EXERCISE #24

SSDLC REVIEW

Write your name and answer the following on a piece of paper

At what point in the software development life cycle should threat modelling begin?

ADMINISTRIVIA AND ANNOUNCEMENTS

Preparing for Quiz 2

Review session Wednesday at 7:00 – 9:00 (tentative)

We have to talk about Quiz 1

A tale of two classes…

EECS 677:	EECS 700:
Highest grade: 50/50	Highest grade: 50/50
Lowest grade: 25/50	Lowest grade: 8/50
Average grade: ~84%	Average grade: ~53%
Median grade: ~89%	Median grade: ~50%

LINTING

EECS 677: Software Security Evaluation

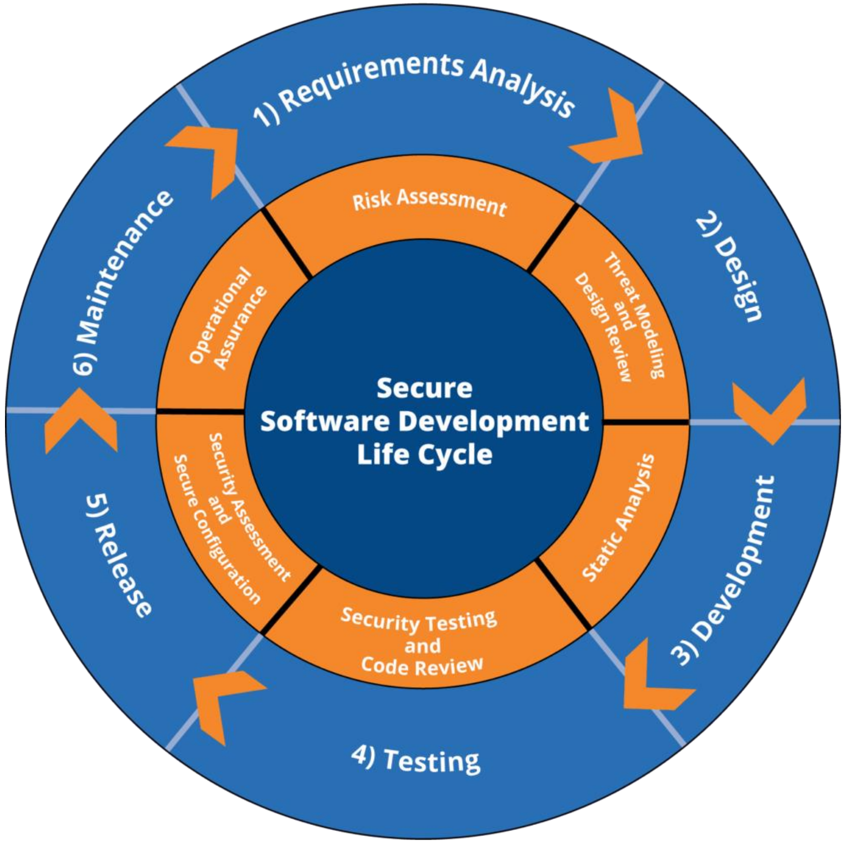
Drew Davidson

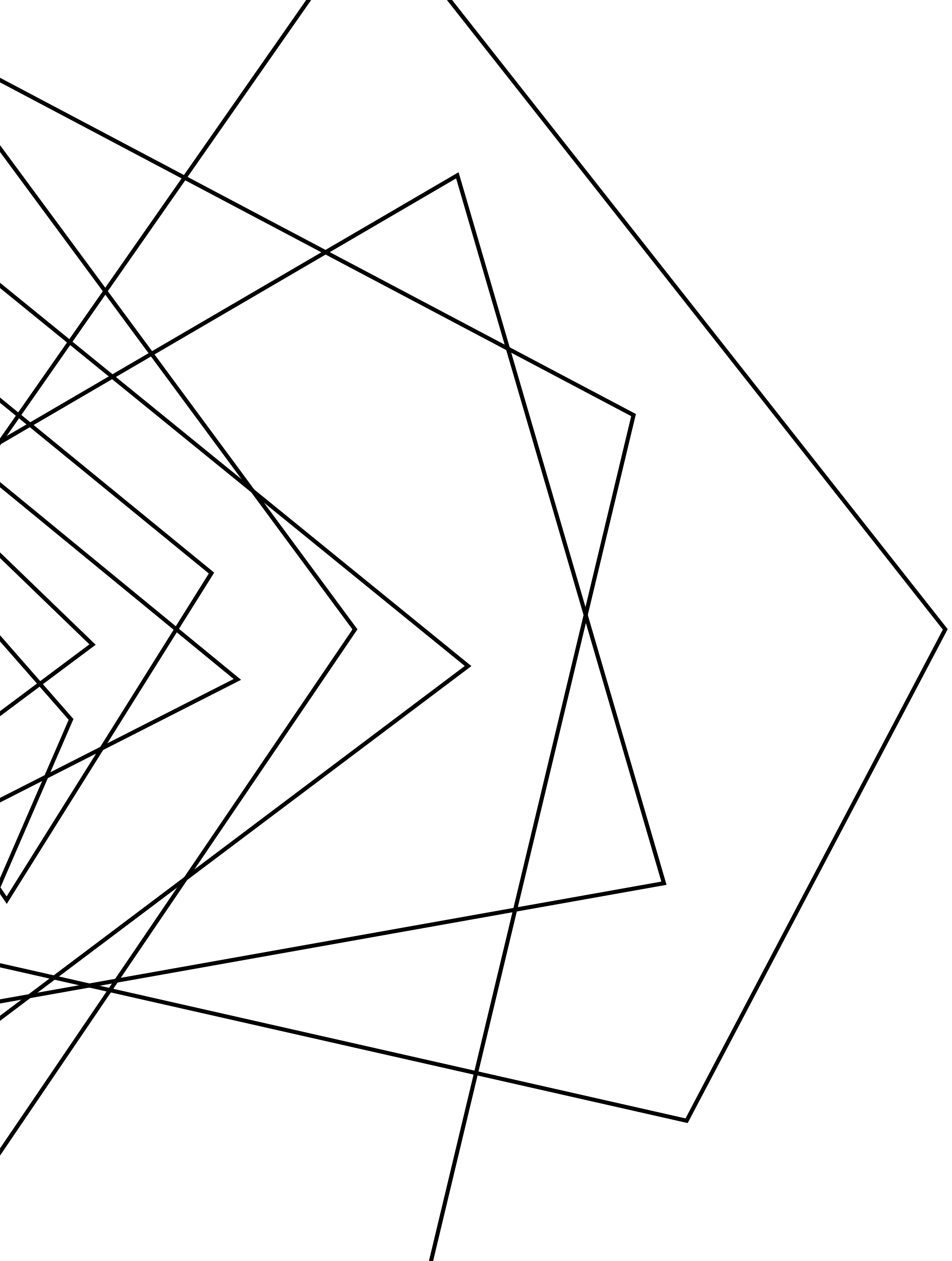
LAST TIME: SSDLC

REVIEW: LAST LECTURE

Corresponding Security Tasks for the Software Development Lifecycle

Requirement Analysis – Risk Assessment and Threat models
Design – Security Design Review
Development – Automated Code Analysis
Testing – Security Testing and Code Review
Maintenance and Evolution – Security Assessment and Configuration


1) Requirements Analysis
2) Design
3) Development
4) Testing
5) Release
6) Maintenance
Risk Assessment
Threat Modeling and Design Review
Static Analysis
Security Testing and Code Review
Security Assessment and Secure Configuration
Operational Assurance
Secure Software Development Life Cycle



CLASS PROGRESS

HANDLING THE “SOFTER SIDE” OF SECURITY EVALUATION

We’ve described some of the high-level best-practices, let’s talk about tool support

LECTURE OUTLINE

- Background / Context
- Linting
- Anti-Patterns
- Splint

SAD FACT: IT’S EASY TO WRITE INSECURE CODE

LINTING: BACKGROUND/CONTEXT

MANY PROGRAMMING LANGUAGES HAVE EXPLOITABLE CONSTRUCTS

Programming constructs that do not operate as intended under unforeseen circumstances

Artistic depiction of C programming

RECALL: SECURITY V USABILITY

LINTING OVERVIEW

MAINSTREAM PL PHILOSOPHY PRIORITIZES SPEED AND SIMPLICITY

C *could* do more checking, but it doesn't

- Bounds checking
- Type safety

"fat pointer"

RECALL: SECURITY V USABILITY

LINTING OVERVIEW

Expectations of efficiency and performance are hard to quit!

Disallowing unsafe behavior means going back on what's already been accomplished
- Rewrite legacy code
- Give up on some performance

CASE STUDY: MELTDOWN AND SPECTRE

LINTING OVERVIEW

The problem: Branch Predictors and speculative Execution

Impact: leaking secrets

The Solution: Mediate Speculative Execution

Early Fix performance:
OS Bench:
Intel Xeon 84~87%
AMD EPYC 91~94%.

RECALL: SECURITY V USABILITY

LINTING OVERVIEW

Waiting for better tools

Some feel that the whole of imperative programming is inherently unsafe

LECTURE OUTLINE

- Background / Context
- Linting
- Anti-Patterns
- Splint

HEURISTIC TOOLS FOR AN IMPERFECT WORLD

LINTING: OVERVIEW

TRY NOT TO SHOOT YOURSELF IN THE FOOT

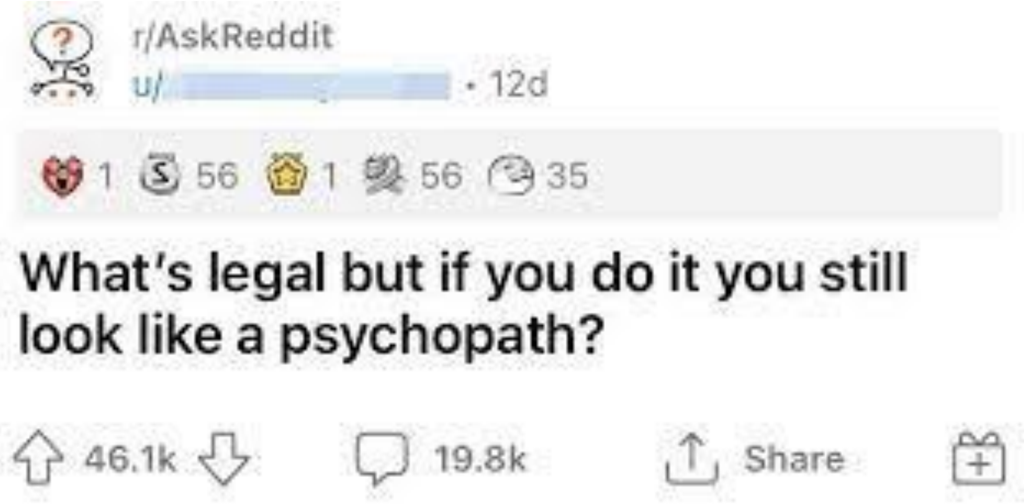
Highlight the stuff you probably shouldn’t be doing in the first place

CATCH "ANTI-PATTERNS"

HUMAN FACTORS OF SECURITY

COMMON LANGUAGE-LEGAL PAIN-POINTS

Code that is highly situational, or simply shouldn't be legal in hindsight



r/AskReddit
u/[redacted] · 12d

What's legal but if you do it you still look like a psychopath?

HISTORY: JOHNSON, 1978

HUMAN FACTORS OF SECURITY

Created a program called "lint"

Aided in the development of YACC

Originally internal to Bell Labs, eventually open-sourced

Name Inspired by Dryer Lint Traps

Capture the "loose fibers" that come off the program

Leave the whole of the program intact

PRODUCTION LINTERS

LINTING

More Modern tools

cppcheck – open-source linter

cpplint – Google's in-house (open-source) linter

flake8 – python linter

Also ensures adherence to style guide:
https://google.github.io/styleguide/cppguide.html

Good reminder that coding is still a human process

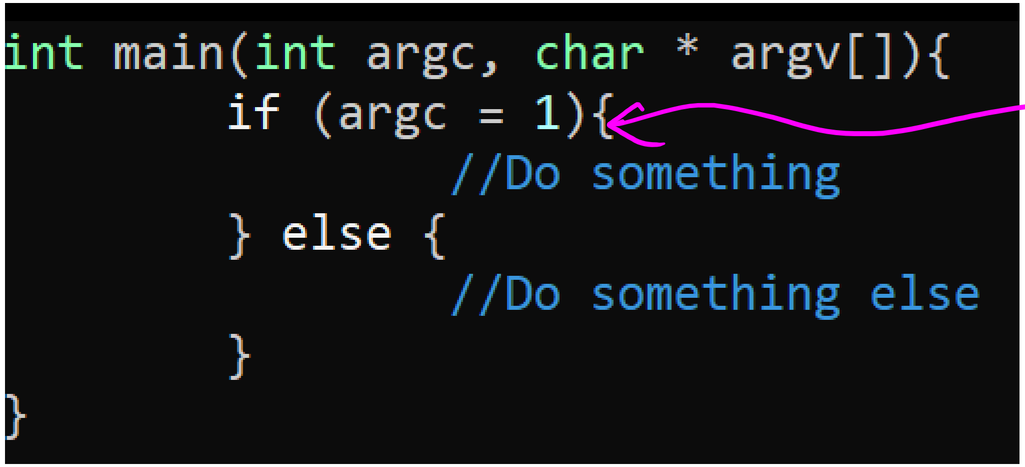
LECTURE OUTLINE

- Background / Context
- Linting
- Anti-Patterns
- Splint

Secure programming
lint

ASSIGNMENT IN PREDICATE

LINTING



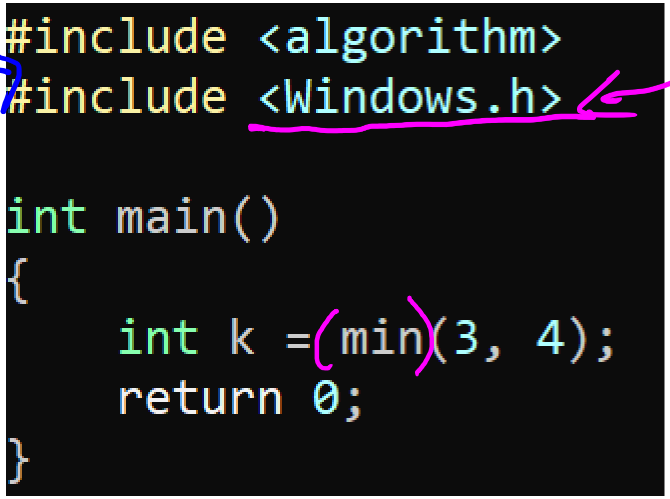
```
int main(int argc, char * argv[]){
        if (argc = 1){
                //Do something
        } else {
                //Do something else
        }
}
```

if (true) {
argc = 1;

MACRO POLLUTION

LINTING

#define no-min_max



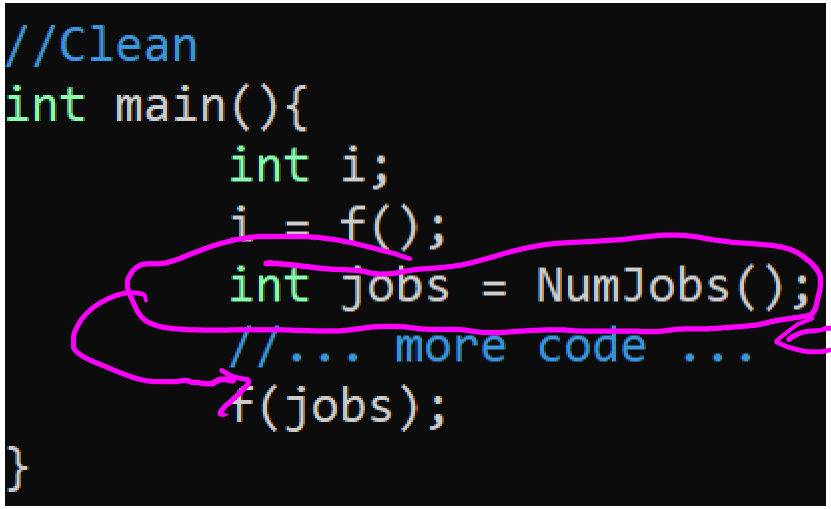
```
#include <algorithm>
#include <Windows.h>

int main()
{
    int k = min(3, 4);
    return 0;
}
```

macro-define
min
max

SEPARATING INITIALIZATION FROM USE

LINTING

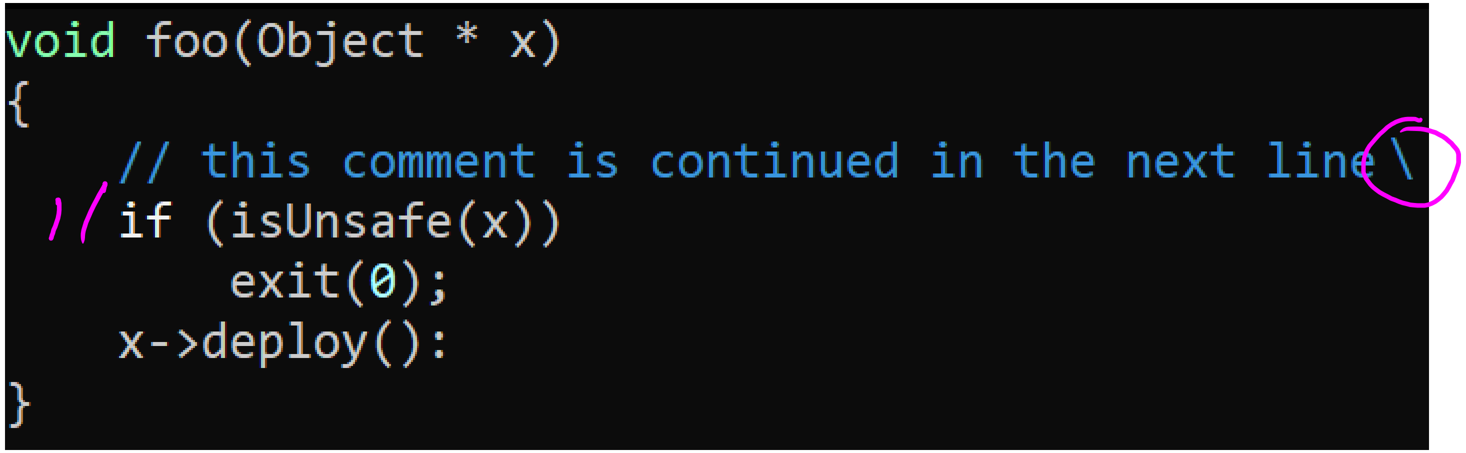


```
//Clean
int main(){
        int i;
        i = f();
        int jobs = NumJobs();
        //... more code ...
        f(jobs);
}
```

that does not use the jobs variable

SEPARATING INITIALIZATION FROM USE

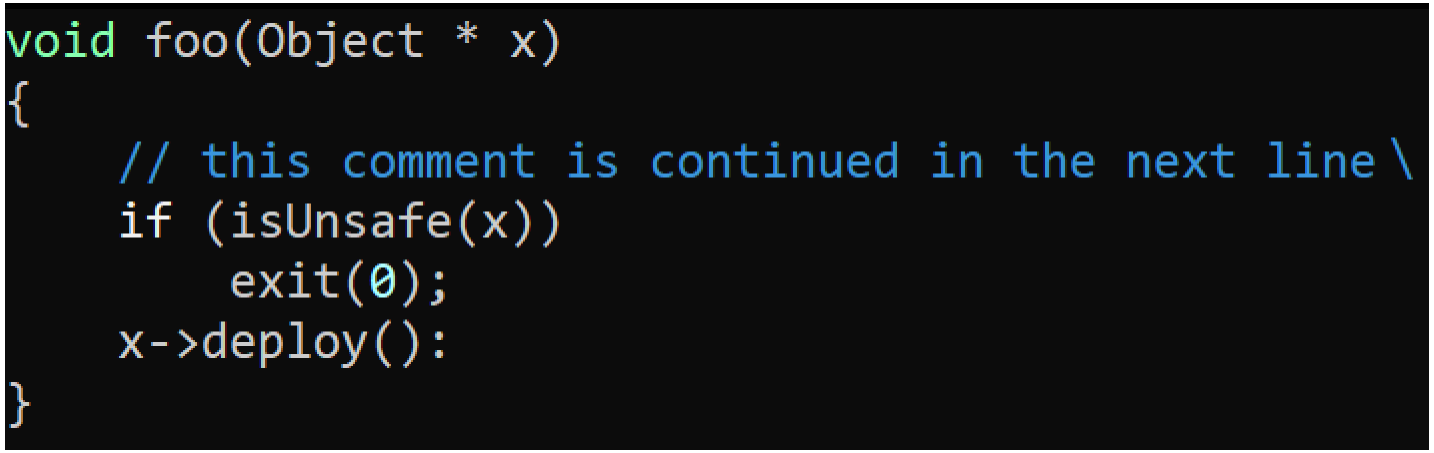
LINTING



```
void foo(Object * x)
{
    // this comment is continued in the next line \
  //if (isUnsafe(x))
        exit(0);
    x->deploy():
}
```

LINE CONTINUATION WEIRDNESS

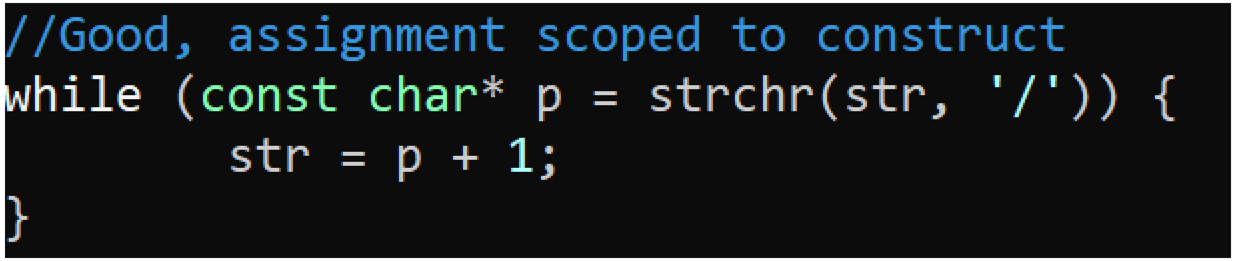
LINTING



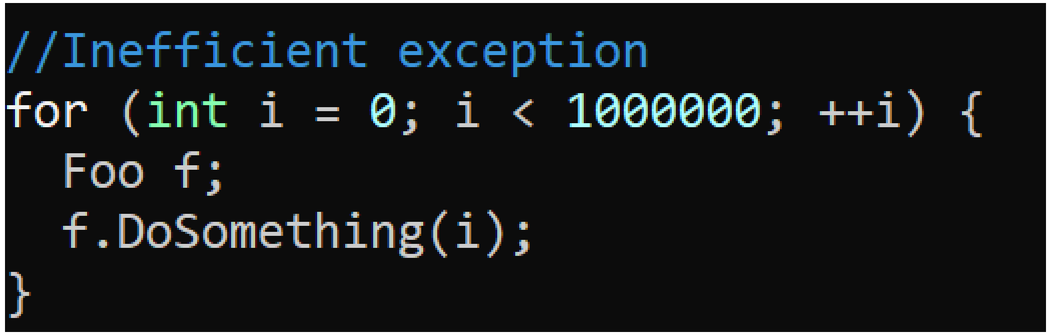
```
void foo(Object * x)
{
    // this comment is continued in the next line\
    if (isUnsafe(x))
        exit(0);
    x->deploy():
}
```

SCOPED INITIALIZATION

LINTING



```
//Good, assignment scoped to construct
while (const char* p = strchr(str, '/')) {
        str = p + 1;
}
```



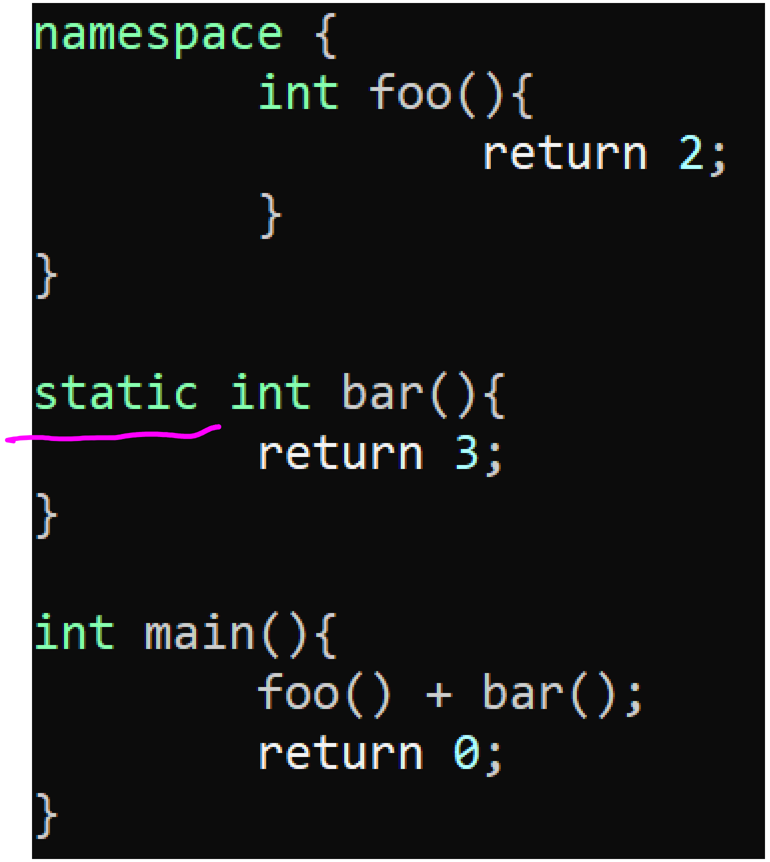
```
//Inefficient exception
for (int i = 0; i < 1000000; ++i) {
  Foo f;
  f.DoSomething(i);
}
```

```
{
Foo f;
for (int i=0;...){
    f.doSomething();
  }
}
```

NAMESPACING (GOOD)

LINTING

not visible outside file



```
namespace {
        int foo(){
                return 2;
        }
}

static int bar(){
        return 3;
}

int main(){
        foo() + bar();
        return 0;
}
```

HEURISTIC TOOLS FOR AN IMPERFECT WORLD

LINTING: OVERVIEW

TRY NOT TO SHOOT YOURSELF IN THE FOOT

Highlight the stuff you probably shouldn’t be doing in the first place

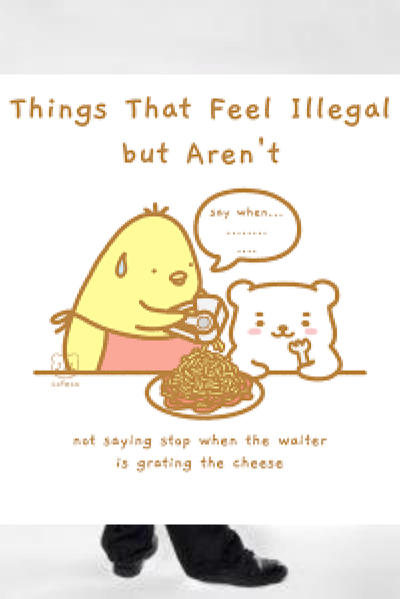
RECALL: SECURITY V USABILITY

LINTING OVERVIEW

MANY PROGRAMMING LANGUAGES HAVE EXPLOITABLE CONSTRUCTS

Capture the “loose fibers” that come off t program

Leave the whole of the program intact


Things That Feel Illegal but Aren't
say when... ....
not saying stop when the waiter is grating the cheese

r/AskReddit
u/[redacted] • 12d
56 1 56 35
t's legal but if you do it you still like a psychopath?
6.1k 19.8k Share