Type Qualifiers:

Lightweight Specifications to Improve Software Quality

CMSC 631, Fall 2003

Software Quality Today

Even after large, extensive testing efforts, commercial software is shipped riddled with errors ("bugs").

-- PITAC Report to the President, February 24, 1999

Trustworthy Computing is computing that is available, reliable, and secure as electricity, water services and telephony....No Trustworthy Computing platform exists today.

-- Bill Gates, January 15, 2002
(highest priority for Microsoft)

Conclusion?

Software is buggy

So What?

- Software has always been buggy
- But now...
  - More people use software
  - Computers keep getting faster
    - Speed/quality tradeoff changing
  - Cost of fixing bugs is high

Common Techniques for Software Quality

- Testing
- Code auditing
- Drawbacks: Expensive, difficult, error-prone, limited assurances
- What more can we do?
  - Tools that analyze source code
  - Techniques for avoiding programming mistakes

Tools Need Specifications

```
spin_lock_irqsave(&tty->read_lock, flags);
put_tty_queue_nolock(c, tty);
spin_unlock_irqrestore(&tty->read_lock, flags);
```

- Goal: Add specifications to programs

  In a way that...
  - Programmers will accept
    - Lightweight
  - Scales to large programs
  - Solves many different problems

Type Qualifiers

- Extend standard type systems (C, Java, ML)
  - Programmers already use types
  - Programmers understand types
  - Get programmers to write down a little more...

const int	ANSI C
ptr(tainted char)	Security vulnerabilities
int → ptr(open FILE)	File operations

Application: Format String Vulnerabilities

- I/O functions in C use format strings

  printf("Hello!"); Hello!

  printf("Hello, %s!", name); Hello, *name* !

- Instead of

  printf("%s", name);

  Why not

  printf(name); ?

Format String Attacks

- Adversary-controlled format specifier

  name := <data-from-network>
  printf(name); /* Oops */

  - Attacker sets name = "%s%s%s" to crash program
  - Attacker sets name = "...%n..." to write to memory

- Lots of these bugs in the wild
  - New ones weekly on bugtraq mailing list
  - Too restrictive to forbid variable format strings

Using Tainted and Untainted

- Add qualifier annotations

  int printf(untainted char *fmt, ...)
  tainted char *getenv(const char *)

tainted = may be controlled by adversary

untainted = must not be controlled by adversary

Subtyping

```
void f(tainted int);
untainted int a;
f(a);
```

OK

f accepts tainted or untainted data

untainted ≤ tainted

```
void g(untainted int);
tainted int b;
f(b);
```

Error

g accepts only untainted data

tainted ≰ untainted

untainted < tainted

Demo of cqual

http://www.cs.umd.edu/~jfoster

Framework

- Pick some qualifiers
  - and relation (partial order) among qualifiers

    untainted int < tainted int

    readwrite FILE < read FILE
- Add a few explicit qualifiers to program
- Infer remaining qualifiers
  - and check consistency

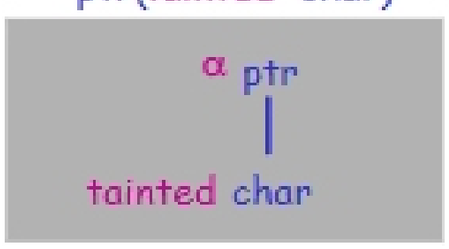
Type Qualifier Inference

- Two kinds of qualifiers
  - Explicit qualifiers: tainted, untainted, ...
  - Unknown qualifiers: $\alpha_0, \alpha_1, \ldots$
- Program yields constraints on qualifiers

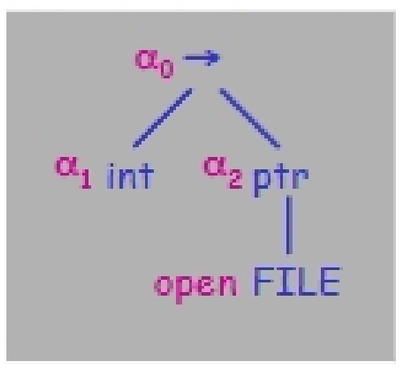
  tainted $\leq \alpha_0$ $\alpha_0 \leq$ untainted
- Solve constraints for unknown qualifiers
  - Error if no solution

Adding Qualifiers to Types

ptr(tainted char)

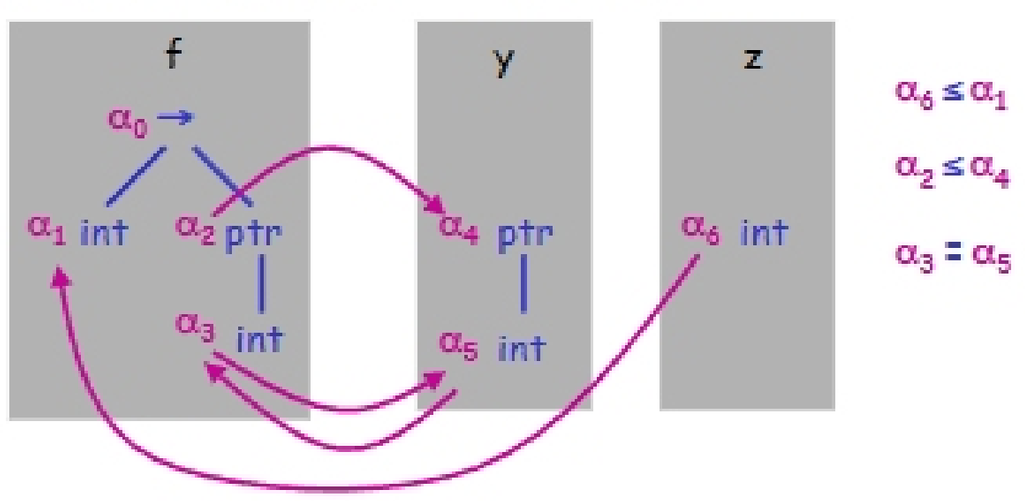


int → ptr(open FILE)



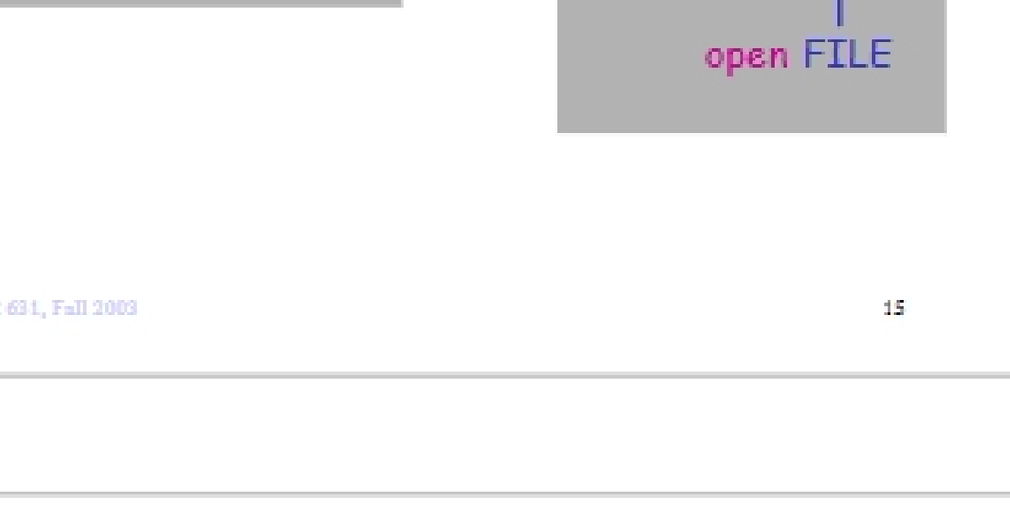
Constraint Generation

ptr(int) f(x : int) = { ... } y := f(z)



Constraints as Graphs

Key idea: programs → constraints → graphs



Satisfiability via Graph Reachability

Is there an inconsistent path through the graph?