Exploring C++ Program Details Related to Security

Dr. Jared DeMott
CTO AND FOUNDER

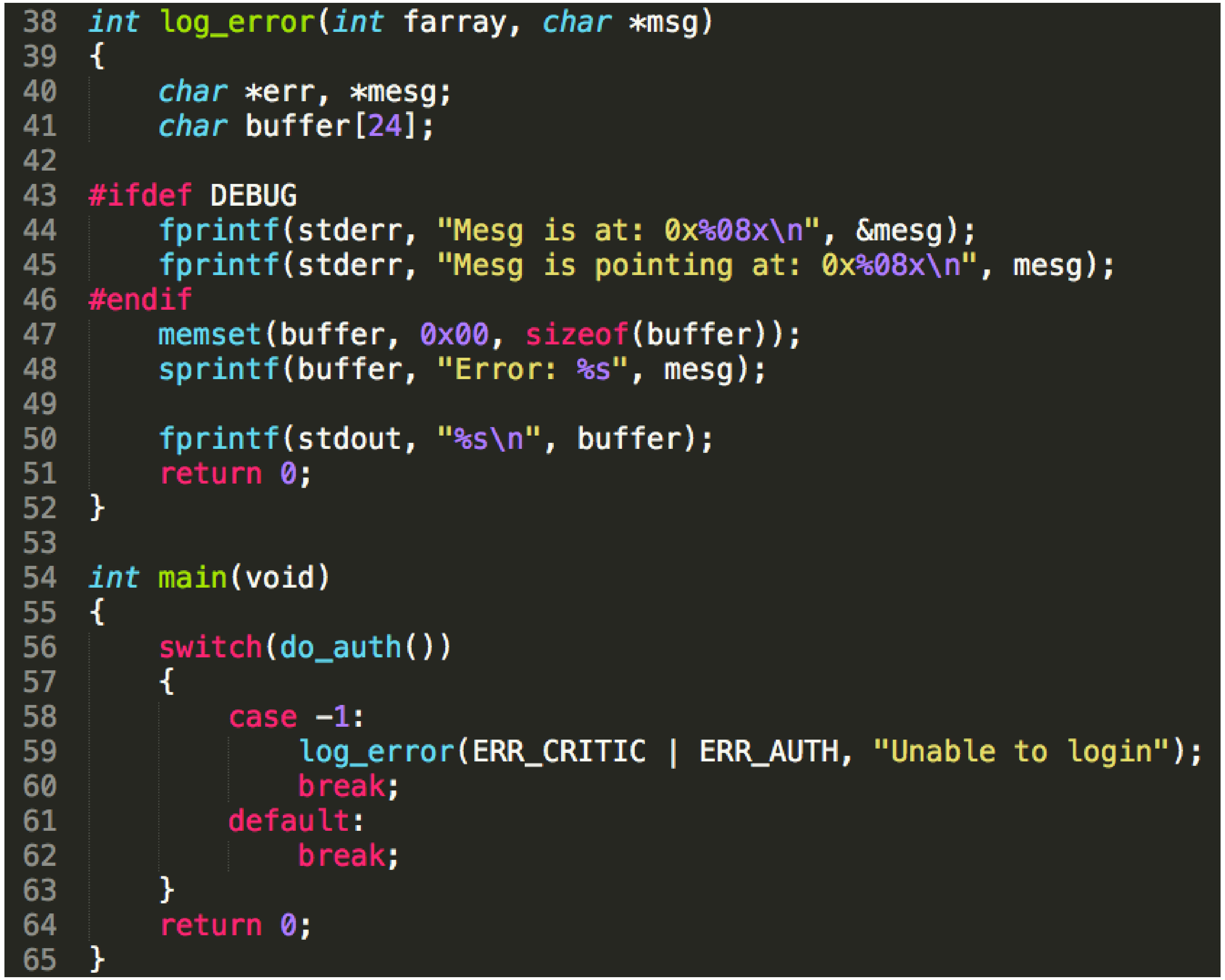
@jareddemott www.vdalabs.com

Overview

Teaser demo

C++

Bugs



```c
int log_error(int farray, char *msg)
{
    char *err, *mesg;
    char buffer[24];

#ifdef DEBUG
    fprintf(stderr, "Mesg is at: 0x%08x\n", &mesg);
    fprintf(stderr, "Mesg is pointing at: 0x%08x\n", mesg);
#endif
    memset(buffer, 0x00, sizeof(buffer));
    sprintf(buffer, "Error: %s", mesg);

    fprintf(stdout, "%s\n", buffer);
    return 0;
}

int main(void)
{
    switch(do_auth())
    {
        case -1:
            log_error(ERR_CRITIC | ERR_AUTH, "Unable to login");
            break;
        default:
            break;
    }
    return 0;
}
```

Demo

Show the bug I’ve been teasing you with

Direct descendant of C

- Self-managed, typed, compiled (native) language
- Middle-level language
  - Both high-level and low-level language features
  - Developed in 1979 at Bell Labs as an enhancement to the C programming language
    - Named "C with Classes”
    - Renamed to C++ in 1983

Interpreted code does not work for everything

- Desktop apps, embedded software, high-performance networking, kernels, hypervisors, and entertainment

Several groups provide both free and commercial C++ compiler software

- LLVM/CLANG, GNU Project, Microsoft, Intel, Borland

“Practically every computer language has *gotchas* -- constructs or combinations of constructs that software developers are likely to use incorrectly. Sadly, the C and C++ languages have an unusually large number of gotchas, and many of these gotchas tend to lead directly to dangerous security vulnerabilities.”

-- David A. Wheeler

Similar to C

- Code organization
- Types
    - Operations on types
- Arrays
- Escape sequences
- Pointers
- Basic logic constructs
    - *for* loop, etc
- General Build procedures
    - Shared IDE’s like Visual Studio
- Function construct

Differences

- Object-oriented
  - Reusable objects
    - Code and data
- Different compiler
- Extended concepts and syntax
  - Classes, virtual functions, etc.
- Extended std libs
  - E.g. printf vs. cout
- New types
  - Vectors, lists, arrays
    - *Use these and don’t mix!*

Allocation Mismatch

incorrect

```
int *p_var = int;
delete p_var;
```

incorrect

```
int *p_var = malloc(sizeof(int));
delete p_var;
```

Correct

```
int *p_var = new int;
delete p_var;
```

Variable Length Array

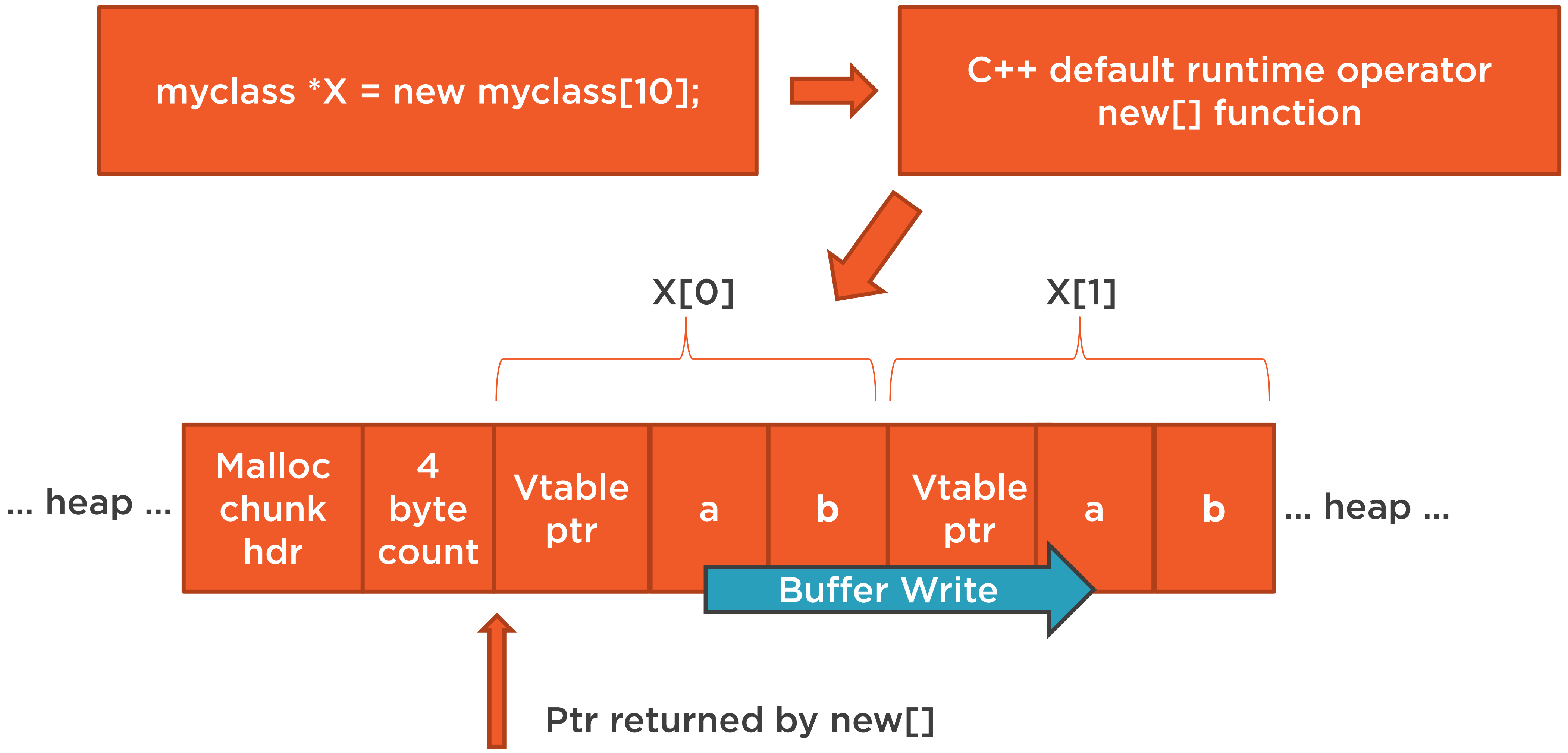
An alternative to container classes

Look and feel like c arrays

```
int * array_of_ints_ptr = new int[40]
x = array_of_ints_ptr[15]
```

Unlike container classes, no access protection is guaranteed

VLA Internals



Using namespace <namespace name>

- Often groups classes
- Indicates where the header file is located
  - #include <iostream> for example is in *std* namespace

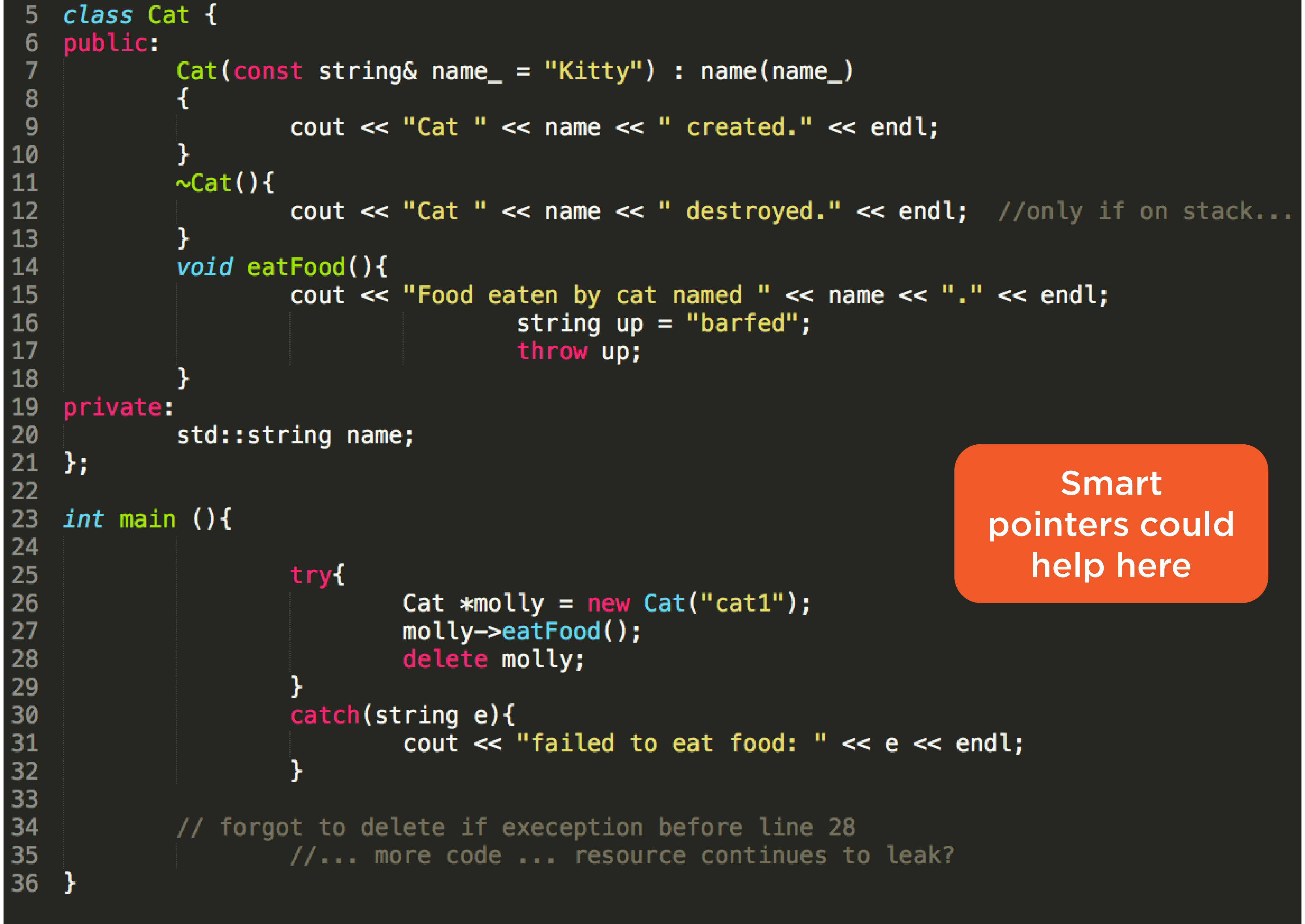
Return values no longer required

- Classes can throw exceptions

```
#include <iostream> using namespace std;

int main() {
  double Operand1, Operand2, Result;
  cout << "This program allows you to perform a division of two numbers\n";
  cout << "To proceed, enter two numbers: ";
  try {
    cout << "First Number: "; cin >> Operand1;
    cout << "Second Number: "; cin >> Operand2;
    if( Operand2 == 0 ) throw "Division by zero not allowed";
    Result = Operand1 / Operand2;
    cout << "\n" << Operand1 << " / " << Operand2 << " = " << Result << "\n\n";
  } catch(const char* Str) {
    cout << "\nBad Operator: " << Str;
  }
  return 0;

}
```



```
 5  class Cat {
 6  public:
 7          Cat(const string& name_ = "Kitty") : name(name_)
 8          {
 9                  cout << "Cat " << name << " created." << endl;
10          }
11          ~Cat(){
12                  cout << "Cat " << name << " destroyed." << endl;  //only if on stack...
13          }
14          void eatFood(){
15                  cout << "Food eaten by cat named " << name << "." << endl;
16                                  string up = "barfed";
17                                  throw up;
18          }
19  private:
20          std::string name;
21  };
22
23  int main (){
24
25                  try{
26                          Cat *molly = new Cat("cat1");
27                          molly->eatFood();
28                          delete molly;
29                  }
30                  catch(string e){
31                          cout << "failed to eat food: " << e << endl;
32                  }
33
34          // forgot to delete if exeception before line 28
35                  //... more code ... resource continues to leak?
36  }
```

Smart pointers could help here

Classes

- Set of values and set of operations
  - Abstraction, encapsulation, inheritance, and polymorphism
- Objects are instances of classes

Templates

- Generic argument functions or “generic programming”
  - Can make code very hard to really audit if over used!

- Encapsulation
    - Allows members to be declared as
        - Public, private, or protected
- Inheritance
    - Allows one data type to acquire properties of other data types
- Polymorphism
    - Enables one common interface for many implementations
        - Objects to act differently under different circumstances

```
class Adder {
     public:
          Adder(int i = 0) { total = i; }                  // constructor
          void addNum(int number) { total += number; }     // interface to outside world
          int getTotal() { return total; };                // interface to outside world
     private:
          int total;     // hidden data from outside world
};
int main( ) {
     Adder a;  a.addNum(10);  a.addNum(20);  a.addNum(30);
     cout << "Total " << a.getTotal() << endl;
}
```

Total = 60

Assignment

- Operator overloading
  - Constructing your own operator in a class
- Use the keyword *operator* in the function declaration

Define the Equal Operator

```
void Data::operator=(Date& newdate)
{
   day   = newdate.day;      // assign the day
   month = newdate.month;    // assign the month
   year  = newdate.year;     // assign the year
   return;
}
```

Inheritance

- Base class
  - Initial class used for derivation
    - Also called parent or superclass
- Derived class
  - Class created from a base class
    - Also called child or subclass
- Derived class incorporates all data and member functions of its base class
  - Can add new or override existing data and member functions

```
class Shape {
    public:
        void setWidth(int w) {
            width = w;
        }
        void setHeight(int h) {
            height = h;
        }
    protected:
        int width;
        int height;
};
```

Base Class

```
class Rectangle: public Shape {
    public:
        int getArea() {
            return (width * height);
        }
};
```

Derived Class

```
int main(void) {
    Rectangle Rect;
    Rect.setWidth(5);
    Rect.setHeight(7);
    // Print the area of the object
    cout << "Total area: " << Rect.getArea() << endl;
    return 0;
}
```

```
Total area: 35
```

Polymorphism

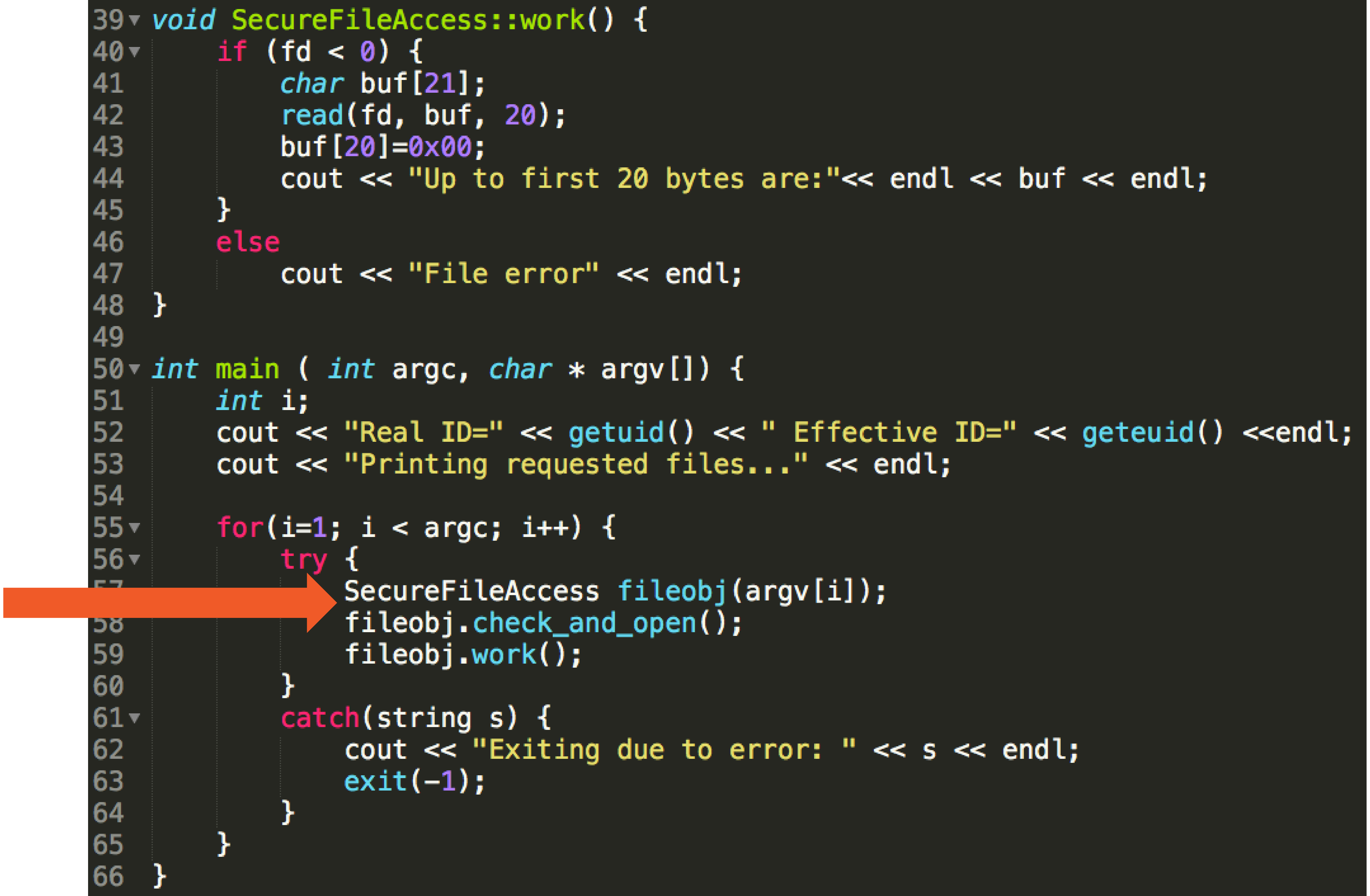
- Manipulation/overloading of objects to suit current need
  - Same function name used in both base and derived
- Static binding
  - Decision of which function to use is made at compile time
- Dynamic (virtual) binding
  - Decision of which function to use is made at run time
- RTTI
  - Information about an object data type in memory at runtime

Dynamic binding

- Creates a pointer to a function
  - Value not assigned until the function is actually called
- Use ***`virtual`*** keyword in base class
  - Override versions must have the same return type and parameter list
    - *Same name, different actions*

Demo

Show a basic program with virtual functions

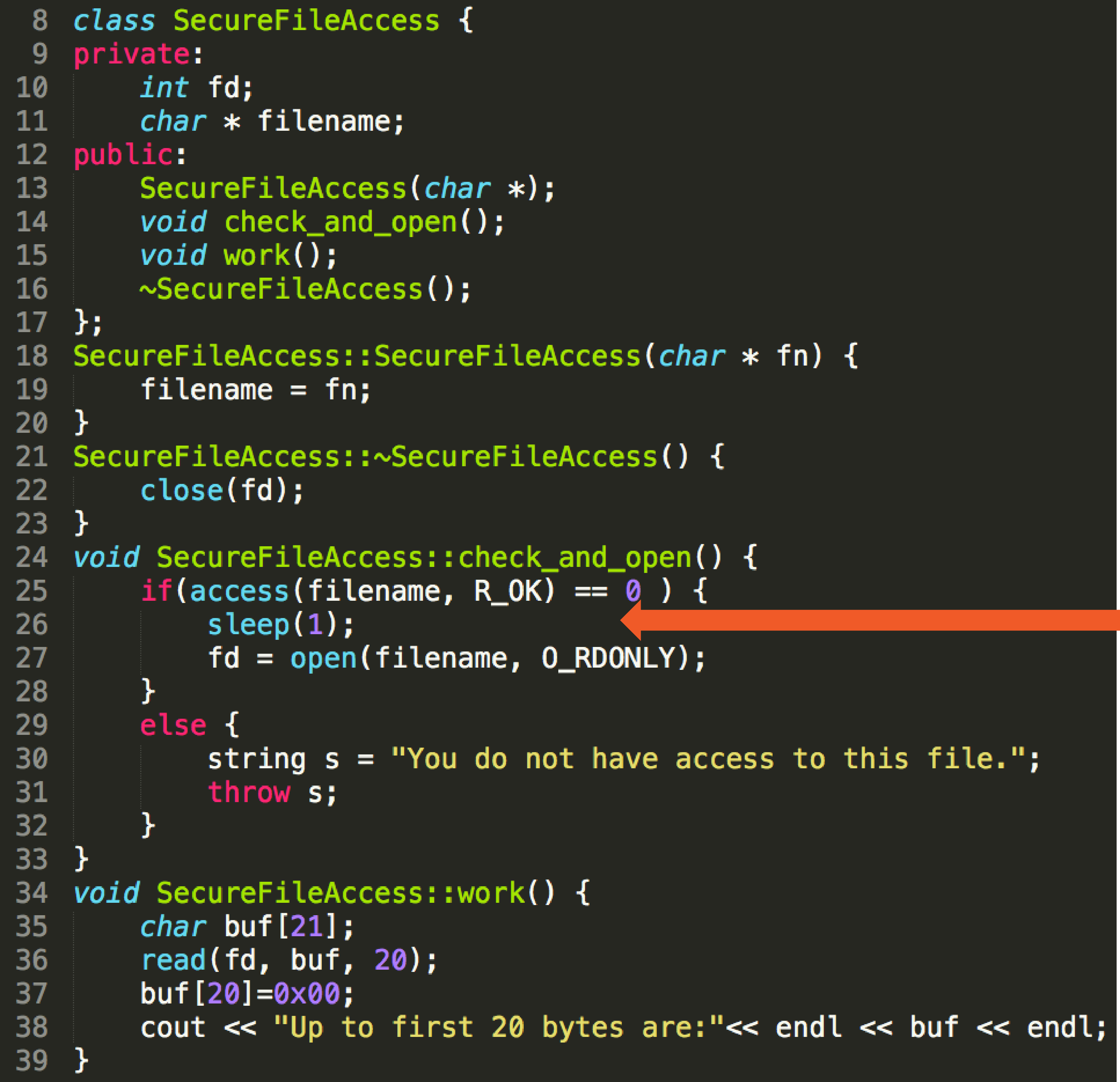


```cpp
void SecureFileAccess::work() {
    if (fd < 0) {
        char buf[21];
        read(fd, buf, 20);
        buf[20]=0x00;
        cout << "Up to first 20 bytes are:"<< endl << buf << endl;
    }
    else
        cout << "File error" << endl;
}

int main ( int argc, char * argv[]) {
    int i;
    cout << "Real ID=" << getuid() << " Effective ID=" << geteuid() <<endl;
    cout << "Printing requested files..." << endl;

    for(i=1; i < argc; i++) {
        try {
            SecureFileAccess fileobj(argv[i]);
            fileobj.check_and_open();
            fileobj.work();
        }
        catch(string s) {
            cout << "Exiting due to error: " << s << endl;
            exit(-1);
        }
    }
}
```

File Access Race Conditions



```cpp
class SecureFileAccess {
private:
    int fd;
    char * filename;
public:
    SecureFileAccess(char *);
    void check_and_open();
    void work();
    ~SecureFileAccess();
};
SecureFileAccess::SecureFileAccess(char * fn) {
    filename = fn;
}
SecureFileAccess::~SecureFileAccess() {
    close(fd);
}
void SecureFileAccess::check_and_open() {
    if(access(filename, R_OK) == 0 ) {
        sleep(1);
        fd = open(filename, O_RDONLY);
    }
    else {
        string s = "You do not have access to this file.";
        throw s;
    }
}
void SecureFileAccess::work() {
    char buf[21];
    read(fd, buf, 20);
    buf[20]=0x00;
    cout << "Up to first 20 bytes are:"<< endl << buf << endl;
}
```

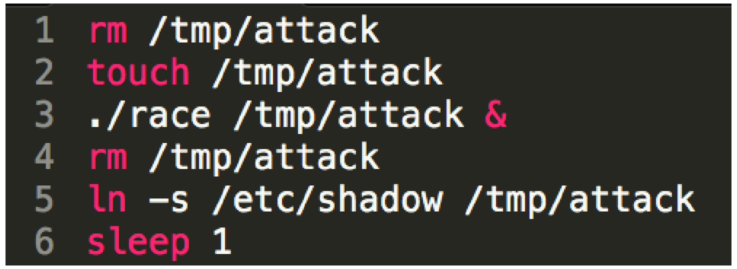
The program is SUID root for bug to have max security impact

Warning: Using **access**() to check if a user is authorized to, for example, open a file before actually doing so using ***open****(2)* creates a security hole, because the user might exploit the short time interval between checking and opening the file to manipulate it. **For this reason, the use of this system call should be avoided.** (In the example just described, a safer alternative would be to temporarily switch the process's effective user ID to the real ID and then call ***open****(2)*.)

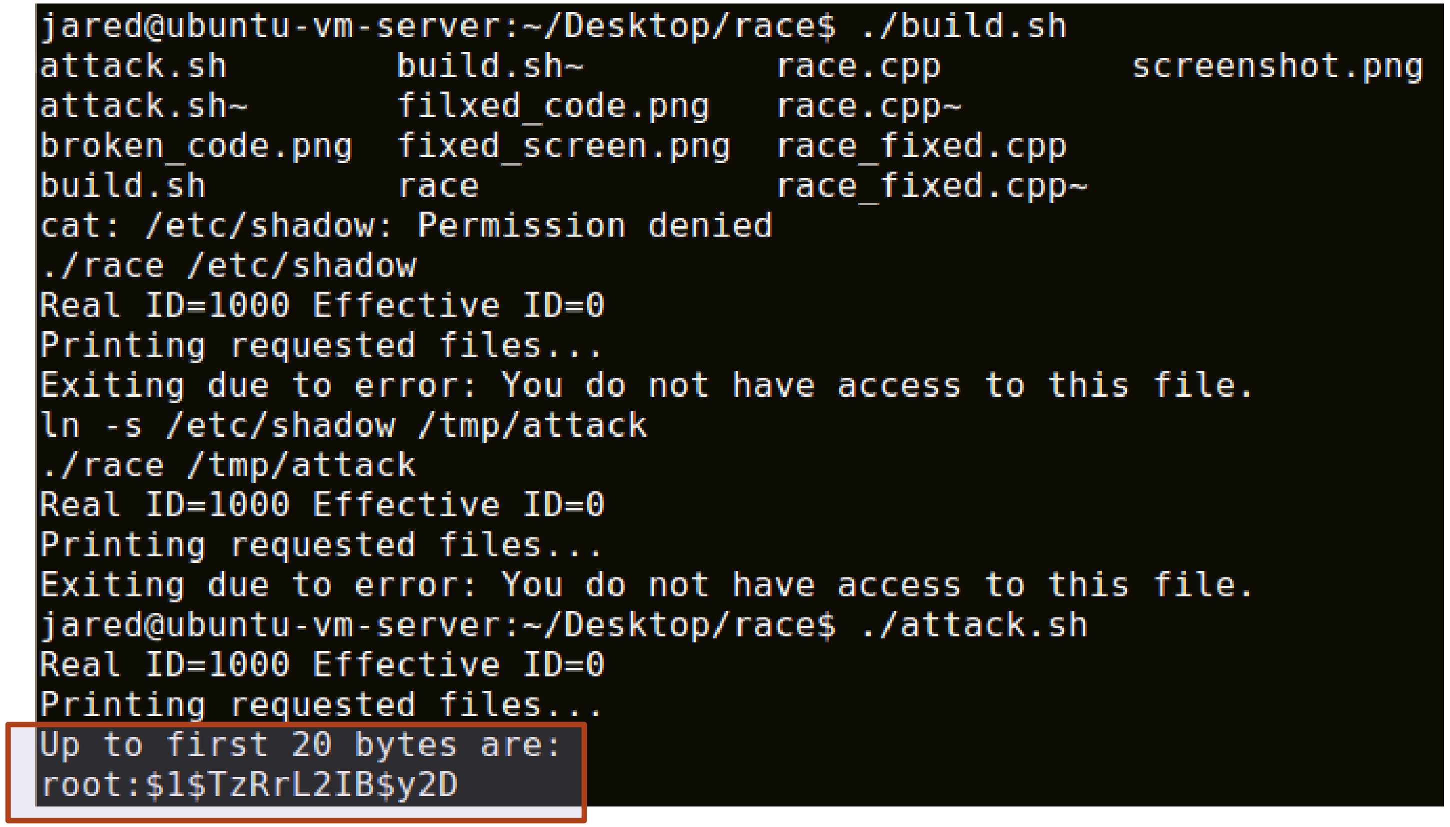
access() always dereferences symbolic links. If you need to check the permissions on a symbolic link, use ***faccessat****(2)* with the flag **AT_SYMLINK_NOFOLLOW**.

access() returns an error if any of the access types in *mode* is denied, even if some of the other access types in *mode* are permitted.

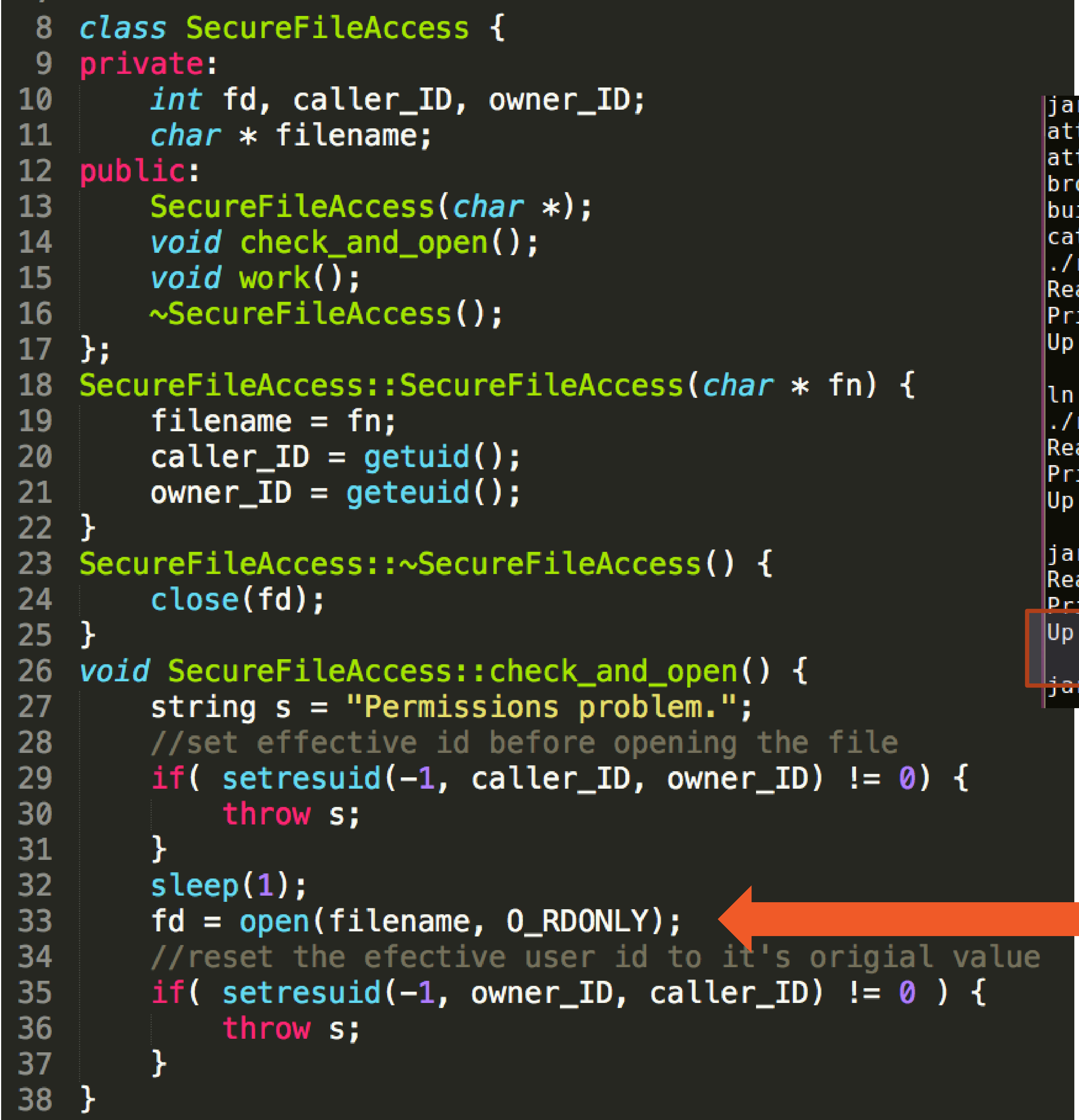
Exploit



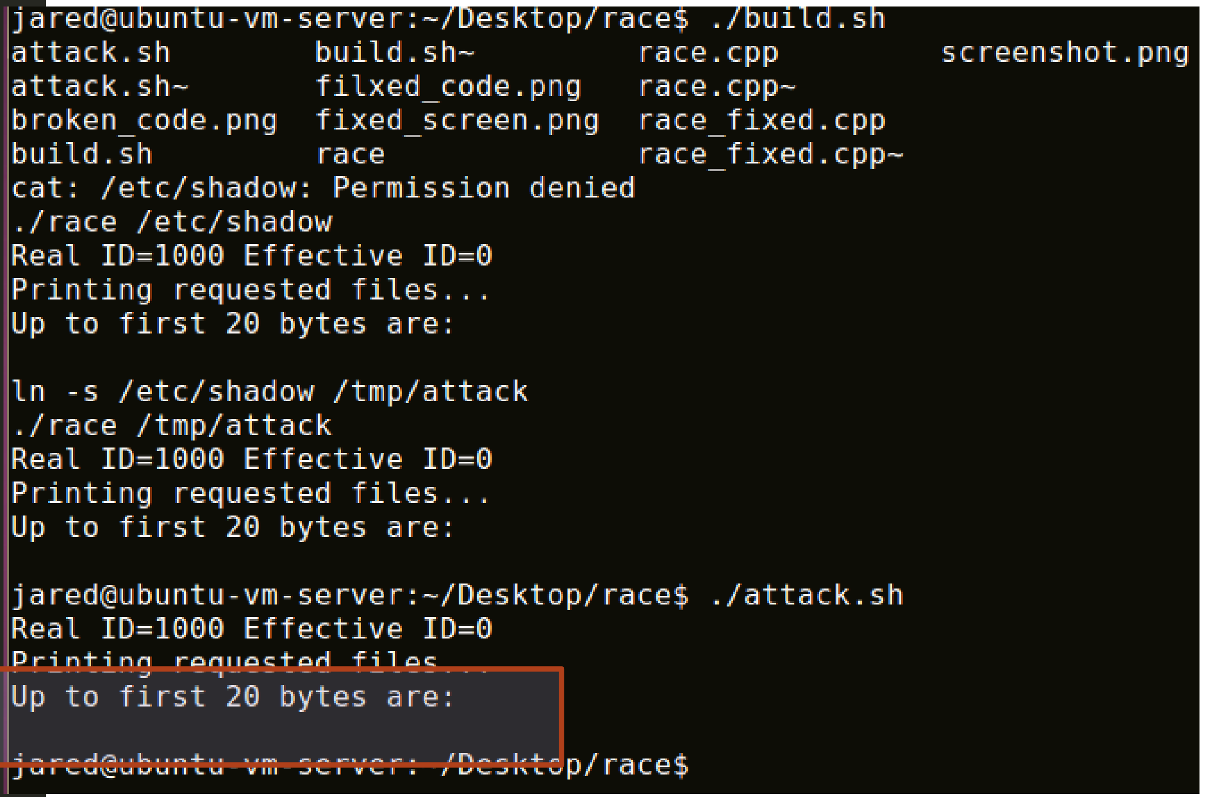
```
rm /tmp/attack
touch /tmp/attack
./race /tmp/attack &
rm /tmp/attack
ln -s /etc/shadow /tmp/attack
sleep 1
```



```
jared@ubuntu-vm-server:~/Desktop/race$ ./build.sh
attack.sh         build.sh~          race.cpp          screenshot.png
attack.sh~        filxed_code.png    race.cpp~
broken_code.png   fixed_screen.png   race_fixed.cpp
build.sh          race               race_fixed.cpp~
cat: /etc/shadow: Permission denied
./race /etc/shadow
Real ID=1000 Effective ID=0
Printing requested files...
Exiting due to error: You do not have access to this file.
ln -s /etc/shadow /tmp/attack
./race /tmp/attack
Real ID=1000 Effective ID=0
Printing requested files...
Exiting due to error: You do not have access to this file.
jared@ubuntu-vm-server:~/Desktop/race$ ./attack.sh
Real ID=1000 Effective ID=0
Printing requested files...
Up to first 20 bytes are:
root:$1$TzRrL2IB$y2D
```



```cpp
class SecureFileAccess {
private:
    int fd, caller_ID, owner_ID;
    char * filename;
public:
    SecureFileAccess(char *);
    void check_and_open();
    void work();
    ~SecureFileAccess();
};
SecureFileAccess::SecureFileAccess(char * fn) {
    filename = fn;
    caller_ID = getuid();
    owner_ID = geteuid();
}
SecureFileAccess::~SecureFileAccess() {
    close(fd);
}
void SecureFileAccess::check_and_open() {
    string s = "Permissions problem.";
    //set effective id before opening the file
    if( setresuid(-1, caller_ID, owner_ID) != 0) {
        throw s;
    }
    sleep(1);
    fd = open(filename, O_RDONLY);
    //reset the efective user id to it's origial value
    if( setresuid(-1, owner_ID, caller_ID) != 0 ) {
        throw s;
    }
}
```



```
jared@ubuntu-vm-server:~/Desktop/race$ ./build.sh
attack.sh         build.sh~          race.cpp          screenshot.png
attack.sh~        filxed_code.png    race.cpp~
broken_code.png   fixed_screen.png   race_fixed.cpp
build.sh          race               race_fixed.cpp~
cat: /etc/shadow: Permission denied
./race /etc/shadow
Real ID=1000 Effective ID=0
Printing requested files...
Up to first 20 bytes are:

ln -s /etc/shadow /tmp/attack
./race /tmp/attack
Real ID=1000 Effective ID=0
Printing requested files...
Up to first 20 bytes are:

jared@ubuntu-vm-server:~/Desktop/race$ ./attack.sh
Real ID=1000 Effective ID=0
Printing requested files...
Up to first 20 bytes are:

jared@ubuntu-vm-server:~/Desktop/race$
```

Fixed

Summary

C++

- Prefer local objects
    - Less life cycle management
- Or at least use new types

Exception handling

Race condition