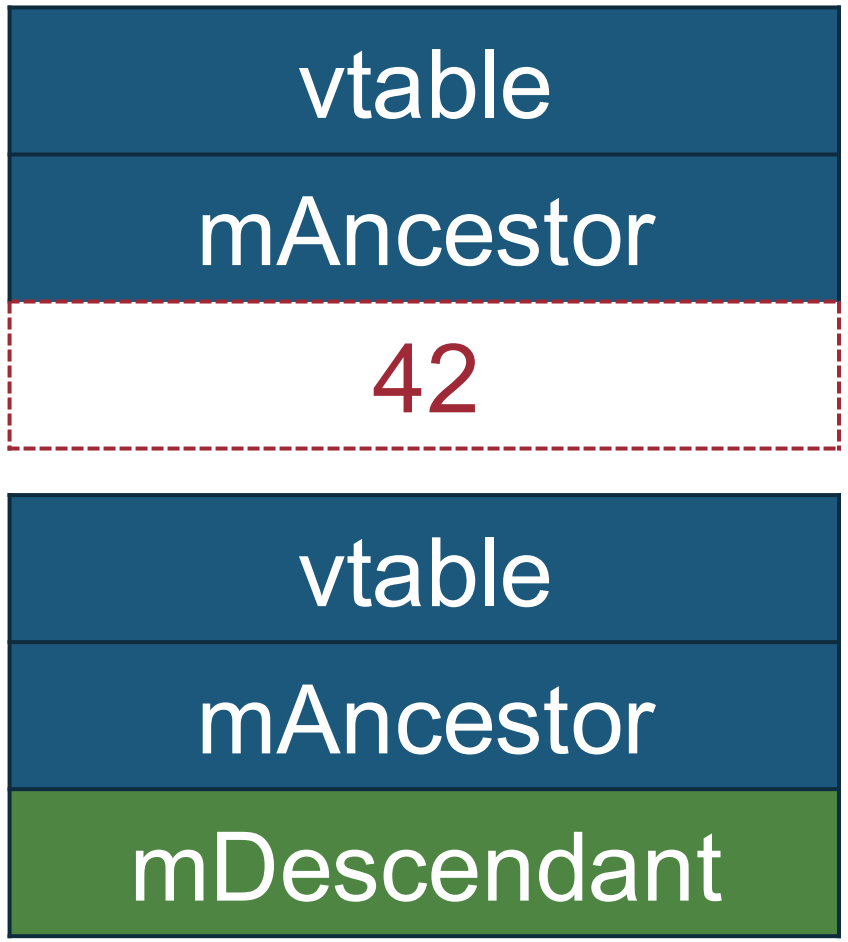


What if a user can write an arbitrary value to the confused pointer?

Vulnerable code

```
Ancestor* a = new Ancestor();
Descendant* d = static_cast<Descendant*>(a);
d->mDescendant = 42;
```

Attacker's Perspective



Unlike other attack vectors, we can ***reliably*** corrupt a certain memory field, *i.e.*, we don't need to know the actual address of `mDescendant`

Vulnerable code

```
Ancestor* a = new Ancestor();
Descendant* d = static_cast<Descendant*>(a);
d->mDescendant = 42;
```

Patch: Use `dynamic_cast`

Limitations:

- Slow
- Compiler options such as `--fno-rtti` can disable it!

Use After Free

(A popular source of type confusion)

Use After Free

- If after freeing a memory location, a program does not clear the pointer to that memory, an attacker can use it to hack the program

Use After Free Example

```
Foo * f = new Foo();
Foo * ptr = f;
ptr->x = 42;
delete f;
f = NULL;
Bar * b = new Bar();
b->y = “hello world”;
cout << ptr->x << endl;
```

Class information

```
class Foo {
  public:
    int x;
};
class Bar {
  public:
    const char* y;
};
```

Use After Free Example


```
Foo * f = new Foo();
Foo * ptr = f;
ptr->x = 42;
delete f;
f = NULL;
Bar * b = new Bar();
b->y = “hello world”;
cout << ptr->x << endl;
```

Allocate a memory block on the heap

Class information

```
class Foo {
  public:
    int x;
};
class Bar {
  public:
    const char* y;
};
```

Use After Free Example

```
Foo * f = new Foo();
Foo * ptr = f;
ptr->x = 42;
delete f;
f = NULL;
Bar * b = new Bar();
b->y = “hello world”;
cout << ptr->x << endl;
```

Class information

```
class Foo {
  public:
    int x;
};
class Bar {
  public:
    const char* y;
};
```

Use After Free Example

```
Foo * f = new Foo();
Foo * ptr = f;
ptr->x = 42;
delete f;
f = NULL;
Bar * b = new Bar();
b->y = “hello world”;
cout << ptr->x << endl;
```

Class information

```
class Foo {
  public:
    int x;
};
class Bar {
  public:
    const char* y;
};
```

Use After Free Example

```
Foo * f = new Foo();
Foo * ptr = f;
ptr->x = 42;
delete f;
f = NULL;
Bar * b = new Bar();
b->y = “hello world”;
cout << ptr->x << endl;
```

Return the block to the free list

Class information

```
class Foo {
  public:
    int x;
};
class Bar {
  public:
    const char* y;
};
```

Use After Free Example

```
Foo * f = new Foo();
Foo * ptr = f;
ptr->x = 42;
delete f;
f = NULL;
Bar * b = new Bar();
b->y = “hello world”;
cout << ptr->x << endl;
```

Class information

```
class Foo {
  public:
    int x;
};
class Bar {
  public:
    const char* y;
};
```

Often called
“***Dangling Pointer***”

Use After Free Example

```
Foo * f = new Foo();
Foo * ptr = f;
ptr->x = 42;
delete f;
f = NULL;
Bar * b = new Bar();
b->y = “hello world”;
cout << ptr->x << endl;
```

Find an appropriate block from the list of free blocks

Class information

```
class Foo {
  public:
    int x;
};
class Bar {
  public:
    const char* y;
};
```

Use After Free Example

```
Foo * f = new Foo();
Foo * ptr = f;
ptr->x = 42;
delete f;
f = NULL;
Bar * b = new Bar();
b->y = “hello world”;
cout << ptr->x << endl;
```

Class information

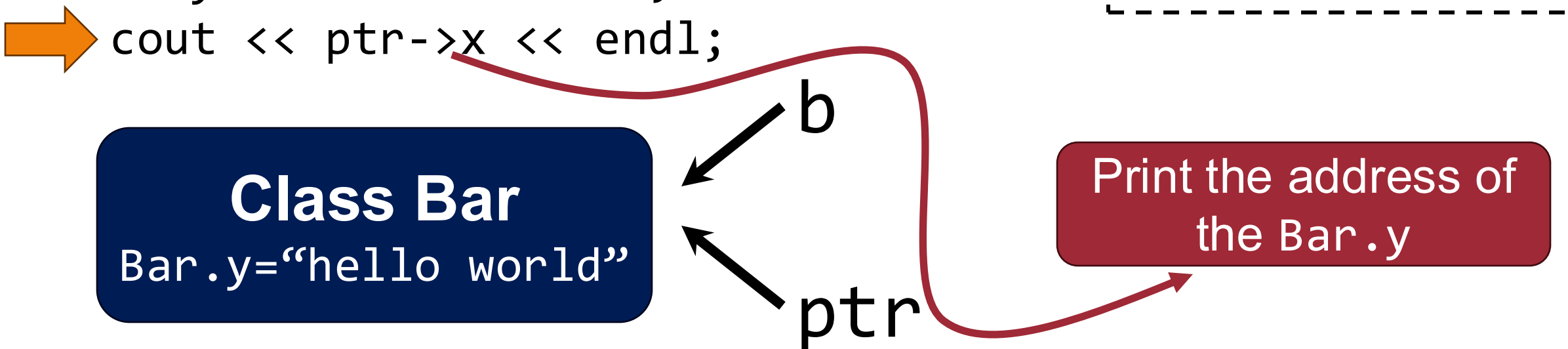
```
class Foo {
  public:
    int x;
};
class Bar {
  public:
    const char* y;
};
```

Use After Free Example

```
Foo * f = new Foo();
Foo * ptr = f;
ptr->x = 42;
delete f;
f = NULL;
Bar * b = new Bar();
b->y = “hello world”;
cout << ptr->x << endl;
```

Class information

```
class Foo {
  public:
    int x;
};
class Bar {
  public:
    const char* y;
};
```



Use After Free Example

```
Foo * f = new Foo();
Foo * ptr = f;
ptr->x = 42;
delete f;
f = NULL;
Bar * b = new Bar();
b->y = “hello world”;
cout << ptr->x << endl;
```

Class information

```
class Foo {
  public:
    int x;
}
```

```
ar* y;
```

We ***used*** this pointer ***after free***

Class Bar
Bar.y=“hello world”

b

ptr

Print the address of the Bar.y

Use After Free can Trigger Type Confusion

- A dangling pointer's type and the corresponding reallocated data's type can be different => Trigger type confusion!

Example: OpenSSL UAF Bug


```
...
dtls1_hm_fragment_free(frag);
pitem_free(item);
if (al==0) {
    *ok = 1;
    return frag->msg_header.frag_len;
}
```

Example: OpenSSL UAF Bug


```
...
dtls1_hm_fragment_free(frag);
pitem_free(item);
if (al==0) {
    *ok = 1;
    return frag->msg_header.frag_len;
}
```

frag is freed

Example: OpenSSL UAF Bug


```
...
dtls1_hm_fragment_free(frag);
pitem_free(item);
if (al==0) {
     *ok = 1;
     return frag->msg_header.frag_len;
}
```

frag is freed

Read after the free

Summary

- Type confusion bugs happen when a program misuses types
- ***Type confusion allows attackers to trigger memory corruption or disclosure***
- Use After Free is one of the major causes of type confusion

Question?