Security Fundamentals: Models

Trent Jaeger
January 12, 2004

Trent Jaeger -- Background

- Graduated from UM with PhD
  - Flexible Control of Downloaded Executable Content
    - Research Thread That Led to Java 2 Security Model
  - IBM TJ Watson Research Center 1996-Present
      - L4 microkernel-based systems: L4Linux, Lava, JavaOS, SawMill
      - Security policy: graphical access control, constraint models
    - Systems Security Research 2001-Present
      - Linux Security: based on the Linux Security Modules framework
      - Linux Security Analysis Project: source code and policy analysis
        - www.research.ibm.com/vali
  - Conference chair, programming chair of ACM SACMAT, CCS
  - PC member of IEEE S&P, USENIX, ESORICS, etc.

Access Control

- Determine whether a principal can perform a requested operation on a target object
- Principal: user, process, etc.
- Operation: read, write, etc.
- Object: file, tuple, etc.
- Lampson defined the familiar access matrix and its two interpretations ACLs and capabilities [Lampson70]

Why are we still talking about access control?

- An access control policy is a specification for an access decision function
- The policy aims to achieve
  - Permit the principal’s intended function (availability)
  - Ensure security properties are met (integrity, confidentiality)
- Limit to “Least Privilege,” Protect system integrity, Prevent unauthorized leakage, etc.
- Also known as ‘constraints’
  - Enable administration of a changeable system (simplicity)

“Simple” example

- Prof Alice manages access to course objects
  - Assign access to individual (principal: Bob)
  - Assign access to aggregate (course-students)
  - Assign access to relation (students(course))
  - Assign students to project groups (student(course, project, group))
- Prof Alice wants certain guarantees
  - Students cannot modify objects written by Prof Alice
  - Students cannot read/modify objects of other groups
- Prof Alice must be able to maintain access policy
  - Ensure that individual rights do not violate guarantees
  - However, exceptions are possible – students may distribute their results from previous assignments for an exam

Access Control is Hard Because

- Access control requirements are domain-specific
  - Generic approaches over-generalize
- Access control requirements can change
  - Anyone could be an administrator
- The Safety Problem [HRU76]
  - Can only know what is leaked right now
- Access is fail-safe, but Constraints are not
  - And constraints must restrict all future states
**Compare to Other CS Problems**

- Processor design
  - Hard, but can get some smart people together to construct one, fixed, testable design
- Network protocol design
  - TCP: A small number of control parameters necessary to manage all reasonable options, within a layered architecture
  - Constraints, such as DDoS, are ad hoc
- Software design
  - Specific goals in mind to achieve function, constraints are ad hoc

**Reference Monitor Model**

- Entry Points
  - Access Hook
  - Security-sensitive Operations
  - Yes/No
- System Interface
  - Monitor Policy
  - Authorize Request?
- Monitor
  - Access Hook
  - Security-sensitive Operation

**Reference Monitor Components**

- Interface
  - Where to make access control decisions (mediation)
  - Which access control decisions to make (authorization)
  - Linux Security Modules interface
- Decision function
  - Compute decision based on request and policy
  - E.g., SELinux, LIDS, DTE, etc. modules
- Policy – our focus today
  - How to represent access control policy
  - Main mechanism issue – find mechanism to enable verification that policy achieves function and meets security guarantees

**Access Control Models**

- Basic Access Matrix
  - UNIX, ACL, various capability systems
- Aggregated Access Matrix
  - TE, RBAC, groups and attributes, parameterized
- Plus Domain Transitions
  - DTE, SELinux, Java
- Lattice Access Control Models
  - Bell-LaPadula, Biba, Denning
- Predicate Models
  - ASL, OASIS, domain-specific models, many others
- Safety Models
  - Take-grant, Schematic Protection Model, Typed Access Matrix

**Need for Aggregate Models (RBAC)**

- Practical ease of specification
  - Abstraction for users, permissions, constraints, administration
- Natural access control aggregations – based on organizational roles
  - As new employees join, their permission assignments are determined by their job
  - Permission assignment is largely static
- Central control and maintenance of access rights
- Flexible enough to enforce
  - least privilege, separation of duties, etc.

**Type Enforcement [BoebertKain84]**

- User
  - Type (Subject)
  - Permission Assignment
  - Subject Type Can Access Object Type
  - To Perform Operations On Objects
  - Type (Object)
  - Object
  - Object
Group and Attributes

Role-based Access Control

Role vs. Types Data Structures

Role-based Access Control Model

RBAC Family of Models

Hierarchies and Constraints

- Role hierarchy
  - Problem: does organizational hierarchy correspond to a permission inheritance hierarchy?
  - Problem: do organizational roles make sense for building hierarchies?
- Constraints
  - Problem: constraints apply to all states, so they require a predicate calculus in general
  - Problem: Only certain types of constraints can effectively be administered? Mutual exclusion, separation of duty, cardinality, etc.
- Conflicts
  - May find other concepts useful for resolving conflicts between constraints and hierarchies/assignments
Administration

- Discretionary Access Control
  - Users (typically object owner) can decide permission assignments
- Mandatory Access Control
  - System administrator decides on permission assignments
- Flexible Administrative Management
  - Access control models can be used to express administrative privileges

Does RBAC Achieve Its Goals?

- Practical ease of specification
  - Clear base model – need more help for constraints, admin
- Natural access control aggregations – based on organizational roles
  - In some cases, but not clear that organizational roles help with permission assignment – particularly with inheritance
- Central control and maintenance of access rights
  - Central view is a selling feature of products, but a single view of all can be complex (layering?)
- Flexible enough to enforce
  - Flexible access control expression, but difficult to determine if we enforce our security goals (constraints)

RBAC Products

- SUN Solaris
- Sybase SQL Server
- BMC INCONTROL for Security Management
- Systor Security Administration Manager
- Tivoli TME Security Management
- Computer Associates Protect IT
- Siemens rbacDirX

Safety Problem [HRU76]

- Determine if an unauthorized permission is leaked given
  - An initial set of permissions and
  - An access control system, mainly administrative operations
- For a traditional approach, the safety problem is undecidable
  - Access matrix model with multi-operational commands
  - Main culprit is create – create object/subject with own rights
  - Prove reduction of a Turing machine to the multi-operational access matrix system
- Result led to
  - Safe, but limited models: take-grant, schematic protection model, typed access matrix model
  - Further support for models in which the constraints are implicit in the model – e.g., lattice models
  - Check safety on each policy change – constraint approach of RBAC

Lattice Access Control Models

- Subjects and Objects have security levels and optional categories
- Confidentiality Policy (e.g., Bell-LaPadula)
  - Simple property: may read only if the subject’s security level dominates the object’s security level (read-down)
  - “*”-property: may write only if the subject’s security level is dominated by the object’s security level (write-up)
  - Tranquility property: may not change the security level of an object concurrent to its use
- Integrity Policy
  - Biba is the dual of BLP for integrity

Security Levels and Policies
Purpose of BLP and Biba

- **BLP**
  - Prevent Trojan horses from leaking information to lower security levels
  - Mandatory access control and implicit constraints
- **Biba**
  - Prevent low integrity information flows to higher integrity processes
  - E.g., code, configuration, user requests, buffer overflows
  - Categories/Compartments for separation within levels
  - Safety is implicit in the model
  - No additional constraints are needed to express security guarantees

Denning’s Lattice Model

- Formalizes information flow models
  - FM = (N, P, SC, ≤)
- Shows that the information flow model instances form a lattice
  - (SC, ≤) is a partial ordered set,
  - SC is finite,
  - SC has a lower bound,
  - and ≤ is a lub operator
- Implicit and explicit information flows
  - Semantics for verifying that a configuration is secure
  - Static and dynamic binding considered
  - Biba and BLP are among the simplest models of this type

Implicit and explicit flows

- **Explicit**
  - Direct transfer to b from a (e.g., \( b = a \))
- **Implicit**
  - Where value of b may depend on value of a indirectly (e.g., if \( a = 0 \), then \( b = c \))
- Model covers all programs
  - Statement S
  - Sequence S1, S2
  - Conditional c: S1, …, Sm
- Implicit flows only occur in conditionals

Semantics

- Program is secure if:
  - Explicit flow from S is secure
  - Explicit flow of all statements in a sequence are secure (e.g., S1; S2)
  - Conditional c:S1, …, Sm is secure if:
    - The explicit flows of all statements S1, …, Sm are secure
    - The implicit flows between c and the objects in Si are secure

Static and Dynamic Binding

- **Static binding**
  - Security class of an object is fixed
  - This is the case for BLP and Biba
  - This is the case for most system models
- **Dynamic binding**
  - Security class of an object can change
  - For \( b = a \), then the security class of b is \( b ≤ a \)
  - Rare approach

Model Examination

- **Static binding**
  - BLP and Biba check security property at runtime
    - Guarantees security of all implicit and explicit flows
    - If \( a \), then \( b ≤ c \) means that a \( b \), so SC(b) must be dominated by SC(b) in BLP
    - Otherwise, information leakage
  - Data Mark Machine
    - Security class depends on location in program
    - Combine with condition class on entering condition
    - Data flow captured because c \( b \) is checked in condition and \( b ≤ a \) is checked outside and \( ≤ \) is transitive
    - Does not hold for dynamic binding
**Model Examination**

- Certification Mechanism
  - Static check eliminates covert channels
  - Limits
  - Language defect could miss a check (buffer overflow)
  - Hardware malfunction
- Approach
  - Verify information flow within a statement
    - \( d = a + b \); \( d \) must dominate
  - Set statement security level \( S = d \)
  - Statement sequence \( S = S_1 \rightarrow S_2 \) must be able to flow to greatest lower bound
  - Verify \( c \) dominates \( d_1, \ldots, d_n \) for implicit flow

**Verification Example**

\[
\begin{array}{c|c|c|c}
PS & MS & TS & NS \\
\hline
PS & MS & TS & NS \\
\hline
PS & MS & TS & NS \\
\hline
PS & MS & TS & NS \\
\end{array}
\]

**Dynamic Binding**

- Definition
  - May change the security class of an object that removes it from view – could leak covertly!
- Dynamic Data Mark
  - Combine program class with actual data flow class
  - Static binding mechanism doesn’t work
    - Paper example shows that dynamic class updating does not account for implicit flows
    - Local object solution ensures correct classification, but seems functionally limiting
  - Static analysis to insert classification statements to cover implicit flows
- High Water Mark
  - Class is raised to prevent leakage \( a < b \) whenever \( b = a \)
  - Similar flaw to Dynamic Data Mark – and similar fix

**Lattice Model Features**

- Safe policies w/o constraints
  - Bell-LaPadula is consistent with military security policy
  - Biba is not consistent with any practical integrity policy
- Problem: Downgraders/Upgraders
  - Changing the security label of an object requires sanitization
    - Remove secrets for confidentiality downgrade
    - Remove low integrity data for integrity upgrade
  - Sanitization is ad hoc
  - Sanitizer must be of trusted (more stringent requirement than simply being at the higher security class)
- Lattice model effectiveness is limited by the number of downgraders

**Recent Information Flow Policies**

- Information flow in programming languages
  - Andrew Myers (Cornell), Steve Zdancewic (U Penn), Barbara Liskov (MIT)
- Decentralized Label Model
  - Flow policies of principals implemented by labeled data
    - System guarantees to enforce all policies simultaneously
    - Declassification enables removal of restrictions “where appropriate”
    - Determined by programmer
    - Enables static checking
  - Advantage: assume programmer is trusted to describe security policies
    - Programmers can prevent Trojan horse leakage in downgrade

**Decentralized Labels**

- Labels have owners and readers
  - \( L = \{ o_1: r_1, r_2; o_2: r_2, r_3 \} \)
  - Effective readers of \( L \) (b) because only it can read from \( o_1 \) and \( o_2 \)
- Static binding
  - Label of a value is forgotten when assigned to a variable
- Relabeling semantics
  - A new label contains the owners of the old, but fewer readers
- Declassification semantics
  - An authority for an owner can add the owner or readers of the owner
- Label join/meet semantics
  - \( \text{Join (e.g., multiply 2 numbers):} \) Union owners and intersect readers
  - \( \text{Meet (dual of join):} \) Intersect owners and union readers
Other Models

- Plus Domain Transitions
  - DTE, SELinux, Java
- Predicate Models
  - ASL, OASIS, domain-specific models, many others
- Safety Models
  - Take-grant, Schematic Protection Model, Typed Access Matrix

References

- Bell-LaPadula: MITRE tech report – hard to find
- Biba, K. Tech report – hard to find