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]
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]
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:

 Java:
  - J. Alves-Foss (Ed.), Formal Syntax and Semantics of Java. LNCS 1523, 1999
Simple Expression Language

◆ Syntax

\[
\begin{align*}
\text{Types} & \quad \tau ::= \text{nat} | \text{str} \\
\text{Terms} & \quad e ::= x | \text{num}[n] | \text{plus}(e_1, e_2) | \text{str}[s] | \text{cat}(e_1, e_2) | \text{let}(e_1, x, e_2)
\end{align*}
\]

◆ Static Semantics
(Type System)

\[
\begin{align*}
\frac{x : \tau \vdash x : \tau}{\text{num}[n] : \text{nat}} \\
\frac{\text{str}[s] : \text{str}}{\text{cat}(e_1, e_2) : \text{str}} \\
\frac{e_1 : \text{nat} \quad e_2 : \text{nat}}{\text{plus}(e_1, e_2) : \text{nat}} \\
\frac{e_1 : \text{str} \quad e_2 : \text{str}}{\text{let}(e_1, x, e_2) : \tau_2}
\end{align*}
\]
Simple Expression Language

◆ Semantics
  • Operational/Dynamic

\[\begin{array}{c}
\text{num}[n] \rightarrow \text{val} \\
\hline
\text{str}[s] \rightarrow \text{val}
\end{array}\]

\[\begin{array}{c}
(p = m + n) \\
\hline
\text{plus}(\text{num}[m], \text{num}[n]) \rightarrow \text{num}[p]
\end{array}\]

\[\begin{array}{c}
(u = s \cdot t) \\
\hline
\text{cat}(\text{str}[s], \text{str}[t]) \rightarrow \text{str}[u]
\end{array}\]

\[\begin{array}{c}
e_1 \rightarrow e'_1 \\
\hline
\text{let}(e_1, x, e_2) \rightarrow [e_1/x]e_2
\end{array}\]
Type Safety

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

Theorem 12.1 (Type Safety).

1. If $e : \tau$ and $e \rightarrow e'$, then $e' : \tau$.
2. If $e : \tau$, then either $e \text{ val}$, or there exists $e'$ such that $e \rightarrow e'$. 
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:

$$\forall \text{tr} \in I^*, c \in I \cdot \text{Output}_L(S, \text{tr}, c) = \text{Output}_L(S, \text{purge}_{HI}(\text{tr}), c)$$

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

Questions?