

# Language-Based Information-Flow Security

CS 6V81 – Language Based Security

Jonathan Burke

3/5/2008

# Motivation and Problem Statement



- Beyond Buffer Overflows
- Protecting Confidentiality of Information
- Advent of Complex Systems
  - Military, medical, and financial institutions are all sensitive to data theft
  - Web systems – email, shopping, business transactions – all sensitive to piracy and data theft
- Techniques that are currently employed provide weak guarantees of data safety

# Standard Security Methods and Information Flow

- Access Control/Capabilities – permissions allow access to data/operations
  - Faults:
    - Doesn't effect how data is used after it has been accessed
    - To enforce Information flow policies, it can only give data to those processes that won't leak data
    - cannot identify security compliant processes
- Firewalls, Anti-Virus, Encryption
  - Faults:
    - Don't support end-to-end security
    - Don't care what happens when data is released



# Language Based Security

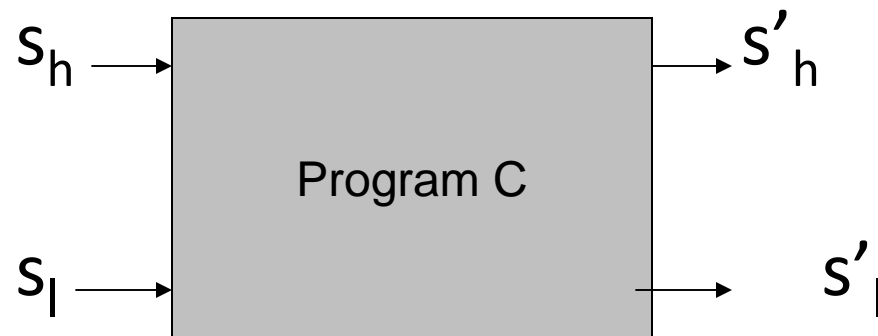
- Traditional approaches of LBS aren't focused on information-flow
- End – to –end policies need to be enforced
- Many LBS approaches discussed so far don't address covert channels (see slide 8)
- On the other hand, type safety is a key component in many information flow control techniques

# Information-Flow Security

- Concerned with the dissemination of data
- analysis to ensure that the system as a whole conforms to the confidentiality policies of its users
- Information cannot flow to a location where policy is violated
- End-to-end security

# Basic Model

- Input/Output State  $s = (s_h, s_l)$ ,  $s' = (s'_h, s'_l)$



# Desired Properties

- Confidentiality – confidential information flows only to appropriate destinations



- Integrity – information cannot come from inappropriate sources



- Availability – information/resources are accessible in a timely manner



- Noninterference - No two executions are observably different if they differ solely by confidential input

# Noninterference

- Confidentiality Noninterference – equivalent low-confidentiality inputs always result in equivalent low-confidentiality outputs
- Integrity Noninterference – equivalent high-integrity inputs always result in equivalent high-integrity outputs
- Availability Noninterference – availability of an output is not interfered with by the availability of an input, whose policy is not as strong as the output

# Covert Channels



- Channel – a mechanism for signaling information through computing
- Explicit flows – assignment of sensitive information to observable variables
- Covert Channel – a mechanism exploited for information signaling, when information signaling was not it's intended use
  - Presents some of the greatest challenges

# Covert Channels

- Implicit flow – signals information through control structure of the program
- Termination channel – signals info through termination or non-termination of a computation
- Timing channels – info signaled by the time an action occurs rather than its data

```
h := h mod 2;  
l := 0;  
if h = 1 then l := 1  
    else skip
```

```
while sensitive = 1 do skip;
```

```
if sensitive = 1 then Clong  
    else skip;
```

- Probabilistic channels – confidential data changes the distribution of output data
- Resource exhaustion – information revealed through exhaustion of a finite resource
- Power channel – information is embedded based on the power consumed by a computer
- An abstract computer  $\neq$  concrete computer

# Early Flow Control Techniques

- Mandatory Access Control

- Each data item is labeled with a security level, computed simultaneously at run-time



- Security level/label – a simple confidentiality policy

- Avoid assignments from high level objects to low level

- Problem: it's hard to identify implicit flows

- Process Sensitivity Label
  - Monitors the confidentiality level of data used to control process
  - kept track for the current location of program
  - Label Creep
    - Once high the process sensitivity label must remain high to ensure confidentiality
    - Data labels increase monotonically

- Static Information-Flow Control
  - Uses static program analysis and type checking
  - Static Security Type Checking
    - Every program has a security type
    - Security type – ordinary type + policy labels
    - Security enforced through type checking
  - Program Counter Label – tracks dependencies of program counter
  - Combats Label Creep – program counter label can be reset after high-conditionals and high-guarded loops

# Semantics Based Security

- allows reasoning about concepts such as noninterference, in this case we use a binary model (high, low)
- Input/Output State  $s = (s_h, s_l)$
- $\llbracket C \rrbracket : S \rightarrow S_{\perp}$  (where  $S_{\perp} = S \cup \{\perp\}$  and  $\perp \notin S$ )
- Equivalence of inputs
  - $s =_L s'$  iff  $s^l = s'^l$
- Low – view equivalence –
  - $\forall s_1, s_2 \in S. s_1 =_L s_2 \implies \llbracket C \rrbracket s_1 \approx_L \llbracket C \rrbracket s_2$

# Security Type System

- A collection of typing rules that describe what security type is assigned to a program (expression) based on the types of the subprograms (subexpressions)

$$[\text{E1-2}] \quad \frac{\vdash \text{exp} : \text{high} \quad h \notin \text{Vars}(\text{exp})}{\vdash \text{exp} : \text{low}}$$

$$[\text{C1-3}] \quad \frac{[pc] \vdash \text{skip} \quad [pc] \vdash h := \text{exp} \quad \frac{\vdash \text{exp} : \text{low}}{[low] \vdash l := \text{exp}}}{[pc] \vdash h := \text{exp}}$$

$$[\text{C4-5}] \quad \frac{\frac{[pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash C_1; C_2} \quad \frac{\vdash \text{exp} : pc \quad [pc] \vdash C}{[pc] \vdash \text{while } \text{exp} \text{ do } C}}{[pc] \vdash \text{while } \text{exp} \text{ do } C}$$

$$[\text{C6-7}] \quad \frac{\frac{\vdash \text{exp} : pc \quad [pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash \text{if } \text{exp} \text{ then } C_1 \text{ else } C_2} \quad \frac{[high] \vdash C}{[low] \vdash C}}{[pc] \vdash \text{if } \text{exp} \text{ then } C_1 \text{ else } C_2}$$

E1	$\vdash \text{exp} : \text{high}$	An expression can always be typable as high.
E2	$\frac{h \notin \text{Vars}(\text{exp})}{\vdash \text{exp} : \text{low}}$	An expression is low only if it contains no high variables.
C1	$[pc] \vdash \text{skip}$	The skip command is always typable.
C2	$[pc] \vdash h := \text{exp}$	Assignment is always typable as high.
C3	$\frac{\vdash \text{exp} : \text{low}}{[low] \vdash l := \text{exp}}$	Low variables are only assigned low expressions.
C4	$\frac{[pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash C_1; C_2}$	If commands $C_1$ and $C_2$ are typable in $[pc]$ then so is the sequence $C_1; C_2$ typable in $[pc]$

C5  $\frac{\vdash \text{exp} : pc \quad [pc] \vdash C}{[pc] \vdash \text{while } \text{exp} \text{ do } C}$  If a loop expression is the type pc and the loop command is typable within [pc] then the loop is typable by [pc].

C6  $\frac{\vdash \text{exp} : pc \quad [pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash \text{if } \text{exp} \text{ then } C_1 \text{ else } C_2}$  The if statement is typable by [pc] if both the  $C_1$  and  $C_2$  are typable by [pc].

C7  $\frac{[high] \vdash C}{[low] \vdash C}$  If a program is typable in a high context then it is also typable in a low context.

C3 prevents explicit flows.

No assignments from  $[high] \vdash C$  to a [low] variable.

C5-6 Ensure that a *high guarded loop* or *high conditional* is typed [high]

C7 allows the program counter to be reset to avoid label creep.

# Typed Programs

- [low]  $\vdash h := l + 4; l = l - 5$
- [high]  $\vdash \text{if } h = 1 \text{ then } h := h + 4 \text{ else skip}$
  
- Untypable  $l := h$ 
  - isn't covered by C2 or C3
  
- Untypable  $\text{if } h := 1 \text{ then } l := 1 \text{ else skip}$ 
  - isn't covered by C6

# Trends and Directions of Research

- Language Expressiveness – research related to accommodating full expressiveness of programming languages
  - Procedures – polymorphism, and polymorphic security types
  - Functions – first class functions, and functional languages
  - Exceptions – uncaught exceptions can create implicit flows
  - Objects – subsumes first class functions
  - Concurrency – see next slide

- Concurrency – always makes things interesting
  - Nondeterminism – noninterference is described as a property of sequential languages

– Two Processes

- P1:  $h := 0; l := h$
- P2:  $h := h2$

Two Serialized Schedules

$h := 0; l := h$

$h := h2;$

---

$h := 0; \quad l := h$

$h := h2$

- Leino and Joshi proposed equivalence relation

$$\forall s \in S. \llbracket HH; C; HH \rrbracket s \approx \llbracket C; HH \rrbracket s$$

- Another threading problem:

(if  $h = 1$  then  $C_{long}$  else skip);  $l := 1$  ||  $l := 0$

– If  $h=1$  then  $l$  is much more likely to be 1

- Schedule-independent security could solve this problem

# Distribution Channels

- 3 Assumptions
  - Principals must be able to exchange observable messages
  - Principals may be distrustful of each other
  - Components may fail, including failure by subversion
- Secure Program Partitioning
  - Goal – protect confidentiality of distrustful principals
  - Idea – a principal's security is not threatened unless a machine they trust fails
  - Partitions program into communicating subprogram with security types to specify confidentiality/integrity

# Covert Channels

- Termination Channel – change low-view equivalence

$$s \approx_L s' \text{ iff either } s, s' \in S \text{ and } s =_L s' \text{ or } s = s' = \perp$$

- Timing Channel

– Aimp includes a timed type

– Program transformation  $C \rightarrow C':SI$

- $C$  = transformed program,  $SI$  = low slice of  $C$

$$\frac{C_1 \hookrightarrow C'_1 : Sl_1 \quad C_2 \hookrightarrow C'_2 : Sl_2 \quad \text{exp : high} \quad \text{al}(Sl_1) = \text{al}(Sl_2) = \text{false}}{\text{if exp then } C_1 \text{ else } C_2 \hookrightarrow \text{if exp then } C'_1; Sl_2 \text{ else } Sl_1; C'_2 : \text{if-skip}(Sl_1; Sl_2)}$$

# Security Policies

- Noninterference can be burdensome
  - Implies that you can't downgrade security level from high to low
- Selective declassification –static analysis on process authority and principal relationships to describe declassification capability
- Admissibility – permitted flows are specified in the security condition
- Robust declassification – active attackers (attackers that can affect system behavior) cannot observe any more information than passive attackers

# Open Challenges



- System Wide Security - systems are only secure to their weakest point
- Certifying Compilation/Validation – attempts to reduce TCB
- Abstraction-Violating Attacks – every attacks aren't always portrayed in attacker model
- Dynamic Policies – enforcing policies unknown at run-time
- Practical issues – few type security inference implementations, efficiency, program rejection precision

# Conclusion

- Information flow control describes a means of providing confidentiality over a networked system
- Security Type Systems – apply a security value within a type system, security enforced through type checking
  - Type checking has been thoroughly studied
  - Type checking provides composition
- Semantic-based security models – are suitable to describe end-to-end policies such as noninterference

# Examples

```
h := h mod 2;  
l := 0;  
if h = 1 then l := 1  
    else skip
```

```
while sensitive = 1 do skip;
```

```
if sensitive = 1 then Clong  
    else skip;
```

# Citations

- Andrei Sabelfeld and Andrew C. Myers. [Language-Based Information-Flow Security](#). *IEEE Journal on Selected Areas in Communications*, 21(1):5-19, January 2003.
- Lantian Zheng and Andrew C. Myers. [End-to-End Availability Policies and Noninterference](#). In *Proceedings of the 18th IEEE Computer Security Foundations Workshop (CSFW'05)*, 272-286, June 2005.

## Commands for Typed a Security Language

$$C ::= \text{skip} \mid \text{var} := \text{exp} \mid C_1; C_2 \\ \mid \text{if } \text{exp} \text{ then } C_1 \text{ else } C_2 \mid \text{while } \text{exp} \text{ do } C$$