18739A: Foundations of Security and Privacy

Language-based Security Overview

Anupam Datta
CMU

Fall 2007-08

Software System Security

- Goal:
  - Ensure security of software systems

- Definition of security
  - What does “secure” mean?
  - Our focus: non-interference
- Methods for ensuring that software systems satisfy security definition
  - Our focus: Typed programming languages

Popular Programming languages (I)

- C
  - Security vulnerabilities: buffer overflows, format string vulnerabilities, etc.
  - What features of C cause these problems?
  - Cyclone: A safe dialect of C [Morissett et al]

TODAY

Popular Programming languages (II)

- Java
  - Does not suffer from C-style security vulnerabilities: buffer overflows, format string vulnerabilities, etc.
  - Perfectly good Java program: login program sends password in the clear over the network

    (More subtle attack: Boneh-Brumley timing attack on SSL implementation – not Java)

  - Which programs are “secure”?

Information flow

- Security definition
  - Non-interference [Goguen-Meseguer82]
- Embodiment in a programming language
  - [Denning-Denning77]
  - [Volpano-Smith-Irvine96]
- Extending Java with information flow control
  - Jif [Myers-Liskov97]

TODAY

2^{nd} lecture

3^{rd} lecture

Main tool

- Typed programming languages

- Now: A quick overview
- Relevant CMU courses
  - 15-814: Type Systems for Programming Languages
  - 15-819: Languages and Logics for Security

Programming Language Definition

- Syntax (or “well-formed programs”)
  - Syntax of types and terms
  - Type system (or static semantics)

- Semantics (or “meaning of programs”)
  - Operational (Dynamic)
    - Program execution

Language Definition Examples

- Syntax, Semantics (Static, Dynamic)
- ML:
  - R. Milner, M. Tofte, R. Harper, and D. MacQueen, *The Definition of Standard ML (Revised).* MIT Press, 1997
- Java:
  - J. Alves-Foss (Ed.), *Formal Syntax and Semantics of Java*. LNCS 1523, 1999

Simple Expression Language

- Syntax

$$
\begin{array}{llll}
\textit{Types} & \tau & ::= & \texttt{nat} \mid \texttt{str} \\
\textit{Terms} & e & ::= & x \mid \texttt{num}[n] \mid \texttt{plus}(e_1, e_2) \mid \texttt{str}[s] \mid \texttt{cat}(e_1, e_2) \mid \texttt{let}(e_1, x.e_2)
\end{array}
$$

- Static Semantics (Type System)

$$\frac{}{x : \tau \vdash_{x} x : \tau}$$

$$\frac{}{\texttt{str}[s] : \texttt{str}}$$

$$\frac{}{\texttt{num}[n] : \texttt{nat}}$$

$$\frac{e_1 : \texttt{nat} \quad e_2 : \texttt{nat}}{\texttt{plus}(e_1, e_2) : \texttt{nat}}$$

$$\frac{e_1 : \texttt{str} \quad e_2 : \texttt{str}}{\texttt{cat}(e_1, e_2) : \texttt{str}}$$

$$\frac{e_1 : \tau_1 \quad x : \tau_1 \vdash_{\{x\}} e_2 : \tau_2}{\texttt{let}(e_1, x.e_2) : \tau_2}$$

Simple Expression Language

- **Semantics**
  - Operational/Dynamic

$$\frac{}{\texttt{num}[n]\ \mathsf{val}}$$

$$\frac{}{\texttt{str}[s]\ \mathsf{val}}$$

$$\frac{(p = m + n)}{\texttt{plus}(\texttt{num}[m], \texttt{num}[n]) \mapsto \texttt{num}[p]}$$

$$\frac{(u = s\,\hat{}\,t)}{\texttt{cat}(\texttt{str}[s], \texttt{str}[t]) \mapsto \texttt{str}[u]}$$

$$\frac{e_1\ \mathsf{val}}{\texttt{let}(e_1, x.e_2) \mapsto [e_1/x]e_2}$$

$$\frac{e_1 \mapsto e_1'}{\texttt{plus}(e_1, e_2) \mapsto \texttt{plus}(e_1', e_2)}$$

$$\frac{e_1\ \mathsf{val} \quad e_2 \mapsto e_2'}{\texttt{plus}(e_1, e_2) \mapsto \texttt{plus}(e_1, e_2')}$$

$$\frac{e_1 \mapsto e_1'}{\texttt{cat}(e_1, e_2) \mapsto \texttt{cat}(e_1', e_2)}$$

$$\frac{e_1\ \mathsf{val} \quad e_2 \mapsto e_2'}{\texttt{cat}(e_1, e_2) \mapsto \texttt{cat}(e_1, e_2')}$$

$$\frac{e_1 \mapsto e_1'}{\texttt{let}(e_1, x.e_2) \mapsto \texttt{let}(e_1', x.e_2)}$$

Type Safety

Theorem 12.1 (Type Safety). 1. *If* $e : \tau$ *and* $e \mapsto e'$, *then* $e' : \tau$.

2. *If* $e : \tau$, *then either* e val, *or there exists* e' *such that* $e \mapsto e'$.

1. *Preservation:* Evaluation steps preserve types
2. *Progress:* Well-typed expressions are either values or can be further evaluated

Relates static semantics to dynamic semantics

Type Systems Features

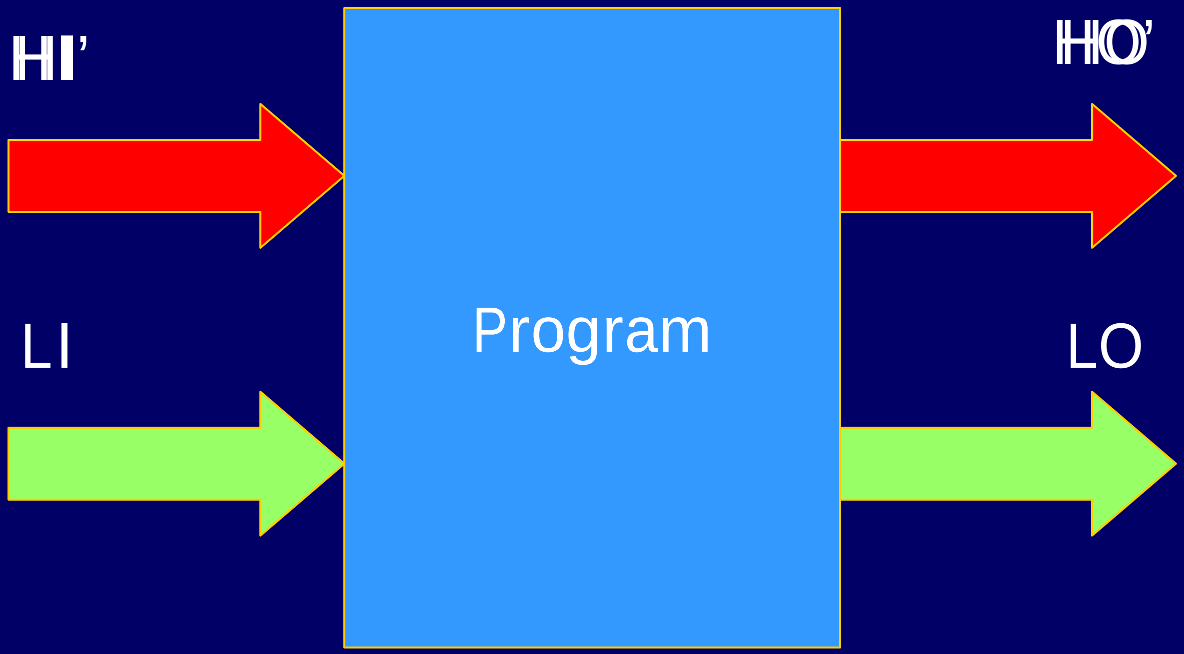
- Tools for reasoning about *programs*
- *Classification* of terms into types provides static approximation of run-time behavior of the terms in a program
- *Conservative*: Can prove the absence of some bad program behaviors, but not their presence
- *Tractable*: Automated typecheckers typically built into compiler or linker

Cyclone

- Presented by Kumar Avijit

Definition of Security

- Non-interference (idea)



No information flows from high inputs to low outputs

Formal definition

A system S is said to be non-interfering from High to Low iff:

$$\forall\, tr \in I^*, c \in I \bullet Output_L(S, tr, c) = Output_L(S, purge_{HI}(tr), c)$$

- System is deterministic finite state machine: takes input and transitions to next state producing output
- Trace *tr* is a sequence of inputs and outputs (high & low)
- $Output_L(S,tr,c)$: low output of system *S* when input *c* is applied to the state corresponding to trace *tr*
- $purge_{HI}(tr)$: returns a trace with all high inputs in *tr* removed

Thanks

Questions?