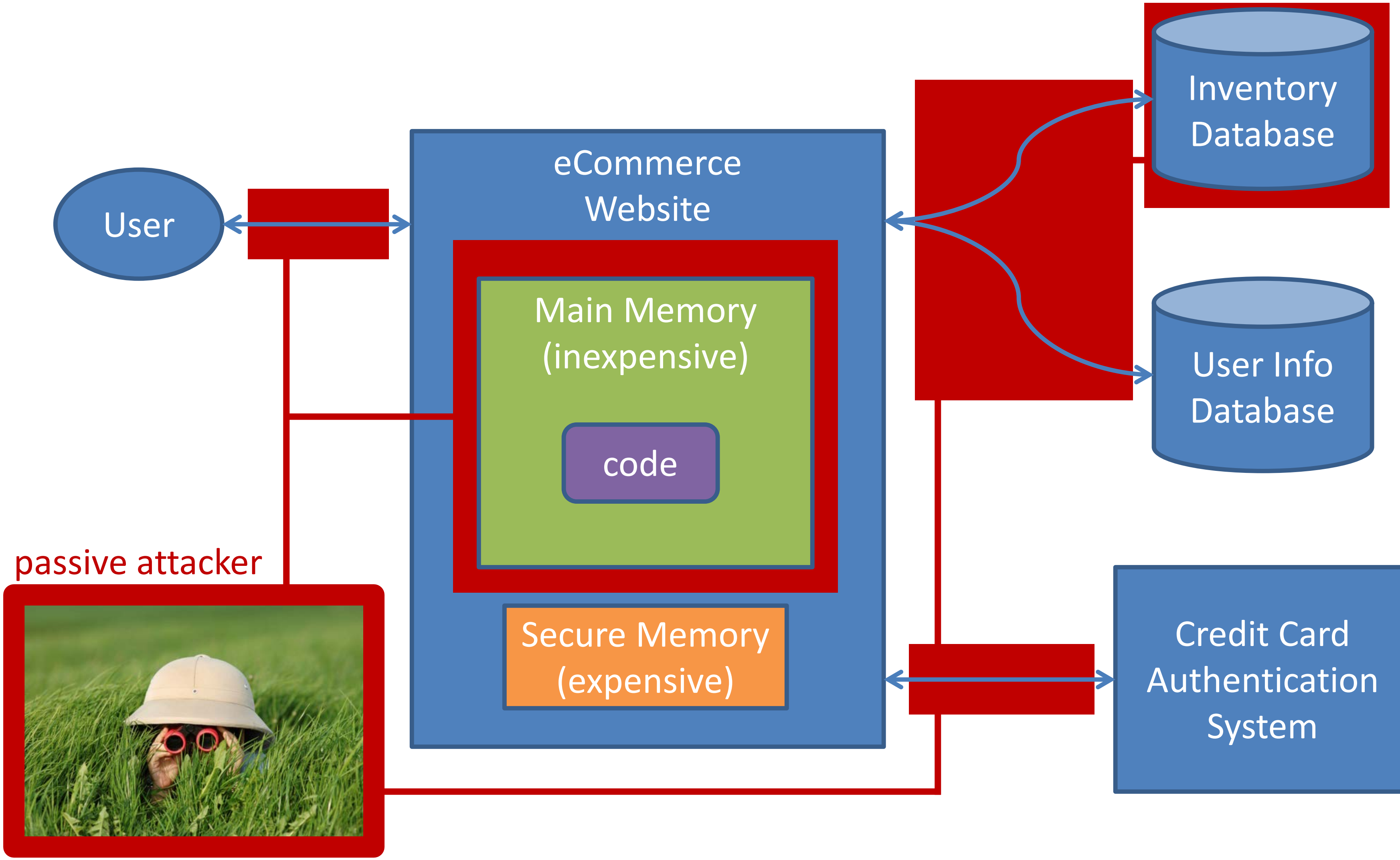
Language-Based Information-Flow Security

Dr. Kevin W. Hamlen

End-to-end Confidentiality

Problem: How to prevent information leaks?



Goals

- Provide tools to…
  - write software that doesn't leak secrets
  - detect potential information leaks in existing code
  - measure worst-case information leaks quantitatively
- End-to-end security
  - modular verification strategies
  - comprehensive separate verification = full-system verification
  - cross-language, cross-hardware
- Mathematical Foundations
  - what does "information leak" really mean?
  - how to model information flow in complex systems?
  - relation to data integrity enforcement?

Non-LBS Approaches

- Access control
  - deny read-access to untrusted principals
  - examples: OS access control lists (ACL's), private fields in Java
  - no guarantee that principals granted read-access will not (accidentally) leak the secret!
  - how to identify these untrustworthy principals?
- Firewalls
  - some info always exchanged
  - how to prove that info is free of secrets?
  - not enough to scan for byte sequences
- Encryption
  - protects from man-in-middle eavesdropping
  - eventually data is decrypted
  - how to prove that decrypted secrets are not leaked?

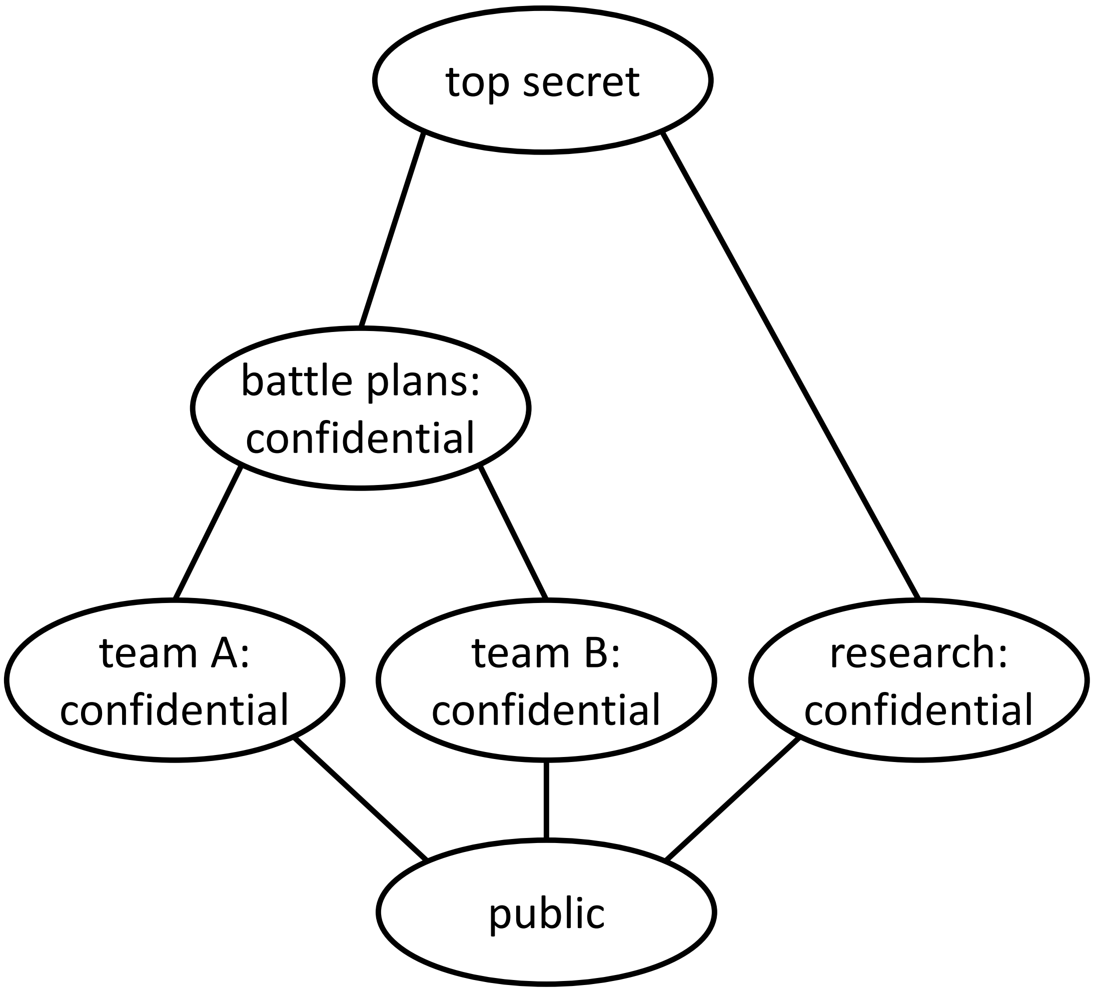
Channels

- Notation:
  - low-security (attacker-readable) variables: ℓ
  - high-security (secret) variables: h
- Information Flows
  - **Explicit:** $\ell := h$
  - **Implicit:** if h>0 then ℓ:=0 else ℓ:=1
- Covert Channels
  - **Termination:** if h>0 then halt
  - **Probabilistic:** $\ell := h$ + rand(100)
  - **Resource exhaustion:** for i:=1 to ℓ do malloc(h)
  - **Power:** if h>0 then decrypt(database) else skip

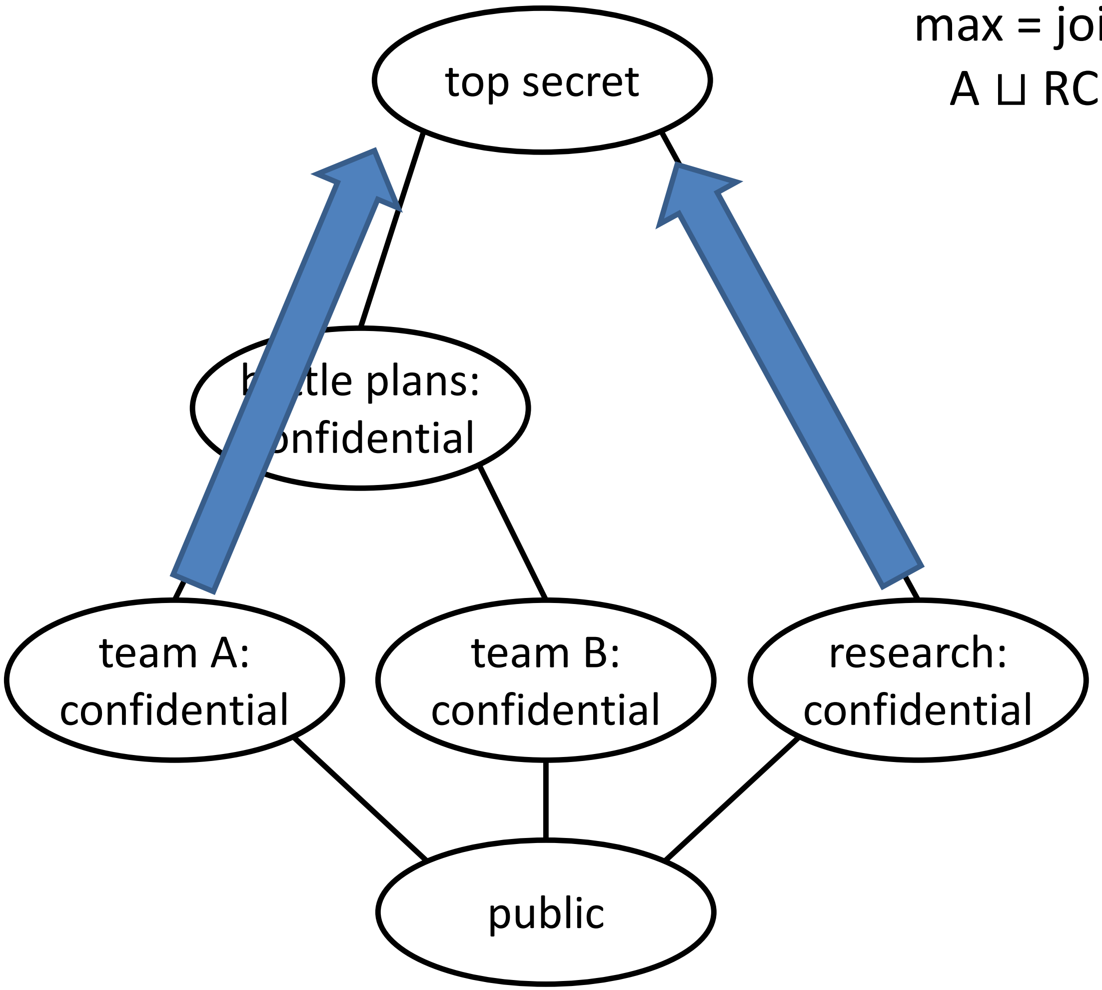
Integrity & Confidentiality

- Low-integrity data must not be treated as trustworthy
- Can be seen as duals [Biba, USAF ’77]
  - Confidentiality: no flows (reads) from high to low
  - Integrity: no flows (writes) from low to high
- Mandatory Access Control approach [Bell and LaPadula, MITRE ’73]
  - each variable x gets a confidentiality label c(x) and an integrity label i(x)
  - flows from y to x (e.g., x:=y) change labels as follows:
    - confidentiality increases: c(x) := max(c(x),c(y))
    - integrity decreases: i(x) := min(i(x),i(y))
  - labels conform to a security lattice

A Confidentiality Label Lattice



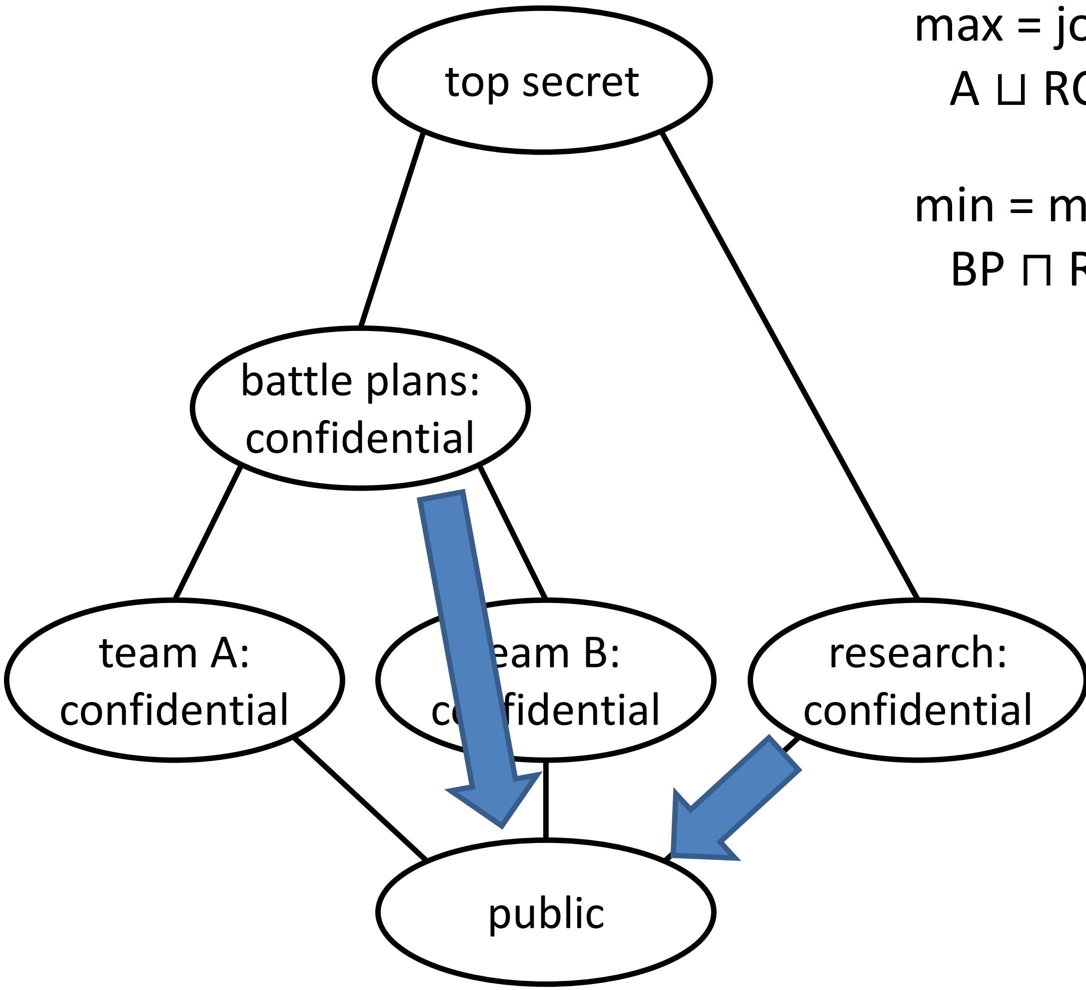
A Confidentiality Label Lattice



max = join = least common parent

$A \sqcup RC = TS$

A Confidentiality Label Lattice



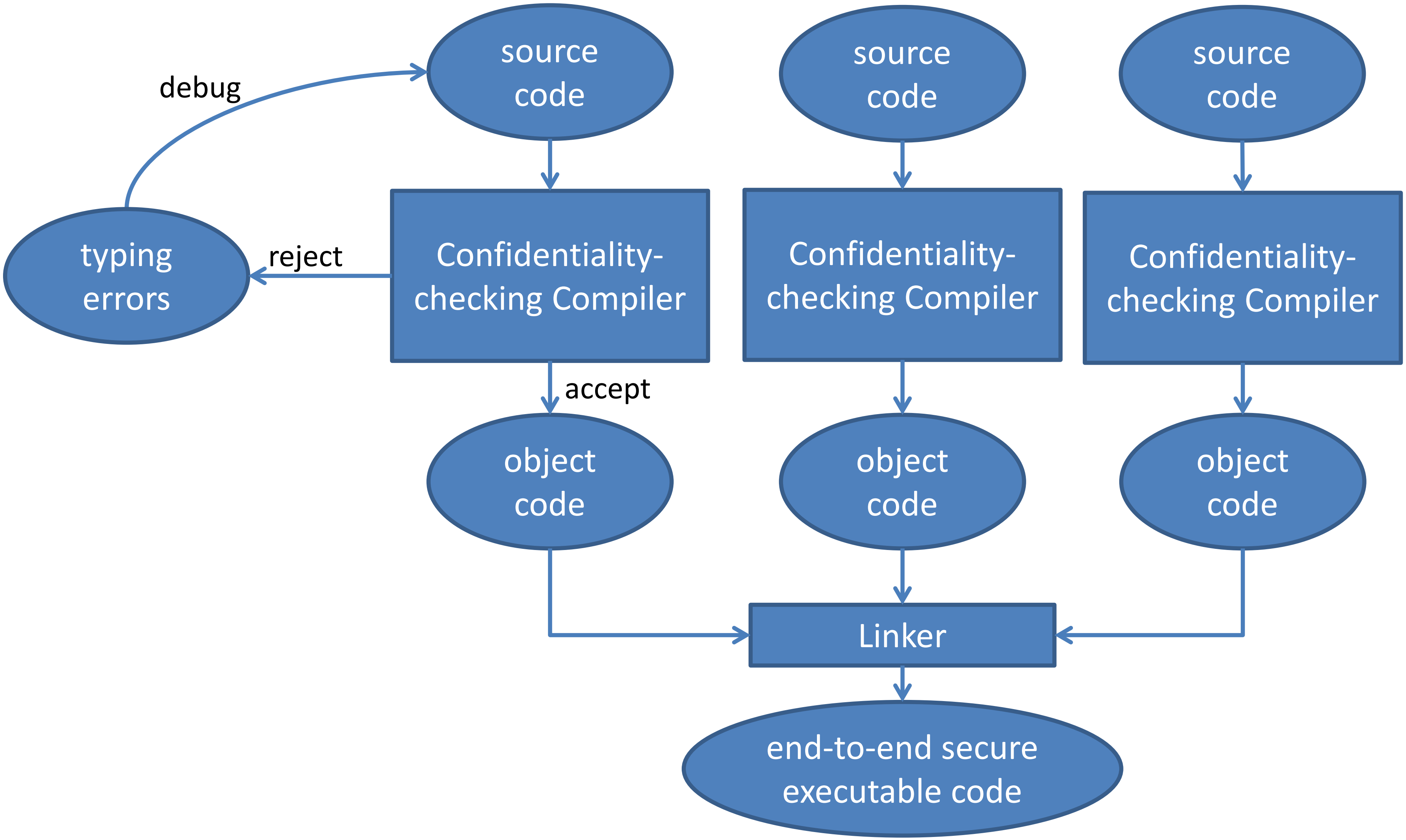
max = join = least common parent

A ⊔ RC = TS

min = meet = greatest common child

BP ⊓ RC = P

Type-based Approach



Type-based Information Flow Control

[Sabelfeld & Myers, IEEE J. Selected Areas in Communications 21(1), 2003]

c ::= **skip** | c_1**;**c_2 | v**:=**e | **if** e **then** c_1 **else** c_2 | **while** e **do** c

e ::= n | v | e_1**+**e_2

τ ::= **high** | **low**

$\Gamma : (v \cup \{\mathbf{pc}\}) \rightarrow \tau$

Typing Rules for Expressions:

$$\Gamma \vdash n : \mathbf{low}$$

$$\Gamma \vdash v : \Gamma(v)$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1 + e_2 : \tau_1 \sqcup \tau_2}$$

Type-checking Commands

$$\Gamma \vdash \textbf{skip}$$

$$\frac{\Gamma \vdash c_1 \qquad \Gamma \vdash c_2}{\Gamma \vdash c_1; c_2}$$

$$\frac{\Gamma \vdash e : \tau \qquad \Gamma(v) \geq \tau \qquad \Gamma(v) \geq \Gamma(\textbf{pc})}{\Gamma \vdash \mathrm{v{:=}e}}$$

$$\frac{\Gamma \vdash e : \tau \qquad \Gamma[\textbf{pc}{:=}\tau] \vdash c_1 \qquad \Gamma[\textbf{pc}{:=}\tau] \vdash c_2}{\Gamma \vdash \textbf{if } e \textbf{ then } c_1 \textbf{ else } c_2}$$

$$\frac{\Gamma \vdash e : \tau \qquad \Gamma[\textbf{pc}{:=}\tau] \vdash c}{\Gamma \vdash \textbf{while } e \textbf{ do } c}$$

implicit flow protection!

Proving Noninterference

- Noninterference
  - **Def:** *x* interferes with *y* if the value of *x* affects the value of *y*
  - wish to prove that *h* does not interfere with ℓ
- Low views
  - **Def:** Low view of store σ is its low-security variables
  - **Def:** $\sigma_1 =_L \sigma_2$ if for all low-security variables ℓ, we have $\sigma_1(\ell) = \sigma_2(\ell)$
- Proof goal:
  - If *c* is well-typed and $\sigma_1 =_L \sigma_2$ then $\mathcal{D}[\![c]\!]\sigma_1 =_L \mathcal{D}[\![c]\!]\sigma_2$
  - Running *c* does not make secret low-viewable

Active Research Directions

- Functions/Procedures
  - recursion and polymorphism
  - SLam calculus [Heintze & Riecke, POPL'98]
    - λ-calculus with confidentiality & integrity labels
- Exceptions
  - many opportunities for information disclosure
  - overly conservative rejection problematic
- Objects
  - JFlow [Myers, POPL '99]
- Distributed Computing
  - Secure Program Partitioning [Zdancewic, Zheng, Nystrom & Myers, SOSP'01]
  - common source split among mutually-distrusting hosts
  - synthesize appropriate communication protocols for servers/clients

Active Research Directions

- Concurrency
  - Nondeterminism
    - possibilistic approach – high inputs must not interfere with SET of possible low views
    - equational approach – define HH="havoc on *h*" and prove $\mathcal{D}[\![\text{HH};c;\text{HH}]\!]\sigma = \mathcal{D}[\![c;\text{HH}]\!]\sigma$ [Leino & Joshi, MPC'98]
  - Multithreading
    - desynchronized use of *h*: $(h{:=}0;\ \ell{:=}h) \parallel (h{:=}h')$
    - timing-to-explicit: $(\textbf{if}\ h{=}1\ \textbf{then}\ c_{long}\ \textbf{else skip};\ \ell{:=}1) \parallel (\ell{:=}0)$
    - scheduler-dependence
    - synchronization strategies

The Declassification Problem

- Example:
  - password authenticator application
  - always rejected by this type system! Why?
- Approaches
  - trusted declassification operations
  - spi-calculus: π-calculus for cryptography [Abadi & Gordon, Information and Computation, 148(1), 1999]
  - robust declassification: active attackers are no more powerful than passive ones [Zdancewic & Myers, CSFW'01]

Open Problems

- System-wide (end-to-end) security
- Certifying compilation for confidentiality
  – not quite so open anymore
- Dynamic policy-changes
  – see Flow Locks [Broberg & Sands, ESOP'06]
- Practical issues
  – hard to satisfy the type-checker
  – many covert channels (e.g., caches)
  – power channels (e.g., smartcards)

Discussion

- Why aren’t confidentiality-checking compilers standard practice yet?
  - It’s been 10 years now…
- Is the covert channel problem surmountable?
- What about quantitative instead of binary information flow?
  - still a significant open question
  - number of bits of information disclosed?
  - number of bits per time interval?
  - probability of bits disclosed?
- Could this be done at the binary level? Would there be any advantage to this over source-level?
- Would it be better to devise a new language instead of retrofitting an existing one (e.g., Java)?