

JFlow

Lavin Urbano

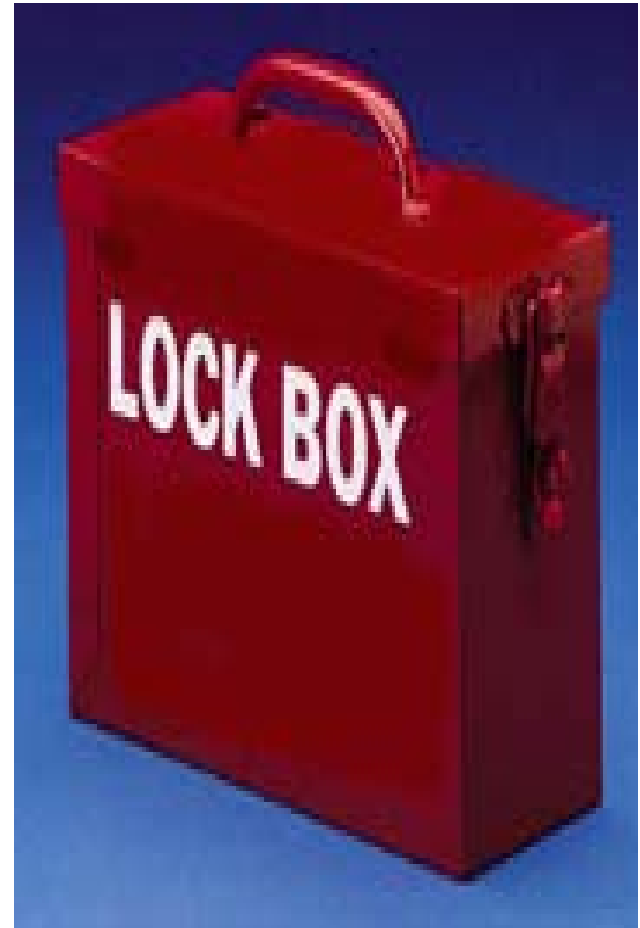
The Problem

- Growing need for data privacy with the Increased mobility of data
- Secure data locally and globally
- Information leak through computation



Traditional Methods

- Access control mechanisms
 - Firewalls, anti-virus, encryption
- Mandatory access control mechanisms
 - Dynamically associate run-time security class
 - Associate with each computed data



Traditional Methods Fail

- Access control mechanisms
 - Control information release
 - Do not control information propagation
- Mandatory access control mechanisms
 - Slow hierarchical systems
 - Restrictiveness prevents easy writing of useful applications

Static Checking Needed

- Information Flow Checking Within Programs
- Permits Security Class Tracking through computations
- Avoids dynamic security class overhead

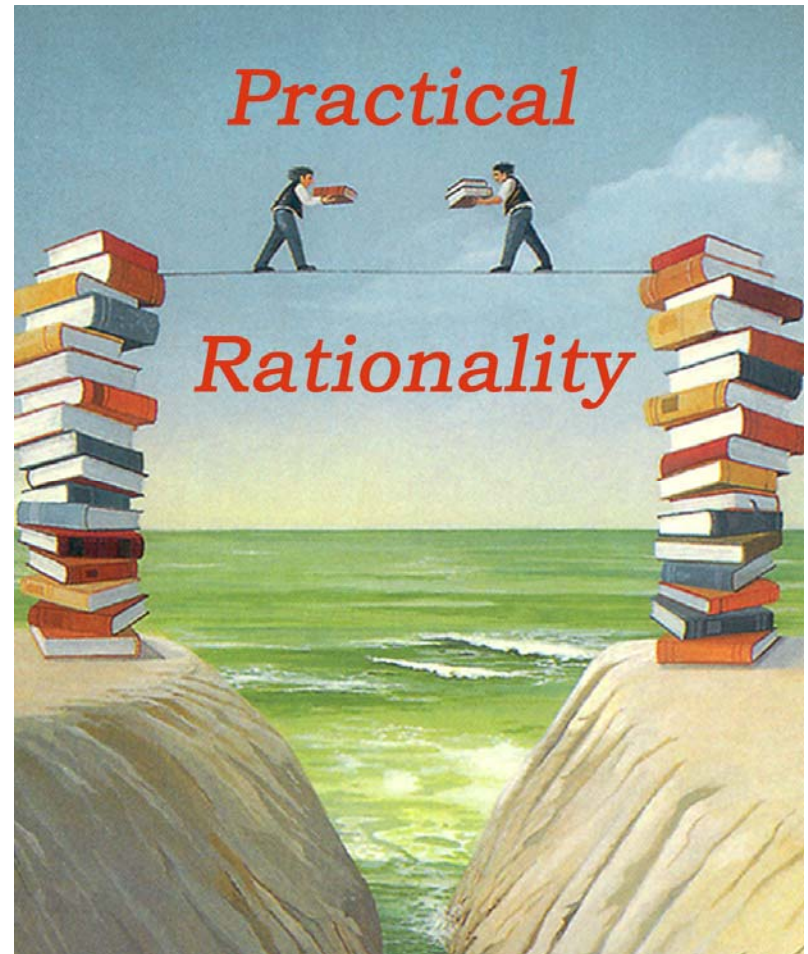


Previous Static Checking Programs

- Slam Calculus
- Focus on accuracy of static checks
- Either too limited or too restrictive

JFlow

- Claims the first practical static checking language
- Combines accuracy with practicality
 - Provides wider scope
 - Less restrictive
- Uses Type Checking



Advantages of JFlow

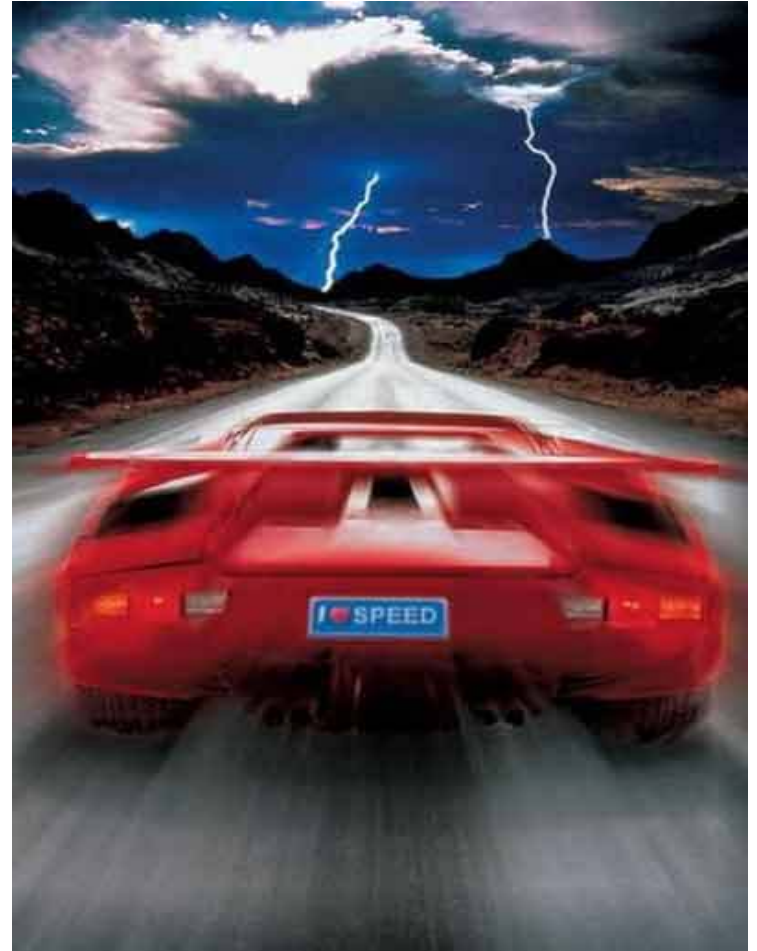
- Prevents information leak through Storage Channels
- Allows secure server and applet writing
- Extends Java
- Supports many language features not previously integrated

New Features Added

- Decentralized Label Model
- Access control
- Label polymorphism
- Run-time label checking
- First class label values
- Automatic Label Inference

JFlow Compiler

- Source to source translator
- Offers advantages
 - Little code space
 - Little data space
 - Little run time overhead



Language Description

- Labels
- Labeled types
- Implicit Flows
- Run-time labels
- Run-time principals
- Authority and declassification
- Classes
- Methods

Labels

- Security policies label data values
- Policies restrict data movement
- $L = \{O_1: r_1, r_2; O_2: r_2, r_3\}$
- **Owner = principal**
- Policy reader agreement determines data access

Definitions

- Declassification:
 - A principal, p , may choose to relax a policy that it owns
- Decentralization
 - P cannot weaken other principal's policies
- Principal Hierarchy
 - Some principals can act for other principals

Labeled Types

- Every value has a labeled type
 - Ordinary Java type
 - Propagation behavior description
- Goals:
 - Ensures that every expression is at least as restrictive as the label of every value that it might produce
- Label expressions
 - {a}
 - Labeled value as restricted as a is
 - {a; o: r}
 - Reader can only be r
 - {}
 - Contains no policies

Static Checks

- Type checks
 - Expression's static type is a supertype of resultant dynamic expression
- Label checks
 - Apparent label at least as restrictive as resultant actual label
 - Apparent label at least as restrictive as affective actual labels

Implicit Flows

- Determines label through evaluation context
- Prevents information leaks through implicit flows using program counter label, pc
- Associates pc with every statement and expression
- pc represents the information learned from knowledge that statement or expression was evaluated

Dynamic Labels and Principals

- Labels and principals can be values
- Arguments are always final
- Dynamic labels handled statically
- Enables static propagation of unknown labels
- Principals can be declassified

Authority and declassification

- Authority:
 - the capability to act for some set of principals
- Authority determines declassification
- Both are checked statically

Static Checking

- Performs two types of static checks
 - Type checking
 - Label checking
- Labels = type constructors
- Classification of statements and arguments differ
- Type checks similar to Java
- Label checking differs

Judgements

- $A \vdash_T E : T$
 - E has type T in environment A
- $A \vdash E : X$
 - E has path labels X in environment A

Sample Label Checking Rules

- $\text{true} \mid \{A \vdash \text{literal} : X_{\emptyset}[\underline{n} := A[\underline{pc}], \text{nv} := A[\underline{pc}]]\}$
 - a literal expression always terminates normally
 - its value is labeled with the current \underline{pc}
- $\text{true} \mid \{A \vdash ; : X_{\emptyset}[\underline{n} := A[\underline{pc}]]\}$
 - an empty statement always terminates normally
 - its \underline{pc} remains the same as at the start



Translation

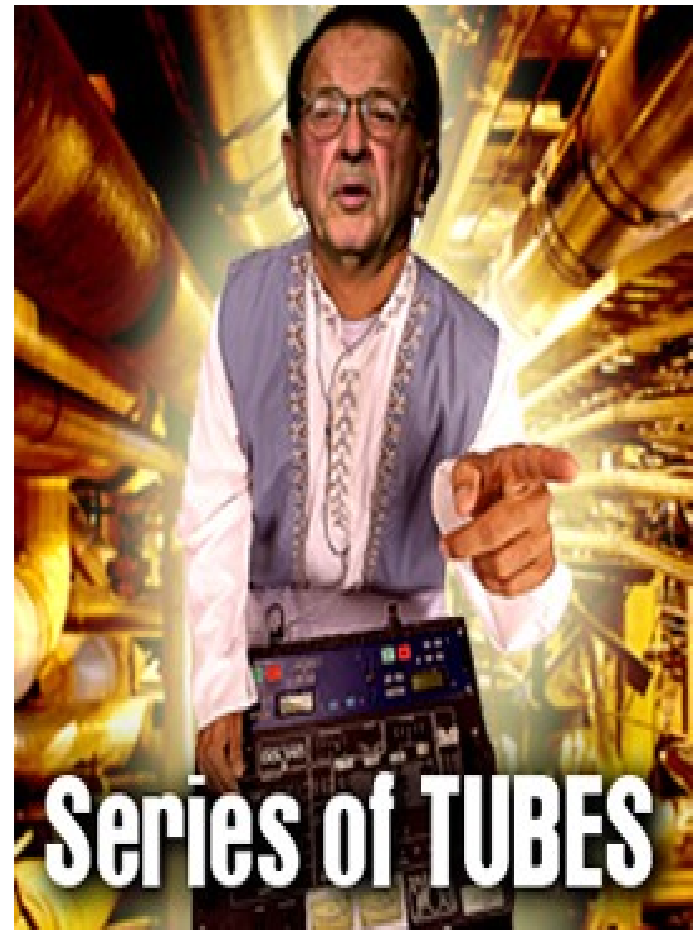
- most annotations have no run-time representation
- translation erases them
- leaves a java program
- translation code is fast

Limitations

- Threads
 - Does not prevent covert information communication
- Timing Channels
 - Information can be gained through using system clock
- Finalizers
 - not part of JFlow
- Backward Compatibility
 - Not backward compatible with Java

Conclusion

- First static checking language
- Supports Java
- Less restrictive
- Integrates many features



Questions

- How does backward incompatibility affect adoption?
- Does it prevent information gathering through covert channels?
- Can it be made backward compatible?

Citations

- Andrew C. Meyers. JFlow: Practical Mostly-Static Information Flow Control
POPL'00, January 1999
- Andrei Sabelfeld and Andrew C. Myers. Language-Based Information-Flow Security. IEEE Journal on Selected Areas in Communications, 21(1):5-19, January 2003.