Auditing C++

Dr. Jared DeMott
CTO AND FOUNDER

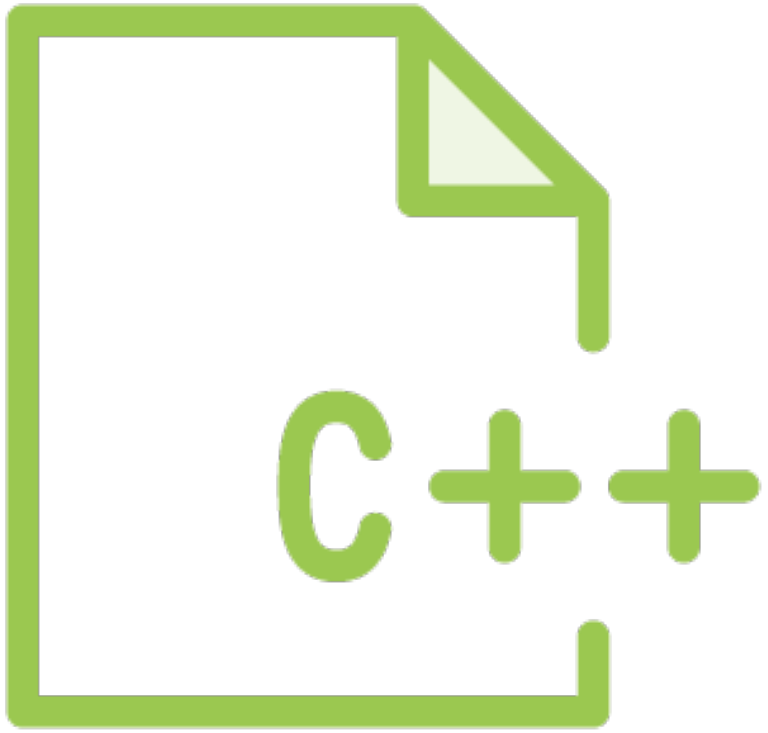
@jareddemott www.vdalabs.com

Overview

C++ specific bugs

How to Audit C++ Classes

Examples

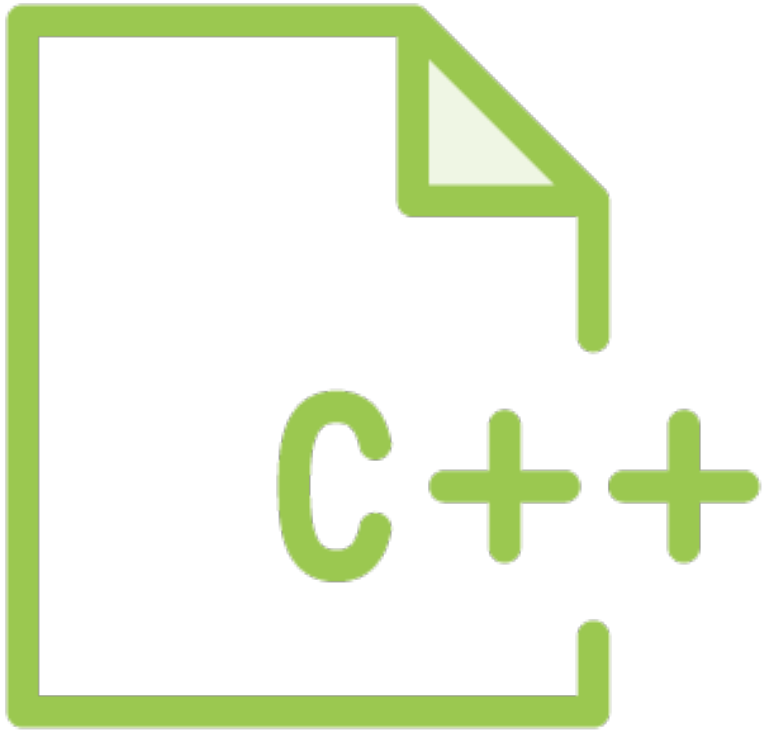


Casting

- Type confusion

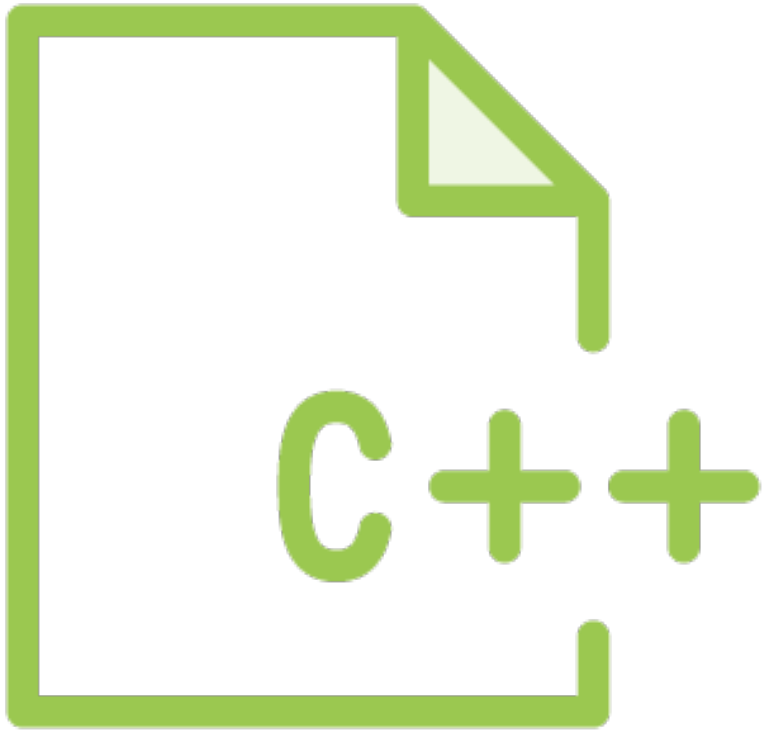
Unexpected paths

- UAF
  - Double frees due to error in destructor, etc.
- Class issues
  - Not properly tracking construct/destroy
  - Memory leak is common



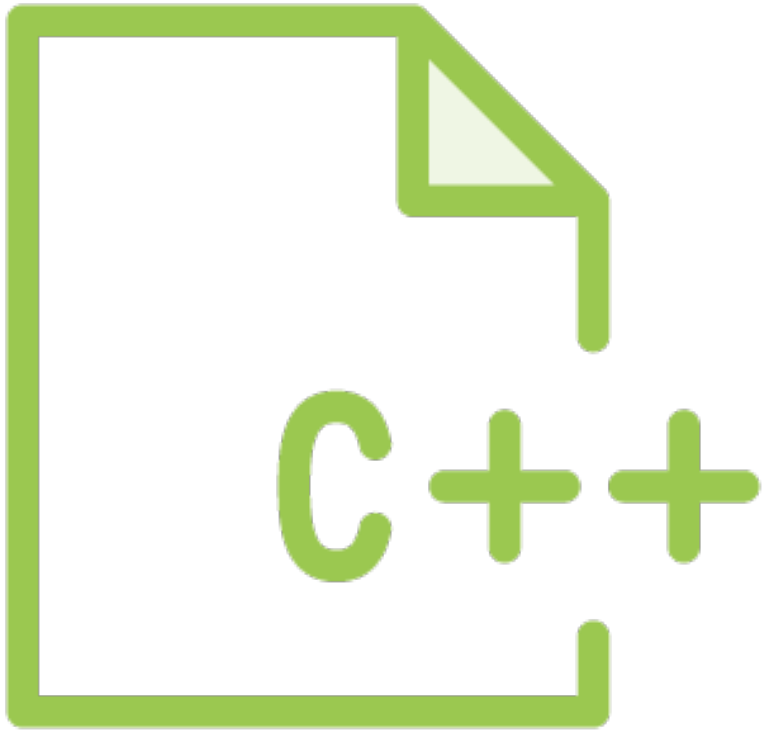
Use latest compiler if possible

- Dynamic binding issues
- STL code static in executable
  - Recompile old code to get latest STL and latest generic protections
- Compiler differences

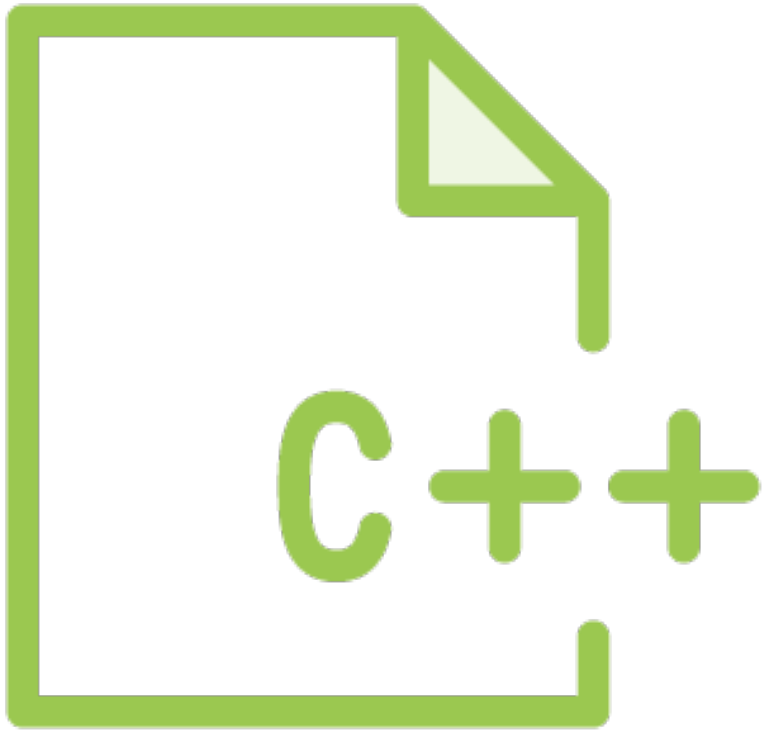


How to audit classes?

- Step 1
  - Enumerate Internal State
    - Note each piece of internal state
    - References to external variables and functions
    - Inherited state from parent classes



- Step 2
  - Determine Responsibility
    - For members that have associated memory
      - Who is responsible for allocation of the memory
      - Who is responsible for deallocation
      - Where does this happen
    - Object lifecycle is often complex

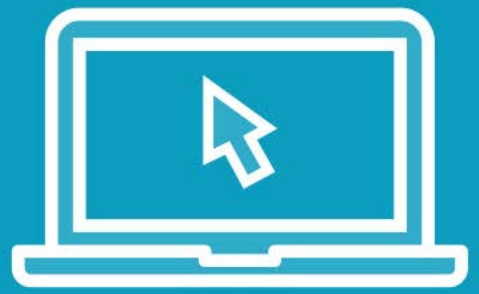


- Step 3
  - Determine Invariants
    - Relationships between member variables
    - Relationships that should hold true throughout lifetime of class

```
class buffer
{
        void * memory_ptr
        size_t buffer_size
        size_t write_ptr
...
```

Name	Type	Responsibilities	Invariants
Memory_ptr	Void *	Init in constructor Free in destructor Reallocated in resize()	Must point to valid memory Must not be NULL
buffer_size	Size_t	Set in constructor Modified in resize()	Must track length of memory at memory_ptr
Write_ptr	Size_t	Set in constructor Modified in write() Modified in resize()	Must be between 0 and buffer_size-1

Demo



Audit a class

Newer bugs

C++

- Browsers
- Office software
- Virtualization
- Complex data parsing/processing
  - Flash, VLC, pdf viewers, etc.

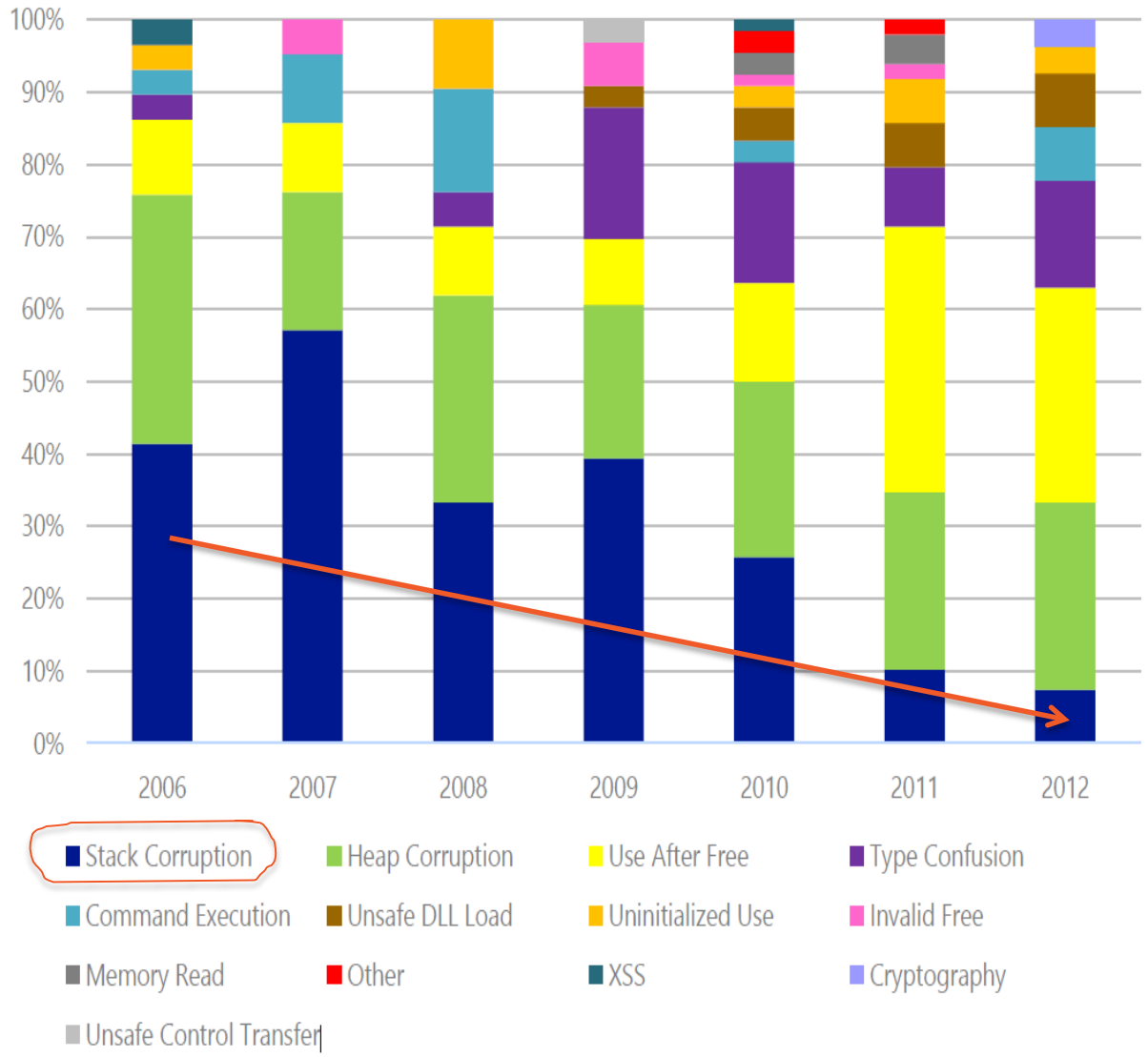
Step 1: Malformed webpage

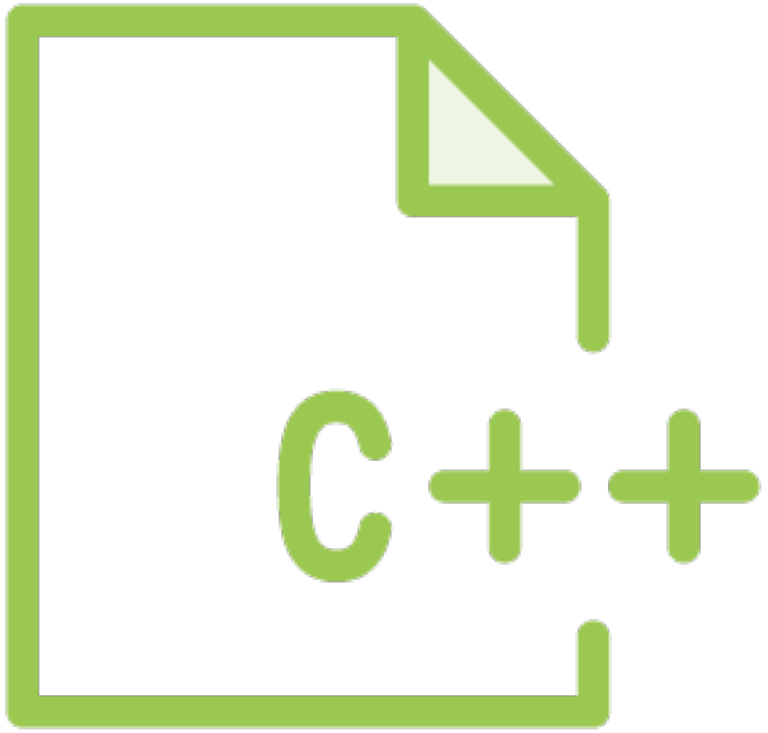
C

- Kernels
- Drivers
- Embedded

Step 2: Kernel or low level exploit to escape sandbox

Current Trends



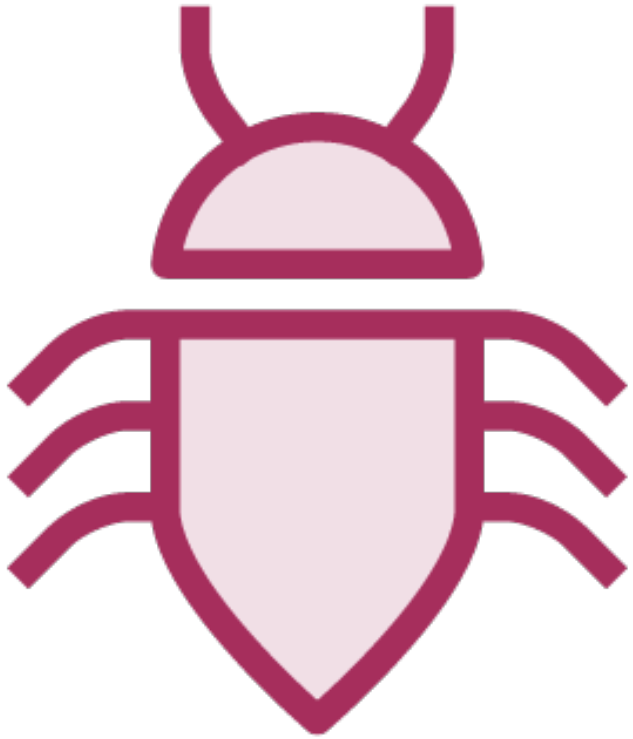


Why are the Bugs changing?

- Awareness
  - Training
- Less general use
  - Don’t use C for cafeteria menu app
- Better Testing
  - Static/Dynamic analysis

But, more complex code

- Simpler bugs might be mostly gone
  - But browsers
    - New bug patterns always seem to surface

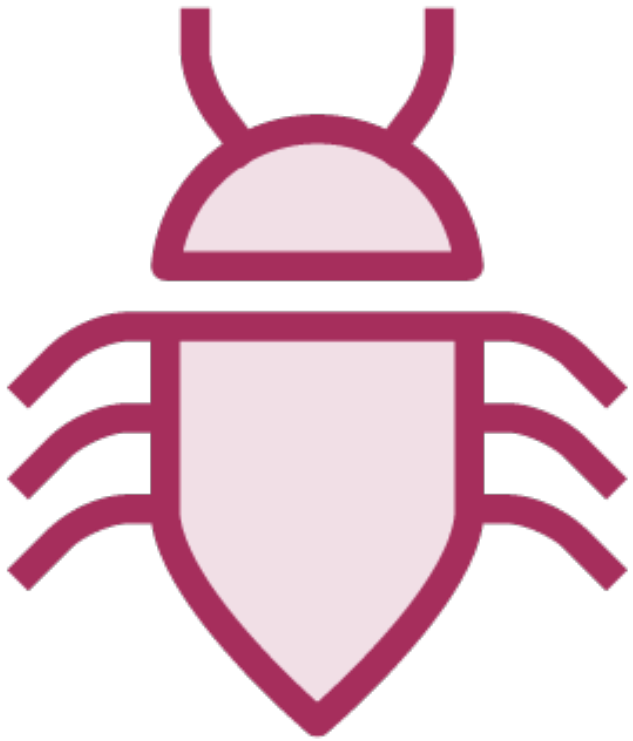


Object lifecycle management

- The dreaded *use-after-free*
- Native pointers
  - Destruction managed by programmer
    - Exploitation requires knowledge of allocations and garbage collection
- Smart pointers
  - Destruction managed by framework
    - But are they consistently used?

Casting complex objects

- *Type confusion*
  - Very weary of *cast* used upon remote data

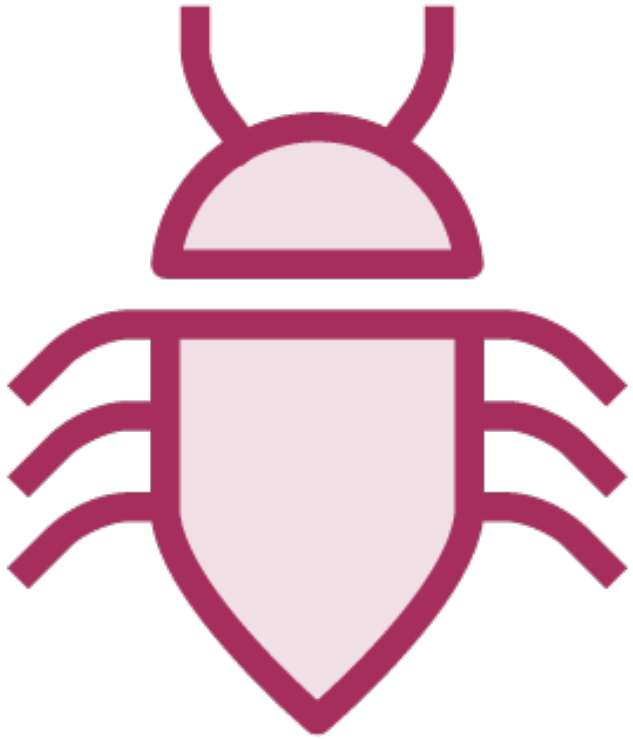


How to Find?

- Fuzzing
  - Specific to domain, like JavaScript language fuzzing
- Manual Code Review
  - Reviews and correlating open source intel
- Automated Code Review tools
- Iterative
    - E.g. find something via fuzzing
      - Understand root cause
      - Look for pattern manually or via static analysis tools

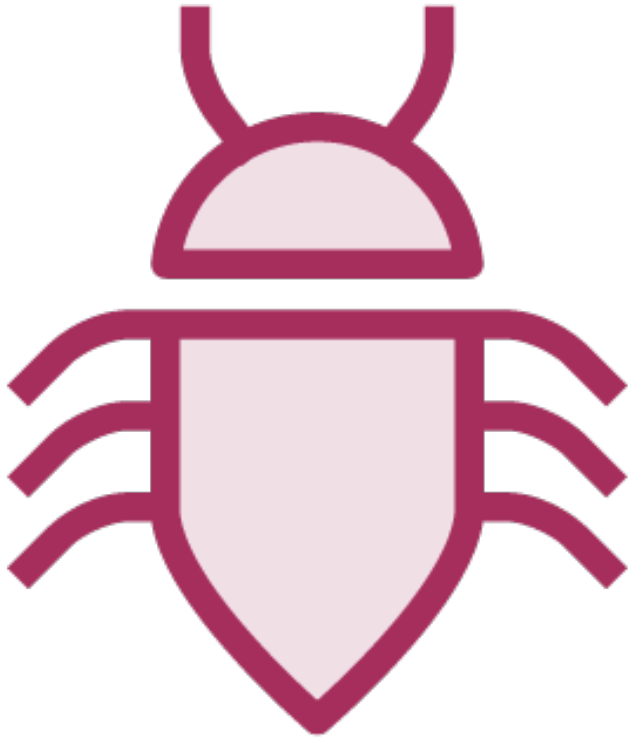
Use-after-free

```
4  class Cat {
5  public:
6          Cat(const std::string& name_ = "Kitty")
7          : name(name_)
8          {
9                  std::cout << "Cat " << name << " created." << std::endl;
10         }
11         ~Cat(){
12                 std::cout << "Cat " << name << " destroyed." << std::endl;
13         }
14         void eatFood(){
15                 std::cout << "Food eaten by cat named " << name << "." << std::endl;
16         }
17 private:
18         std::string name;
19 };
20
21 int main (){
22         Cat *molly = new Cat("cat1");
23         molly->eatFood();
24         delete molly;
25
26                 //.. normally code paths such as after errors etc.
27                 //make the use-after-free non-obvious
28                 molly->eatFood();
29
30         return 0;
31 }
```

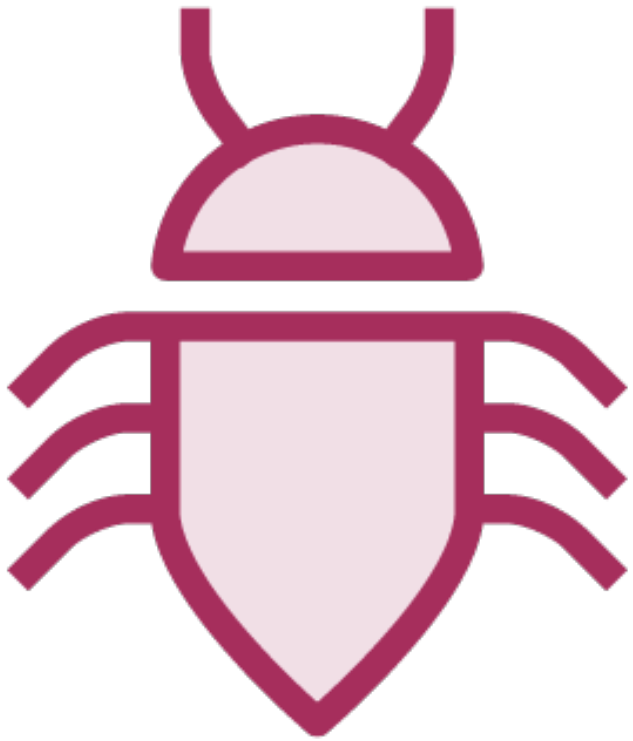


Common in browsers

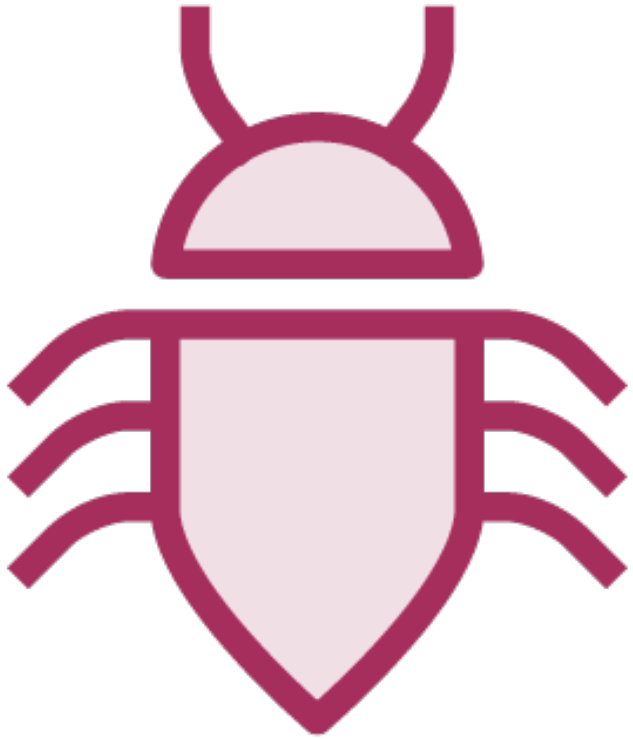
- JavaScript events can delete an object at unexpected times
  - While back in C++ of browser
    - Object is about to get used again



- Chrome
  - Best sandbox, but look out for kernel exploits, and sandbox escapes
  - The usual bugs as well, but less of them
- Safari
  - Webkit
    - Google forked to their blink
    - Not thinking that will help Apples security posture



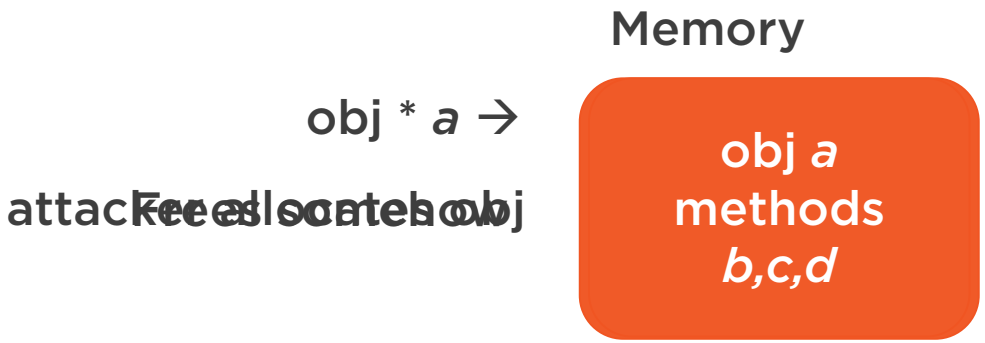
- Internet Explorer
  - UAF examples in Metasploit
  - Edge seems a bit better
- Firefox
  - Bugzilla is helpful for finding new bugs to explore
    - Open source can make security harder actually
- Opera
  - Security through obscurity?
    - RWX in mem, bugs galore, etc.



Many well-publicized campaigns leveraged a zero-day IE UaF vulnerability

- Operation SnowMan (CVE-2014-0322)
- Operation Clandestine Fox (CVE-2014-1776)
- MS14-65
  - 10 memory corruption bugs were patched
  - Most were UaF issues
    - CVE-2014-4143 is one that affected IE6-IE11

Use-after-Free (UAF)

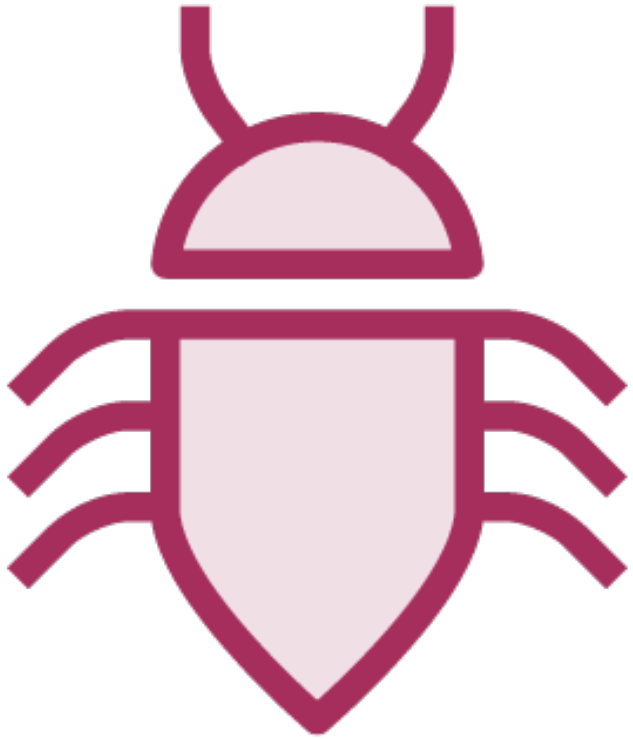


1. *a→b*() is called by application(e.g. original obj used after freed)
2. But expected virtual pointer is not present
3. Instead program dereferences attacker controlled data (asfunc pointer)
4. Which may allow any of the three primitives: R/W/X

Smart Pointers

```
// Need to create the object to achieve some goal
MyObject* ptr = new MyObject();
ptr->DoSomething();// Use the object in some way.
delete ptr; // Destroy the object. Done with it.
// Wait, what if DoSomething() raises an exception....
```

```
SomeSmartPtr<MyObject> ptr(new MyObject());
ptr->DoSomething(); // Use the object in some way.

// Destruction of the object happens, depending
// on the policy the smart pointer class uses.

// Destruction would happen even if DoSomething()
// raises an exception
```



Why Exploitable?

- Generally allocator will prefer to return a memory chunk that was just freed
  - Prevent fragmentation
- Use of “smart pointers” vs. “native pointers” is a good bet
  - Some windows APIs may required native pointer

Webkit UAF Example from Older Chrome

setOuterText in HTMLElement.cpp

```
// Is previous node a text node? If so, merge into it.
Node* prev = t->previousSibling();
if (prev && prev->isTextNode()) {
    Text* textPrev = static_cast<Text*>(prev);
    textPrev->appendData(t->data(), ec);
    if (ec)
        return;
    t->remove(ec);
    if (ec)
        return;
    t = textPrev;
}

// Is next node a text node? If so, merge it in.
Node* next = t->nextSibling();
if (next && next->isTextNode()) {
    Text* textNext = static_cast<Text*>(next);
    t->appendData(textNext->data(), ec);  //can remove what textNext points at, since not ref pointers.  look for raw ptrs as pattern
    if (ec)
        return;
    textNext->remove(ec);
    if (ec)
        return;
}
```

Non-ref ptr defined

Uh oh. Possible UAF

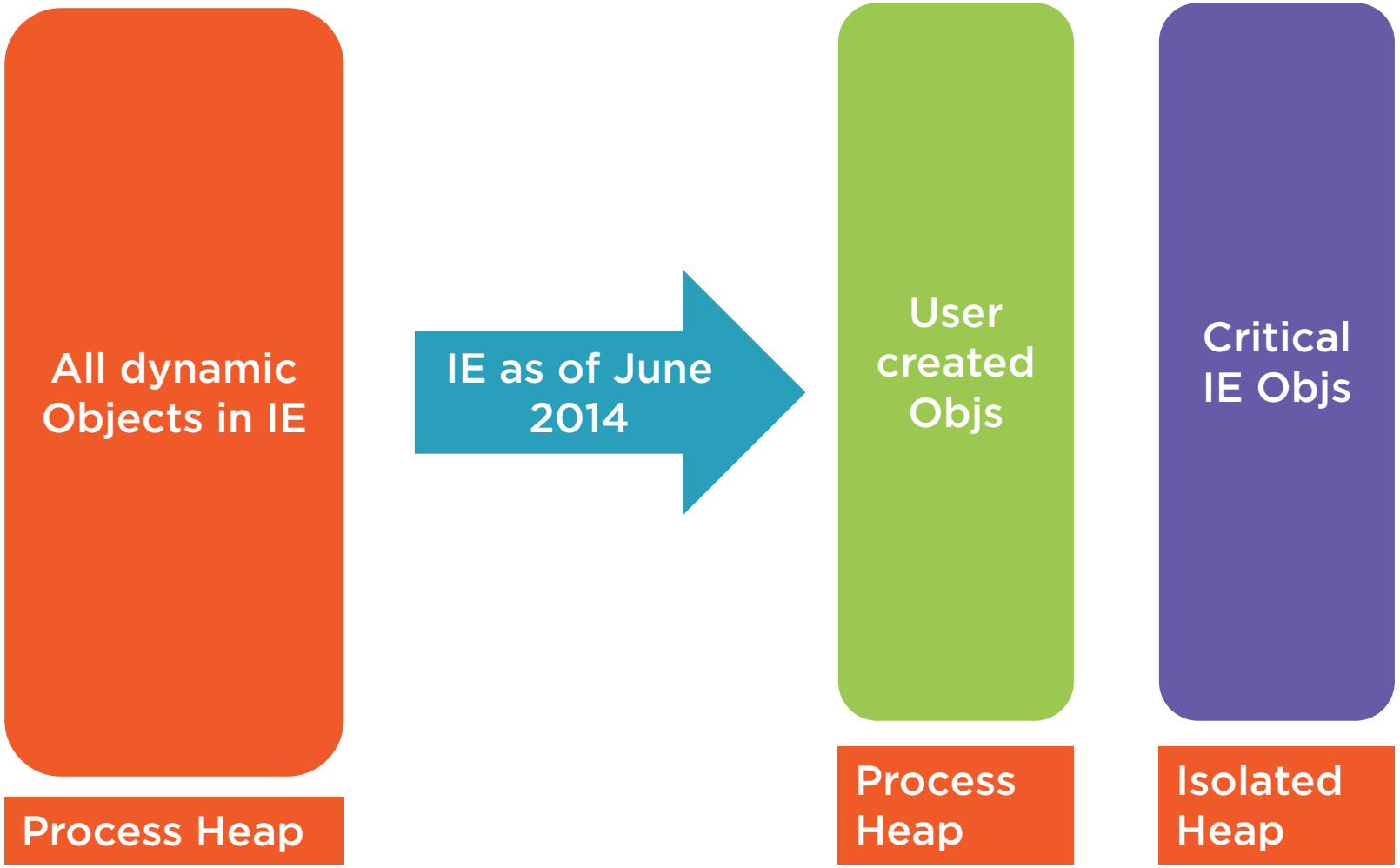
UAF: Fixed

```
static void mergeWithNextTextNode(PassRefPtr<Node> node, ExceptionCode& ec)
{
    ASSERT(node && node->isTextNode());
    Node* next = node->nextSibling();
    if (!next || !next->isTextNode())
        return;

    RefPtr<Text> textNode = static_cast<Text*>(node.get());
    RefPtr<Text> textNext = static_cast<Text*>(next);
    textNode->appendData(textNext->data(), ec);
    if (ec)
        return;
    if (textNext->parentNode()) // Might have been removed by mutation event.
        textNext->remove(ec);
}

void HTMLElement::setOuterHTML(const String& html, ExceptionCode& ec)
{
    Node* p = parentNode();
    if (!p || !p->isHTMLElement()) {
        ec = NO_MODIFICATION_ALLOWED_ERR;
        return;
    }
    RefPtr<HTMLElement> parent = toHTMLElement(p);
    RefPtr<Node> prev = previousSibling();
    RefPtr<Node> next = nextSibling();

    RefPtr<DocumentFragment> fragment = createFragmentFromSource(html, parent.get(), ec);
    if (ec)
        return;

    parent->replaceChild(fragment.release(), this, ec);
    RefPtr<Node> node = next ? next->previousSibling() : 0;
```

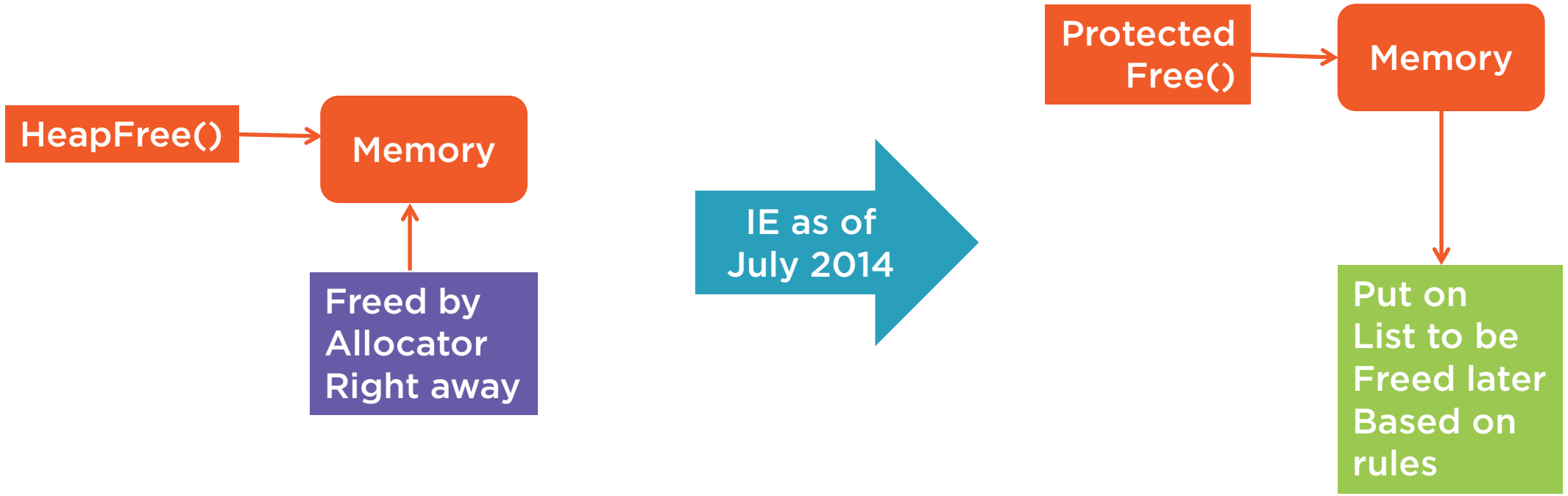
Now uses reference pointer

Further Application Specific Protections?

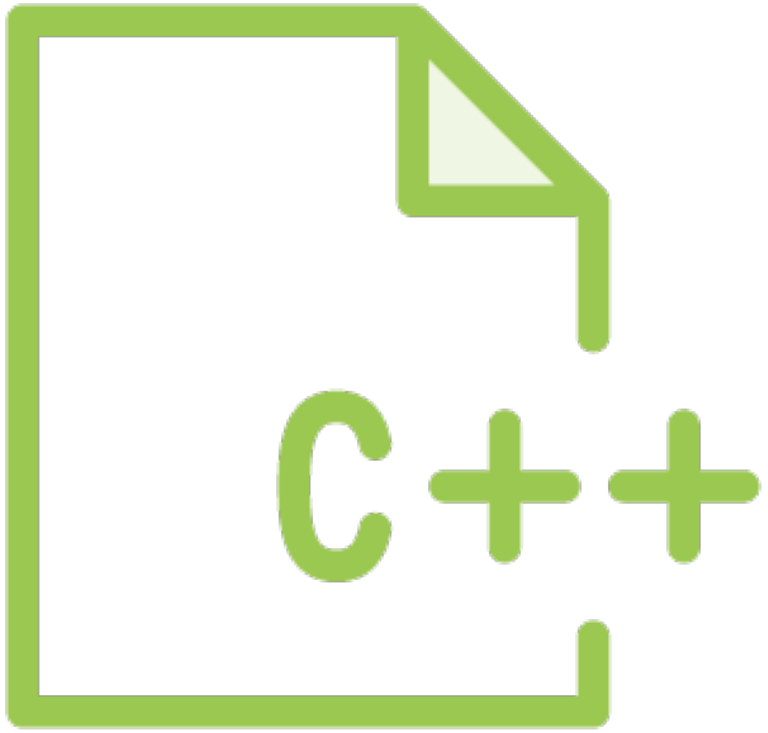
Isolated Heap



Delayed Free

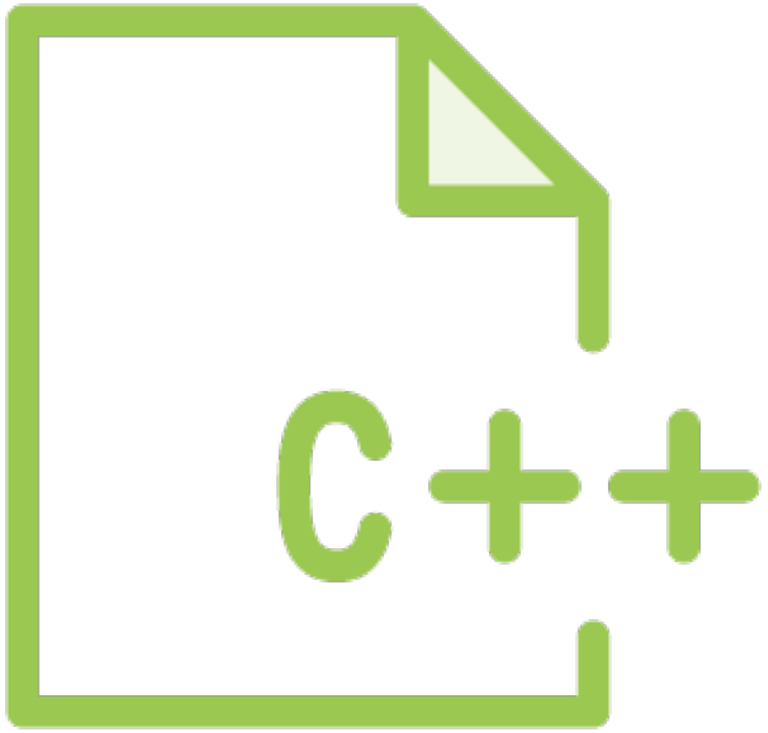


Type Confusion

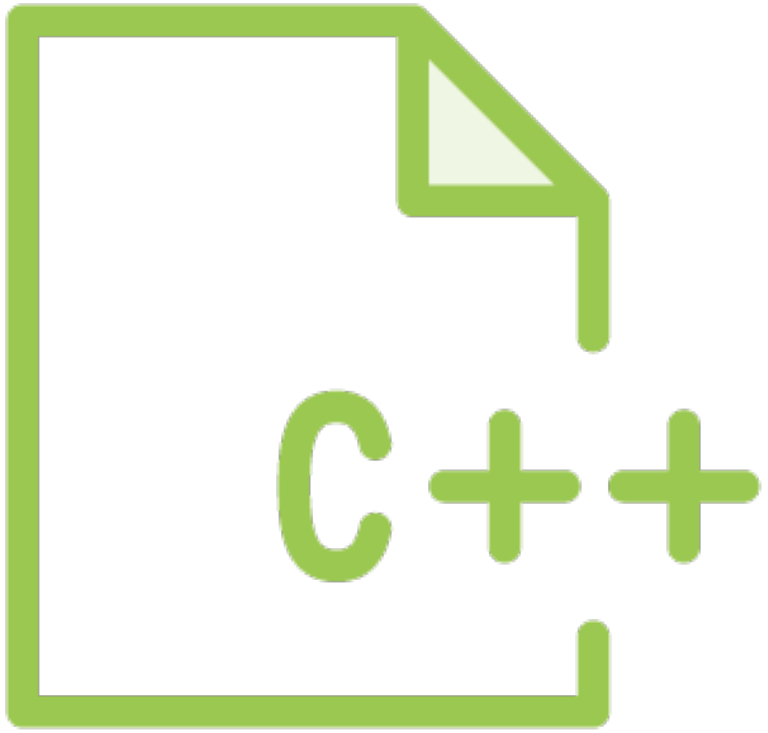


Type Casting

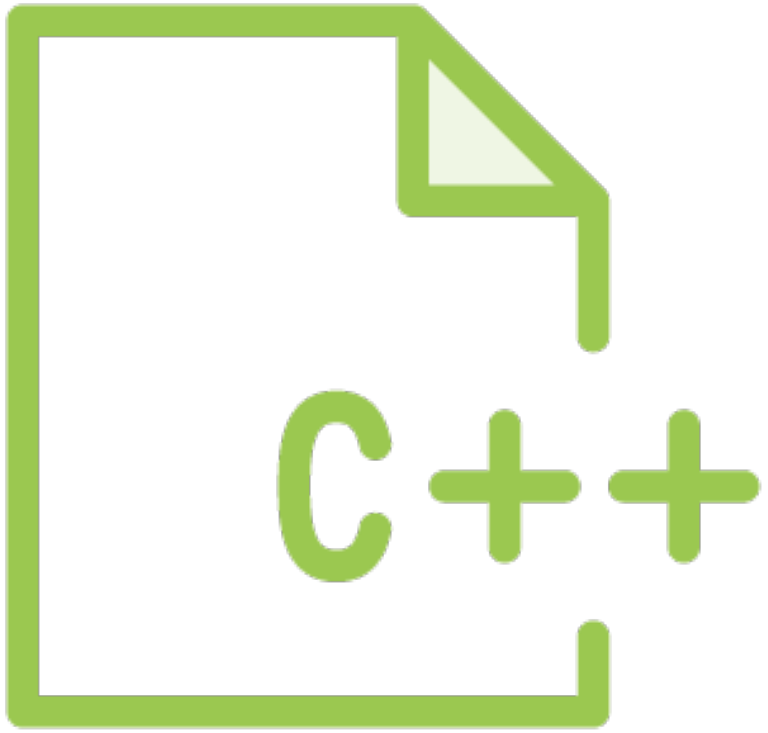
- Generally explicit on built-in types for C
- But, often used with more complex classes for C++
- Three main types



- Dynamic_cast
  - Safer
    - Uses RTTI
    - Must be ptr or reference
    - Returns NULL or throws exception if invalid/mismatched

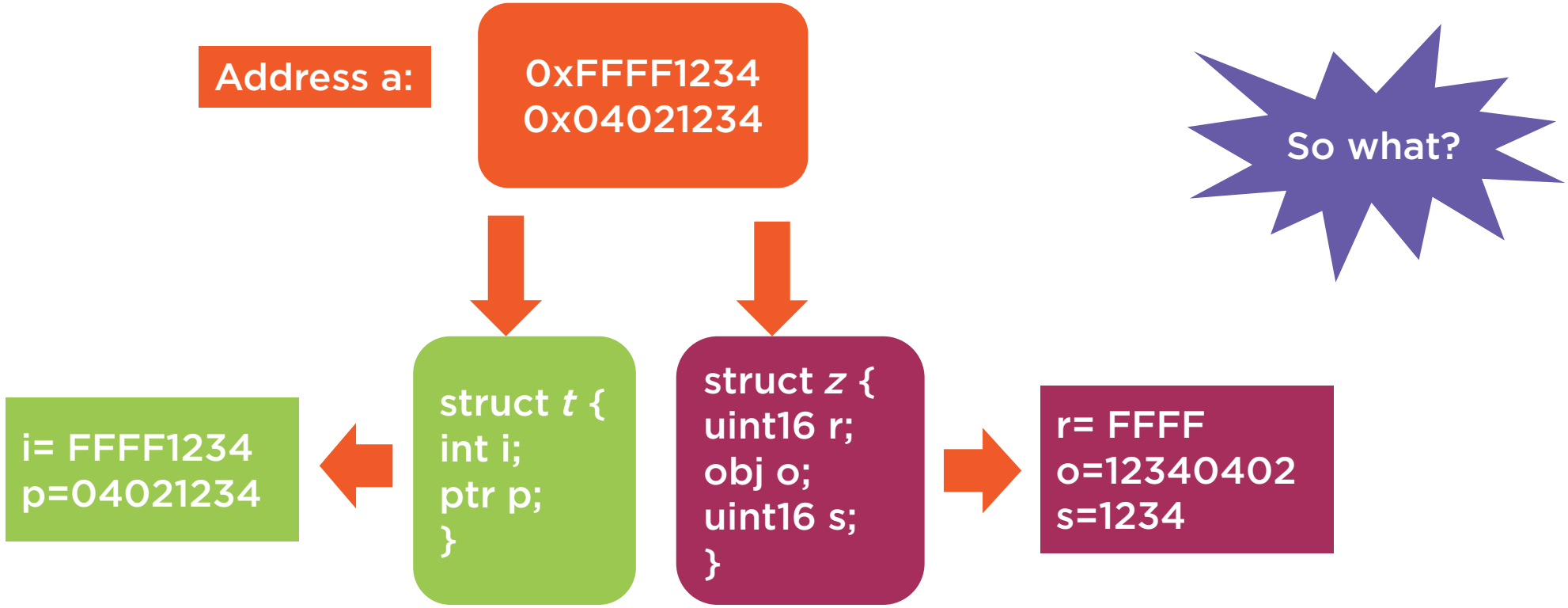


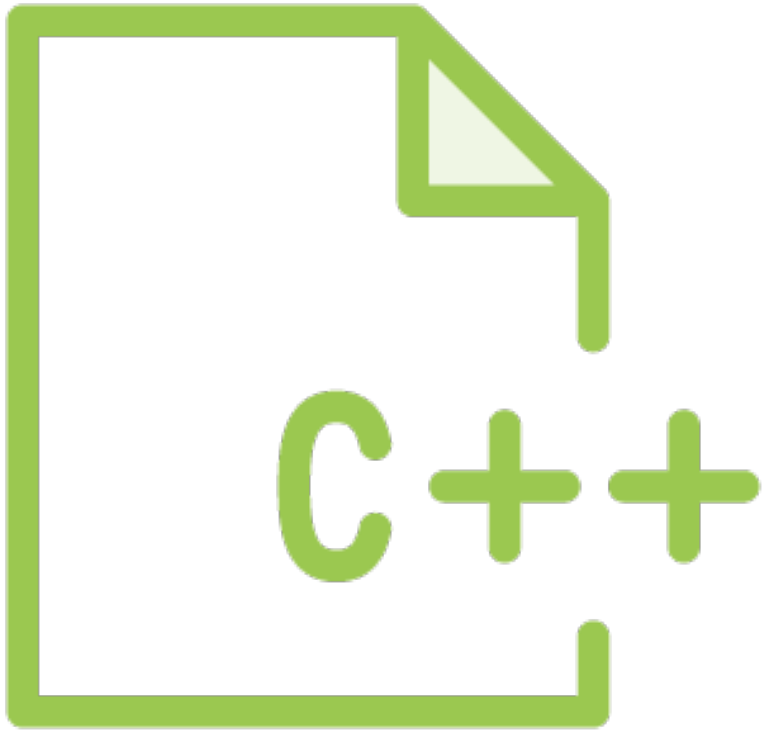
- Static_cast
  - Dangerous for objects
    - But mostly used on basic types
    - No exception or NULL return



- Reinterpret_cast
  - No safety checking
    - Performs conversion
      - *Usually unsafe for structs and objs*

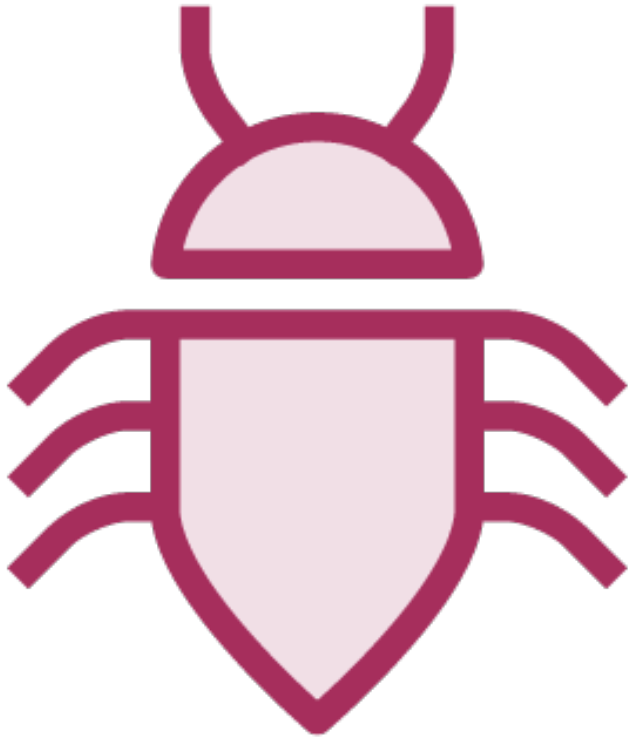
Interpreting data at address *a* of type *t* - to be of type *z*





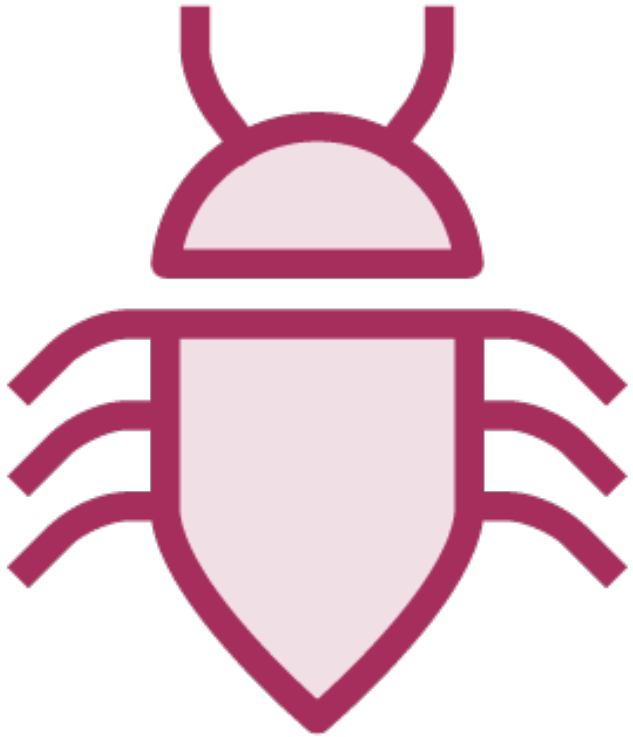
Union/struct with type that describes data

- Java
  - Or others using byte code
- IPC
  - Webkit
    - Chrome
    - Safari
- JavaScript
  - Browsers
- Adobe Flash



Example

- TC in JavaScript
  - Chained with kernel pool exploit to achieve chrome escape



Vulnerability in WebKit

- Type confusion
- Handling of view targets in SVG documents
- Possible to specify a viewTarget for the document, which specifies non-SVG Element
- Vulnerable code from WebCore/svg/SVGViewSpec.cpp

```
SVGElement* SVGViewSpec::viewTarget() const

{

    if (!m_contextElement)

        return 0;

    return static_cast<SVGElement*>(m_contextElement->
        treeScope()->getElementById(m_viewTargetString));

}
```

Flexible Exploit Primitives

SVGElement*

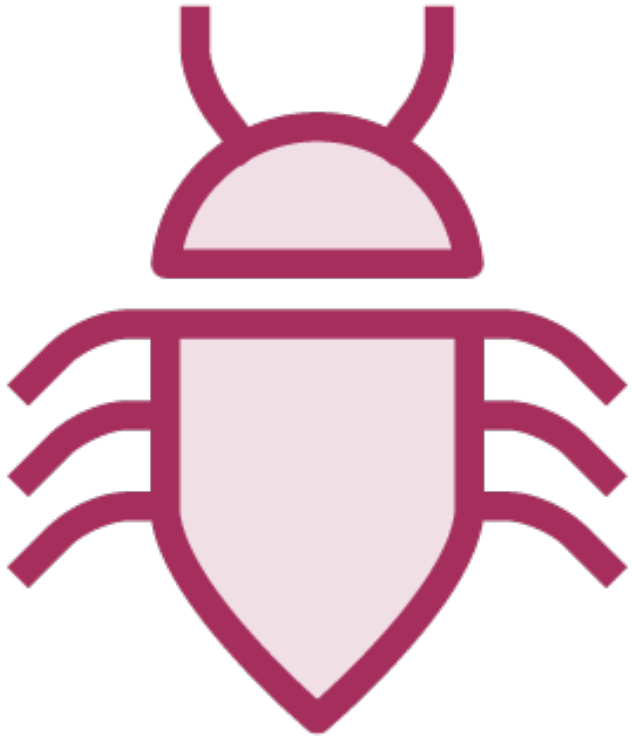
- A
- B
- C
- D
- E
- F
- G

F/G

Legit shorter data type

- (HTMLUnknownElement)
- 1
- 2
- 3
- 4

Accessing these members resulting in out of bounds access. Heap grooming allows for a valid access, and a memory leak



Exploit

- Leak pointer, calculate base address, read DLL for gadgets, build ROP chain, win!

Fix

- No cast
- Input validation
  - Type checking

```
SVGElement* SVGViewSpec::viewTarget() const
{
    if (!m_contextElement)
        return 0;
    Element* element = m_contextElement->
    treeScope()->getElementById(m_viewTargetString));
    if( !element || !element->isSVGElement() )
        return 0;
    return toSVGElement(element);
}
```

Summary

Still memory corruption

- UaF and TC

Best practices

- Design review
  - CFG for example in VS2015
  - Smart pointers
  - Composition analysis
- Automated static analysis
  - Careful peer review
- Dynamic analysis
  - Fuzzing