OS

Operating System Laboratory

Lecture 2: Kernel Static Analysis

Yong XIANG
Dept. of Computer Sci. & Tech., Tsinghua Univ.

Outline

- Background
  - LINT: A Programming Tool
  - The Problem
  - Compiler
- Static Analysis
  - Concept
  - Buffer Overrun
  - Pitfalls of Privileges
  - Untrusted Data
  - A Perfect Work

DOUBLE-CHECKING PROGRAMS: LINT

- Checks your program more thoroughly than cc does:

Utility : **lint** { fileName }*

lint scans the specified source files and displays any potential errors that it finds.

```
$ lint  reverse.c                          ---> check "reverse.c".
 reverse  defined ( reverse.c(12) ), but never used
```

```
$ lint palindrome.c                        ---> check "palindrome.c".
palindrome  defined ( palindrome.c ( 12 ) ), but never used
reverse used ( palindrome.c(14) ), but not defined
```

DOUBLE-CHECKING PROGRAMS: LINT

```
$ lint  main2.c                                  ---> check "main2.c".
main2.c(11) : warning: main() returns random value to invocation
environment
printf returns value which is always ignored
palindrome used ( main2.c(9) ), but not defined
$ lint  main2.c  reverse.c  palindrome.c ---> check all modules together.
main2.c:
min2.c(11): warning: main() returns random value to invocation
environment
reverse.c:
palindrome.c:
Lint pass2:
printf returns value which is always ignored
$ _
```

Outline

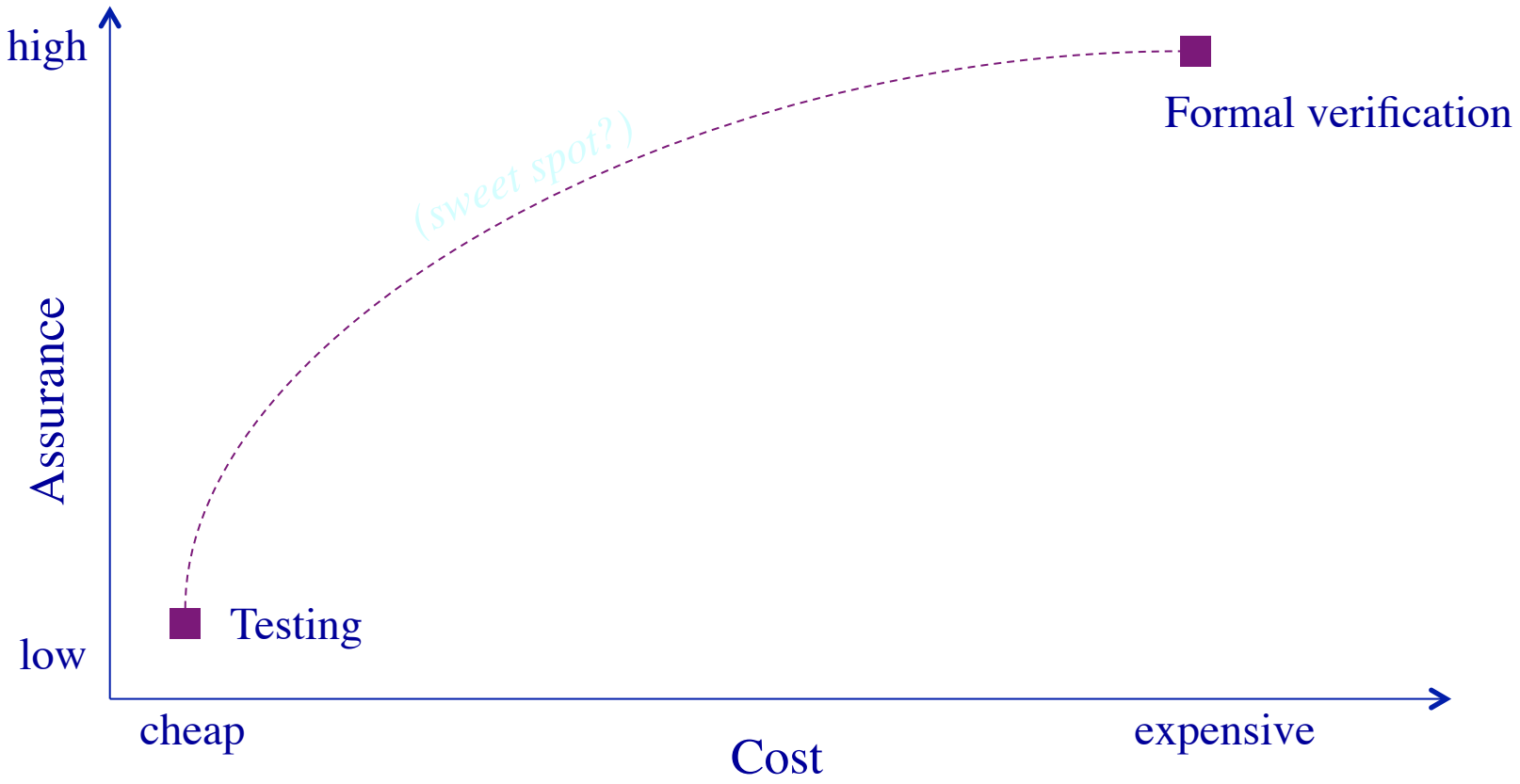
- Background
  - LINT: A Programming Tool
  - The Problem
  - Compiler
- Static Analysis
  - Concept
  - Buffer Overrun
  - Pitfalls of Privileges
  - Untrusted Data
  - A Perfect Work

The Problem

- Building secure systems is hard
  - 2/3 of Internet servers have gaping security holes
- The problem is buggy software
  - And a few pitfalls account for many vulnerabilities

- Challenge: Improve programming technology
  - Need way to gain *assurance* in our software
  - Static analysis can help!

Existing Paradigms



What Makes Security Hard?

- Security is hard because of…
  - buffer overruns
  - privilege pitfalls
  - untrusted data
  - …

Outline

- Background
  - LINT: A Programming Tool
  - The Problem
  - Compiler
- Static Analysis
  - Concept
  - Buffer Overrun
  - Pitfalls of Privileges
  - Untrusted Data
  - A Perfect Work

What is a compiler?

- A program that ***translates*** a program in one language to another language
  - The essential interface between applications & architectures
- Typically ***lowers*** the level of abstraction
  - analyzes and reasons about the program & architecture
- We expect the program to be ***optimized,*** i.e., better than the original
  - ideally exploiting architectural strengths and hiding weaknesses

Role of compilers

- Bridge complexity and evolution in architecture, languages, & applications
- Help programmers with correctness, reliability, program understanding
- Compiler optimizations can significantly improve performance
  - 1 to 10x on conventional processors
- Performance stability: one line change can dramatically alter performance
  - unfortunate, but true

Optimization

What should it do?

1. improve running time, or
2. decrease space requirements
3. decrease power consumption

How does it do it?

Example optimizations

Division of optimizations

1. Machine independent
2. Machine dependent

Faster code optimizations

- common subexpression elimination
- constant folding
- dead code elimination
- register allocation
- scheduling
- loop transformations

Analysis

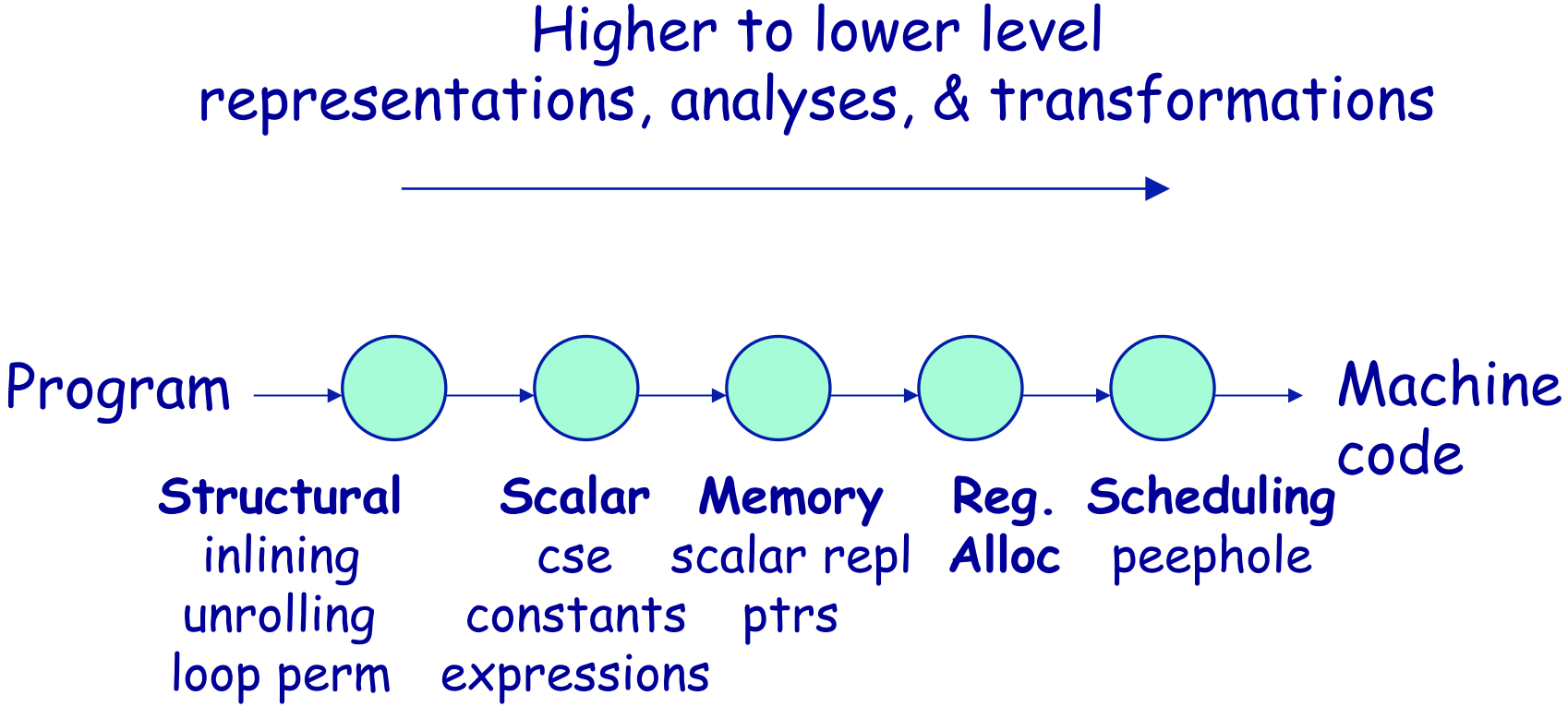
Scope of program analysis

1. within a basic block (local)
2. within a method (global)
3. across methods (interprocedural)

Analysis

1. control flow graph - dominators, loops, etc.
2. dataflow analysis - flow of values
3. static-single-assignment – transform programs such that each variable has a unique definition
4. alias analysis - pointer memory usage
5. dependence analysis - array memory usage

Basic Compiler Structure



Outline

- Background
  - LINT: A Programming Tool
  - The Problem
  - Compiler
- Static Analysis
  - Concept
  - Buffer Overrun
  - Pitfalls of Privileges
  - Untrusted Data
  - A Perfect Work

Static Analysis - Concept

- Examples: compiler optimizations, program verifiers

- Examine program text (no execution)
- Build a model of program state
  - An abstraction of the run-time state
- Reason over possible behaviors
  - E.g., "run" the program over the abstract state

OS

Abstract interpretation

- Typically implemented via dataflow analysis
- Each program statement's transfer function indicates how it transforms state
- Example: What is the transfer function for
- `y = x++;`
- ?

Selecting an abstract domain

⟨ x is odd; y is odd ⟩

```
y = x++;
```

⟨ x is even; y is odd ⟩

⟨ x = { 3, 5, 7 }; y = { 9, 11, 13 } ⟩

```
y = x++;
```

⟨ x = { 4, 6, 8 }; y = { 3, 5, 7 } ⟩

⟨ x is prime; y is prime ⟩

```
y = x++;
```

⟨ x is anything; y is prime ⟩

⟨x=3, y=11⟩, ⟨x=5, y=9⟩, ⟨x=7, y=13⟩

```
y = x++;
```

⟨x=4, y=3⟩, ⟨x=6, y=5⟩, ⟨x=8, y=7⟩

⟨ $x_n = f(a_{n-1},\ldots,z_{n-1})$; $y_n = f(a_{n-1},\ldots,z_{n-1})$ ⟩

```
y = x++;
```

⟨ $x_{n+1} = x_n+1$; $y_{n+1} = x_n$ ⟩

OS

Research challenge: Choose good abstractions

- The abstraction determines the expense (in time and space)
- The abstraction determines the accuracy (what information is lost)
  - Less accurate results are poor for applications that require precision
  - Cannot conclude all true properties in the grammar

Static analysis: Characteristic

- Slow to analyze large models of state, so use abstraction
- Conservative: account for abstracted-away state
- Sound: (weak) properties are guaranteed to be true
  - *Some static analyses are not sound

Outline

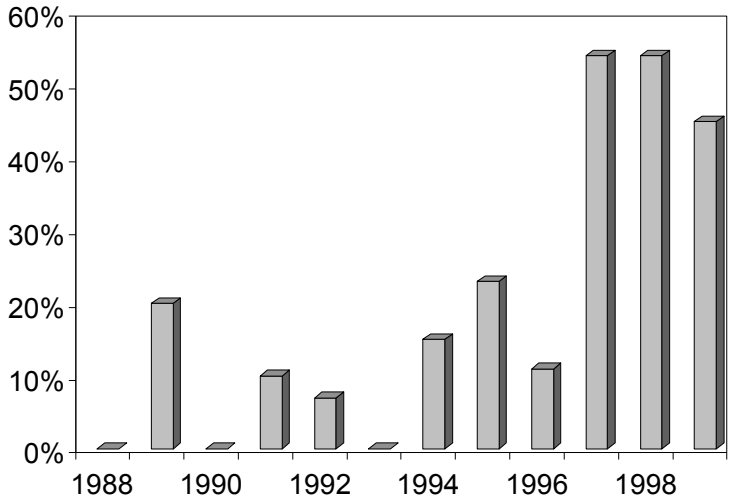
- Background
  - LINT: A Programming Tool
  - The Problem
  - Compiler
- Static Analysis
  - Concept
  - Buffer Overrun
  - Pitfalls of Privileges
  - Untrusted Data
  - A Perfect Work

Buffer Overruns

- An example bug:

```
char buf[80];
hp = gethostbyaddr(...);
strcpy(buf, hp->hp_hname);
```

- Accounts for 50% of recent vulnerabilities



Percentage of CERT advisories due to buffer overruns each year

Static Detection of Overruns

- Introduce implicit variables:
  - alloc(buf) = # bytes allocated for buf
  - len(buf) = # bytes stored in buf
  - Safety condition: len(buf) ≤ alloc(buf)

OS

Current Status

- Experimental results
  - Found new bugs in sendmail (30k LOC), others
  - Analysis is fast, but many false alarms (1/kLOC)

see also Dor, Rodeh, Sagiv

- Research challenges
  - Pointer analysis (support strong updates)
  - Integer analysis (infer linear relations, flow-sensitivity)
  - Soundness, scalability, real-world programs

Outline

- Background
  - LINT: A Programming Tool
  - The Problem
  - Compiler
- Static Analysis
  - Concept
  - Buffer Overrun
  - Pitfalls of Privileges
  - Untrusted Data
  - A Perfect Work

Pitfalls of Privileges

- Spot the bug:

```
setuid(0);
rv = bind(...);
if (rv < 0)
    return rv;
seteuid(getuid());
```

enablePriv()

checkPriv()

Bug! Leaks privilege

disablePriv()

A Common Language

- Various interpretations are possible
  - C: enablePriv(p) lasts until next disablePriv(p)
  - Java: … or until containing stack frame is popped
  - checkPriv(p) throws fatal error if p not enabled

Static Privilege Analysis

- Some problems in privilege analysis:
  - *Privilege inference* (auditing, bug-finding)
    - Find all privileges reaching a given program point
  - *Enforcing privilege-safety* (cleanliness of new code)
    - Verify statically that no checkPriv() operation can fail
    - … or that program behaves same under C & Java styles

Future Directions

- Research challenges
  - Experimental studies on real programs
  - Handling data-directed privilege properties
  - Other access control models

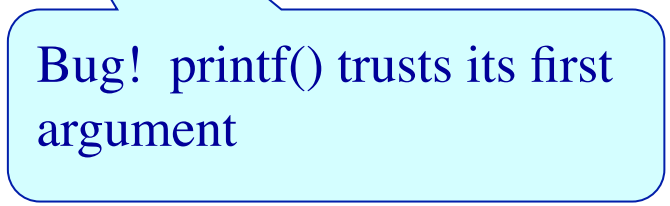
Outline

- Background
  - LINT: A Programming Tool
  - The Problem
  - Compiler
- Static Analysis
  - Concept
  - Buffer Overrun
  - Pitfalls of Privileges
  - Untrusted Data
  - A Perfect Work

Manipulating Untrusted Data

- Spot the bug:

```
hp = gethostbyaddr(...);
printf(hp->hp_hname);
```



Trust Analysis

- Security involves much mental "bookkeeping"
  - Problem: Help programmer keep track of which values can be trusted
- One approach: static *taint analysis*
  - Extend the C type system
  - Qualified types express annotations: e.g., `tainted char *` is an untrusted string
  - Typechecking enforces safe usage
  - Type inference reduces annotation burden

A Tiny Example

a trust annotation

```
void printf(untainted char *, ...);
tainted char * read_from_network(void);
```

```
char *s = read_from_network();
printf(s);
```

… where untainted T ≤ tainted T

After Type Inference…

```
void printf(untainted char *, ...);
tainted char * read_from_network(void);
```

an inferred type

```
tainted  char *s = read_from_network();
printf(s);
```

Doesn't type-check!
Indicates vulnerability

… where untainted T ≤ tainted T

Current Status

- Experimental results
  - Successful on real programs
    - Able to find many previously-known format string bugs
    - Cost: 10-15 minutes per application
  - Type theory seems useful for security engineering
- Research challenges
  - Richer theory to support real programming idioms
  - More broadly-applicable discipline of good coding
  - Finer-grained notions of trust

Concluding Remarks

- Static analysis can help secure our software
  - Buffer overruns, privilege bugs, format string bugs
  - Hits a sweet spot: cheap and proactive
- Security as a source of interesting problems?
  - Motivations for better pointer, integer analysis
  - New problems: privilege analysis, trust analysis

Outline

- Background
  - LINT: A Programming Tool
  - The Problem
  - Compiler
- Static Analysis
  - Concept
  - Buffer Overrun
  - Pitfalls of Privileges
  - Untrusted Data
  - A Perfect Work

A Perfect Example

- Improving Integer Security for Systems with KINT
- Towards Optimization-Safe Systems: Analyzing the Impact of Undefined Behavior

OS

References

- Lint, a C Program Checker
  - http://files.cnblogs.com/bangerlee/10.1.1.56.1841.pdf
- Towards Optimization-Safe Systems: Analyzing the Impact of Undefined Behavior
  - http://adl.csie.ncu.edu.tw/adlab/ppt/553_Towards%20Optimization-Safe%20Systems%20%E2%80%94%20Analyzing%20the%20Impact%20of%20Undefined%20Behavior.pptx
- Improving Integer Security for Systems with KINT
  - http://pdos.csail.mit.edu/~xi/papers/kint-osdi12-slides.pptx
- Static Analysis and Software Assurance
  - http://www.cs.berkeley.edu/~daw/talks/sas01.ppt
- Static and dynamic analysis: synergy and duality
  - http://courses.cs.washington.edu/courses/cse503/10wi/lectures/lecture1-static-dynamic.ppt
- compiler
  - http://www.cs.utexas.edu/users/mckinley/380C/lecs/01.ppt
- Static/Dynamic Analysis Tools
  - http://www.csl.sri.com/users/shankar/VGC05/Aiken.ppt
- C Programming Tools
  - http://www.csun.edu/~andrzej/COMP421/lectures/tools.ppt

Q&A