Information Flow Security

Advanced Programming Languages

A.A. Kabiri Zamani

Overview

1

Secure System Definition

2

Information Flow

3

Flow Proofs

4

A Sound Type System for Flow Security

5

Type Soundness

“

It would be awesome to consume what you learned theoretically, practically.”

1 Secure System Definition

G.Andrews, R.Reitman, “An Axiomatic Approach to Information Flow in Programs”, ACM Transactions on Programming Languages and Systems 2, 1, (1980), 56–76.

1. Secure System Definition

- A computer system is secure if the information it contains can only be manipulated in ways authorized by its security policies.

- Security policies:
  - Access policies: the rights that subjects, such as users, have to access objects that contain information, such as files
  - Information flow policies: the classes of information that can be contained in objects and the allowed relations between object classes.

- Validating flow policies

2 Information Flow

G.Andrews, R.Reitman, “An Axiomatic Approach to Information Flow in Programs”, ACM Transactions on Programming Languages and Systems, (1980), 56–76.

D.Denning, P.Denning, “Certification of Programs for Secure Information Flow”, Communications of the ACM, (1977), 504–513.

2. Information Flow

- Given a program, a flow policy specifies classifications of the variables in the program

- A flow policy is validated by:
    - Solving the information flow problem: Determining the ways in which program execution can cause the information in variables to depend on the *input* and *the information in other variables*
    - Certifying that no execution path can cause information flows that result in variable classifications that violate the policy

2. Information Flow

Basic Concepts

- A program's components with respect to information flow:
  - $Variables$: contain informations
  - $Information\ state$: the current classification
  - $Statements$: modify variables (and hence alter the state) or control the order of execution

Classification of Information

- To characterize the sensitivity of information, each variable has a $classification$
- The set of security classes is:
  - Finite
  - Partially ordered
  - Has a $Least\ Upper\ Bound$ $(LUB, \oplus)$

2. Information Flow

Conventions

- $\underline{x}$: class of variable x which can be determined statically and does not vary in runtime
- $\leq$: relationship between classes
- $x \leq y$: the class of y is at least as great as that of variable x
- $\oplus$: the means by which classes are combined
  - For example: if E is the expression $x * y$, then the class of E, denoted $\underline{E}$, is $\underline{x} \oplus \underline{y}$, since E contains information from both x and y.
- $\underline{low}$: the lowest class
- $\underline{high}$: the highest class

Classification Schemes

- Two specific classification schemes are used:
  - Classes are linearly ordered from $\underline{low}$ to $\underline{high}$; corresponds to military classification scheme where:
    - $\underline{low}$: $unclassified$
    - $\underline{high}$: $top\ secret$
  - A distinct class is associated with each variable:
    - constants have class $\underline{low}$
    - other classes consist of the $powerset$ of the variable classes and $\underline{low}$
    - $\oplus$ is the union operation
    - $\leq$ is set inclusion
    - $\underline{high}$ is the union of all the classes

2. Information Flow

Types of information flow

• $Assignment\colon\ x := \mathrm{E}$	$Direct$
• $Alternation\colon\ if\ \mathrm{B}\ then\ \mathrm{S}_1\ else\ \mathrm{S}_2$	$Indirect/local$
• $Iteration\colon\ while\ \mathrm{B}\ do\ \mathrm{S}$	$Indirect/local$
• $Iteration\colon\ x =: 1;$	$Indirect/Global$
$while\ y = 0\ do\ \text{“}skip\text{”};$	
$x := 0$	

- The classifications of variables change as a result of executing statements

3 Flow Proofs

G.Andrews, R.Reitman, “An Axiomatic Approach to Information Flow in Programs”, ACM Transactions on Programming Languages and Systems, (1980), 56–76.

D.Denning, P.Denning, “Certification of Programs for Secure Information Flow”, Communications of the ACM, (1977), 504–513.

3. Flow Proofs

- $\{x \leq y\}$: the set of states where the classification of x is no more sensitive than that of y
- $\{P\}\, S\, \{Q\}$: If P is true before execution of S, then Q is true after execution of S
  - This is the same notation used in correctness proofs, but P and Q refer to $classes$ rather than $values$

3. Flow Proofs

Proof Rules

- $P[x \leftarrow y]$: the assertion P with every free occurrence of x syntactically replaced by y
- $P \vdash Q$: τ can be derived from P
- $\frac{A_1, A_2, A_3, \ldots, A_n}{B}$: if logical statements $A_1, A_2, A_3, \ldots, A_n$ are true, then so is B

Assignment

- Can change the classification of a variable
- $x \coloneqq E$: x receives information form three sources, E, $\underline{local}$, $\underline{global}$
- Let P be an assertion that is to be true after executing $x := E$. The proof rule for assignment then is:

$$\left\{P\left[\underline{x} \leftarrow \underline{E} \oplus \underline{local} \oplus \underline{global}\right]\right\} x \coloneqq E\{P\}$$

3. Flow Proofs

Alternation

- The effect of the statement

$$if\ \mathrm{B}\ then\ \mathrm{S}_1\ else\ \mathrm{S}_2$$

is to make the information in B available to S_1 and S_2 in correctness logic:

$$\frac{\{P \wedge B\}\ \mathrm{S}_1\ \{Q\}, \{P \wedge \neg B\}\ \mathrm{S}_2\ \{Q\}}{\{P\}\ if\ \mathrm{B}\ then\ \mathrm{S}_1\ else\ \mathrm{S}_2\ \{Q\}}$$

$$\frac{\begin{array}{c}\{V, L', G\}S_1\{V', L', G'\}, \{V, L', G\}S_2\{V', L', G'\} \\ V, L, G \ \vdash \ L'[\underline{local} \leftarrow \underline{local} \oplus \underline{B}]\end{array}}{\{V, L, G\}if\ B\ then\ S_1\ else\ S_2\{V', L, G'\}}$$

Iteration

- $$\frac{\begin{array}{c}\{V, L', G\}S\{V, L', G\}, \\ V, L, G \vdash L'[\underline{local} \leftarrow local \ \oplus \underline{B}], \\ V, L, G \vdash G'[\underline{global} \leftarrow \underline{global} \oplus local \ \oplus \underline{B}]\end{array}}{\{V, L, G\}while\ B\ do\ S\ \{V, L, G'\}}$$

4 A Sound Type System for Flow Security

D.Volpano, G.Smith, C.Irvine “A Sound Type System For Secure Flow Analysis”, Journal of Computer Security, draft printout, (2009), 1–20 IOS Press.

4. Flow Security Sound Type System

- Ensuring secure information flow within programs in the context of multiple sensitivity levels has been widely studied:
  - Example: Denning's work in secure flow analysis and the lattice model

- The soundness of Denning's analysis has not been established
- This paper's purpose: formulate Denning's approach as a $type\ system$ and present a notion of $soundness$ for the system
- The issue this paper is going to deal with is $Soundness$:
  - So far, existing flow logics haven't addressed their $Soundness$

4. Flow Security Sound Type System

The Lattice Model of Information Flow

- An information flow policy is defined by a lattice $(SC, \leq)$, where SC is a finite set of security classes partially ordered by $\leq$

4. Flow Security Sound Type System

Secure Flow Types (An Informal Treatment)

- A typing judgment for our purpose has the form:
  - $\gamma \vdash p : \tau$
- The types of this system are stratified into two levels:
  - Data Types (τ): the security classes of SC
  - Phrase Types (ρ): the types given to:
    - Expressions
    - Variable types of the form $\tau\ var$
    - Command types of the form $\tau\ cmd$
- A variable of type $\tau\ var$ store information whose security class is τ or lower
- A command c has type $\tau\ cmd$ only if it is guaranteed that every assignment within c is made to a variable whose security class is τ or higher.
- $Confinement$ property: needed to ensure secure implicit flows
- Extend the $partial\ order$ $\leq$ to a $subtype\ relation$ which we denote $\subseteq$
- The subtype relation is antimonotonic (contravariant) in the types of commands, meaning that if $\tau \subseteq \tau'$ then $\tau'\ cmd \subseteq \tau\ cmd$

4. Flow Security Sound Type System

Secure Flow Typing Rules (An Informal Treatment)

- Consider following rule:
- $$\frac{\gamma \vdash e{:}\tau\ var \quad \gamma \vdash e'{:}\tau}{\gamma \vdash e{:=}e'{:}\tau\ cmd}$$
- This rule says that in order to ensure that the explicit flow from e' to e is secure, e' and e must agree on their security levels, which is τ
- Upward flow from e' to e is still allowed:
  - If $e : H\ var$ and $e' : L$, then with subtyping, the type of e' can be coerced up to H and $\tau = H$

- Notice that the entire assignment is given type $\tau\ cmd$. The reason is to control implicit flow:
  - $If\ \ x = 1\ then\ \ y{:}= 1\ \ else\ \ y{:}= 0$
- To ensure that such implicit flows are secure, we use the following rule for conditionals:
- $$\frac{\gamma \vdash e{:}\tau \quad \gamma \vdash c{:}\tau\ cmd \quad \gamma \vdash c'{:}\tau\ cmd}{\gamma \vdash if\ \ e\ then\ c\ else\ c'{:}\tau\ cmd}$$
- The intuition is that c and c' are executed in a context where information of level τ is implicitly known for this reason c and c' may only assign to variables of level τ or higher

4. Flow Security Sound Type System

Example

- Suppose $\gamma(x) = \gamma(y) = H\ var$ by the suggested rule for assignment we have:
  - $\gamma\ \vdash y \coloneqq 1 : H\ cmd$
  - $\gamma\ \vdash y \coloneqq 0 : H\ cmd$
- This means that each statement of type H can be suggested as the guard of the conditional statement which its branches are $y \coloneqq 1$ and $y \coloneqq 0$:
  - Example:
    - $if\ x = 1\ then\ y \coloneqq 1\ else\ y \coloneqq 0$
- With $\tau = H$, the secure flow typing rule for conditionals gives

4. Flow Security Sound Type System

A Formal Treatment of the Type System

- $(phrases) \quad p \ ::= \quad e \mid c$
- $(expression) \quad e \ ::= \quad x \mid l \mid n \mid e + e' \mid e - e' \mid e = e' \mid e < e'$
- $(commands) \quad c \ ::= \quad e := e' \mid c; c' \mid \textbf{\textit{if}}\ e\ \textbf{\textit{then}}\ c\ \textbf{\textit{else}}\ c' \mid \textbf{\textit{while}}\ e\ \textbf{\textit{do}}\ c \mid \textbf{\textit{letvar}}\ e\ \textbf{\textit{do}}\ c' \mid \textbf{\textit{letvar}}\ x := e\ \textbf{\textit{in}}\ c$

  - *Metavariable x ranges over identifiers, l over locations (addresses), and n over integer literals*

- $(data\ types) \quad \tau \ := \quad s$
- $(phrase\ types) \ \rho \ := \quad \tau \mid \tau\ var \mid \tau\ cmd$

  - Metavariable s ranges over the set SC of security classes, which is assumed to be partially ordered by $\leq$

4. Flow Security Sound Type System

Typing Rules

- Judgments have the form

$$\lambda; \gamma \vdash p: \rho$$

where:

λ: is a location typing

γ: is an identifier typing

- (INT) $\lambda; \gamma \vdash n: \tau$
- (VAR) $\lambda; \gamma \vdash x: \tau\ var \quad if\ \gamma(x) = \tau\ var$
- (VARLOC) $\lambda; \gamma \vdash l: \tau\ var \quad if\ \lambda(l) = \tau$

- (ARITH) $$\frac{\lambda;\gamma\vdash e:\tau\ ,\quad \lambda;\gamma\vdash e':\tau}{\lambda;\gamma\vdash e+e':\tau}$$
- (R-VAL) $$\frac{\lambda;\gamma\vdash e:\tau\ var}{\lambda;\gamma\vdash e:\tau}$$
- (ASSIGN) $$\frac{\lambda;\gamma\vdash e:\tau\ var\ ,\quad \lambda;\gamma\vdash e':\tau}{\lambda;\gamma\vdash e\coloneqq e':\tau\ cmd}$$
- (COMPOSE) $$\frac{\lambda;\gamma\vdash c:\tau\ cmd\ ,\quad \lambda;\gamma\vdash c':\tau\ cmd}{\lambda;\gamma\vdash c;c':\tau\ cmd}$$

4. Flow Security Sound Type System

- (IF) $$\frac{\lambda;\gamma\vdash e:\tau\ ,\quad \lambda;\gamma\vdash c:\tau\ cmd,\quad \lambda;\gamma\vdash c':\tau\ cmd}{\lambda;\gamma\vdash \mathbf{if}\ e\ \mathbf{then}\ c\ \mathbf{else}\ c':\tau\ cmd}$$

- (WHILE) $$\frac{\lambda;\gamma\vdash e:\tau\ ,\quad \lambda;\gamma\vdash c:\tau\ cmd}{\lambda;\gamma\vdash \mathbf{while}\ e\ \mathbf{do}\ c:\tau\ cmd}$$

- (LETVAR) $$\frac{\lambda;\gamma\vdash e:\tau\ ,\quad \lambda;\gamma[x:\tau\ var]\vdash c:\tau'\ cmd}{\lambda;\gamma\vdash \mathbf{letvar}\ x:=e\ \mathbf{in}\ c:\tau'\ cmd}$$

- **Lemma 4.1)** *(Structural Subtyping): If $\vdash \rho\ \subseteq \rho'$ then either:*
  (a) ρ is of the form τ, ρ' is of the form τ', and $\tau' \leq \tau$
  (b) ρ is of the form $\tau\ var$ and $\rho' = \rho$
  (c) ρ is of the form $\tau\ cmd$, ρ' is of the form $\tau'\ cmd$ and $\tau' \leq \tau$

4. Flow Security Sound Type System

- (Base) $$\frac{\tau \le \tau'}{\vdash \tau \subseteq \tau'}$$
- (REFLEX) $\vdash \rho \subseteq \rho$
- (TRANS) $$\frac{\vdash \rho \subseteq \rho',\ \vdash \rho' \subseteq \rho''}{\vdash \rho \subseteq \rho''}$$
- (CMD) $$\frac{\vdash \tau \subseteq \tau'}{\vdash \tau'\ cmd \subseteq \tau\ cmd}$$
- (SUBTYPE) $$\frac{\lambda;\gamma \vdash p : \rho\ ,\quad \vdash \rho \subseteq \rho'}{\lambda;\gamma \vdash p : \rho'}$$
- **Lemma 4.2)** $\subseteq$ *is a partial order*

4. Flow Security Sound Type System

- (BASE) $\mu \vdash n \longmapsto n$
- (CONTENTS) $\mu \vdash l \longmapsto \mu(l)$ **if** $l \in dom(\mu)$
- (ADD) $$\frac{\mu \vdash e \mapsto n, \mu \vdash e' \mapsto n'}{\mu \vdash e + e' \mapsto n + n'}$$
- (UPDATE) $$\frac{\mu \vdash e \mapsto n, \quad l \in dom(\mu)}{\mu \vdash l := e \mapsto \mu[l := n]}$$
- (SEQUENCE) $$\frac{\mu \vdash c \mapsto \mu', \quad \mu' \vdash c' \mapsto \mu''}{\mu \vdash c; c' \mapsto \mu'}$$
- (BRANCH) $$\frac{\mu \vdash e \mapsto 1, \quad \mu \vdash c \mapsto \mu'}{\mu \vdash \textbf{if } e \textbf{ then } c \textbf{ else } c' \mapsto \mu'}$$

$$\frac{\mu \vdash e \mapsto 0, \quad \mu \vdash c' \mapsto \mu'}{\mu \vdash \textbf{if } e \textbf{ then } c \textbf{ else } c' \mapsto \mu'}$$

- (LOOP) $$\frac{\mu \vdash e \mapsto 0}{\mu \vdash \textbf{while } e \textbf{ do } c \mapsto \boldsymbol{\mu}}$$

$$\frac{\begin{array}{c}\mu \vdash e \mapsto 1, \\ \mu \vdash c \mapsto \mu', \\ \mu' \vdash \textbf{while } e \textbf{ do } c \mapsto \mu''\end{array}}{\mu \vdash \textbf{while } e \textbf{ do } c \mapsto \mu''}$$

5 Type Soundness

D.Volpano, G.Smith, C.Irvine “A Sound Type System For Secure Flow Analysis”, Journal of Computer Security, draft printout, (2009), 1–20 IOS Press.

5. Type Soundness

- Soundness: if $\lambda(l) = \tau$, for some location l, then one can arbitrarily alter the initial value of any location l' such that $\lambda(l')$ is not a subtype of τ, execute the program, and the final value of l will be the same provided the program terminates successfully

Make Rules Syntax Directed

- (R−VAL′) $$\frac{\lambda;\gamma \vdash e:\tau\ var\ , \tau' \leq \tau}{\lambda;\gamma \vdash e:\tau}$$

- (ASSIGN′) $$\frac{\lambda;\gamma \vdash e:\tau\ var\ , \quad \lambda;\gamma \vdash e':\tau\ , \tau' \leq \tau}{\lambda;\gamma \vdash e \coloneqq e':\tau\ cmd}$$

- (IF′) $$\frac{\lambda;\gamma \vdash e:\tau\ , \quad \lambda;\gamma \vdash c:\tau\ cmd, \quad \lambda;\gamma \vdash c':\tau\ cmd \quad \tau' \leq \tau}{\lambda;\gamma \vdash \textbf{if}\ e\ \textbf{then}\ c\ \textbf{else}\ c':\tau\ cmd}$$

- (WHILE′) $$\frac{\lambda;\gamma \vdash e:\tau\ , \quad \lambda;\gamma \vdash c:\tau\ cmd \quad \tau' \leq \tau}{\lambda;\gamma \vdash \textbf{while}\ e\ \textbf{do}\ c:\tau\ cmd}$$

5. Type Soundness

- **Lemma 6.1)** If $\lambda\,;\gamma \vdash_s p{:}\,\rho$ and $\vdash \rho \subseteq \rho'$, then $\lambda;\gamma \vdash_s p{:}\,\rho'$
- **Lemma 6.2)** $\lambda\,;\gamma \vdash p{:}\,\rho$ iff $\lambda\,;\gamma \vdash_s p{:}\,\rho$
- **Lemma 6.3)** (Simple Security) If $\lambda \vdash e{:}\,\tau$, then for every l in e, $\lambda(l) \leq \tau$
- **Lemma 6.3)** (Confinement) If $\lambda,\gamma \vdash c{:}\,\tau\ cmd$, then for every l assigned to in c, $\lambda(l) \geq \tau$
- **Lemma 6.5)** (Substitution) If $\lambda\,;\gamma \vdash l{:}\,\tau\ var$ and $\lambda\,;\gamma[x{:}\,\tau\ var] \vdash c{:}\,\tau'\ cmd$, then$\lambda\,;\gamma \vdash [l/x]c{:}\ \tau'\ cmd$

- **Lemma 6.8)** (Type Soundness) Suppose

  (a) $\lambda \vdash c{:}\,p,$

  (b) $\mu \vdash c \mapsto \mu',$

  (c) $v \vdash c \mapsto v',$

  (d) $dom(\mu) = dom(v) = dom\ (\lambda)$

  (e) $v(l) = \mu(l)\ for\ all\ l\ such\ that\ \lambda(l) \leq \tau$

  then $v'(l) = \mu'(l)$ for all l such that $\lambda(l) \leq \tau$

5. Type Soundness

- **Proof)** By induction on the structure of the derivation of $\mu \vdash c \mapsto \mu'$. (UPDATE)

(UPDATE): Suppose the evaluation under μ ends with:

$$\frac{\lambda \vdash e \mapsto n, \quad l \in dom(\mu)}{\mu \vdash l := e \mapsto \mu[l := n]}$$

And the evaluation under v ends with:

$$\frac{v \vdash e \mapsto n', \quad l \in dom(v)}{v \vdash l := e \mapsto v[l := n']}$$

And the typing ends with an application of rule (Assign):

$$\frac{\lambda \vdash l : t_2\ var, \quad \lambda \vdash e : t_2, \quad \tau_1 \leq \tau_2}{\lambda \vdash l := e : \tau_1\ cmd}$$

5. Type Soundness

Type Soundness Proof

1. $\tau_2 \leq \tau$: By the `Simple Security Lemma`, $\lambda(l') \leq \tau_2$ for every l' in e. Since $\leq$ is transitive, $\lambda(l') \leq \tau$ for every l' in e. Thus, by hypothesis (e), $\mu(l') = v(l')$ for every l' in e, so $n = n'$.Therefore, $\mu[l \coloneqq n](l') = v[l \coloneqq n'](l')$ for all l' such that $\lambda(l') \leq \tau$.
2. $\tau_2 \nleq \tau$: By rule (VARLOC), $\lambda(l) = \tau_2$, so $\lambda(l) \nleq \tau$. So by hypothesis (e),

$$\mu[l \coloneqq n](l') = v[l \coloneqq n'](l')$$

for all l' such that $\lambda(l') \leq \tau$.

Resources

- D.Volpano, G.Smith, C.Irvine "A Sound Type System For Secure Flow Analysis", Journal of Computer Security, draft printout, (2009), 1–20 IOS Press.
- G. Andrews, R. Reitman, "An Axiomatic Approach to Information Flow in Programs", ACM Transactions on Programming Languages and Systems 2, 1, (1980), 56–76.
- D. Bell, L. LaPadula, Secure Computer System: Mathematical Foundations and Model, MITRE Corp. Technical Report M74-244, 1973.
- D. Denning, "A Lattice Model of Secure Information Flow", Communications of the ACM 19, 5, (1976), 236–242.
- D. Denning, P. Denning, "Certification of Programs for Secure Information Flow", Communications of the ACM 20, 7, (1977), 504–513.

Thank You