Security Models

David Aspinall
School of Informatics,
University of Edinburgh.
27th January 2005

http://www.infomatics.ed.ac.uk/teaching/courses/cs

Contents

- Access and information ow
- Access control mechanisms
- Security levels
- Security models
  - Bell-LaPadula
  - Harrison-Ruzzo-Ullman
  - Clark-Wilson
- Implementations
- Issues of trust

Security policies describe rules about who is allowed to do what. Security models let you formulate those rules, precisely, and explain how the system makes authorization choices.

A security policy is a description of security requirements for a system. A security model is a way of formalizing security policies. Two paradigms:

- access control: a guard controls whether a principal (the subject) is allowed access to a resource (the object).
- information flow control: dual notion sometimes used when confidentiality is the primary concern. A guard controls whether information may ow from a resource to a principal.

Ownership and identity

- Who is allowed to set the security policy? A resource may have a distinguished owner who controls access to it, or the resource may be controlled by a system-wide policy. Terminology:
  - discretionary access control (DAC): owners decide case-by-case
  - mandatory access control (MAC): policy determined system-wide
- Owners of resources typically appear as principals in the system: subjects themselves subject to access controls. BLP does not (directly) take into account operations for modification of access controls (like chown in Windows NT), nor explain when such operations are safe.
- The identity of subjects is also a flexible notion: sometimes we may want to allow identity changes during operations, for example, assigning identities to executing programs and basing access controls on that identity (SU ID programs in Unix). Again, this doesn’t fit BLP.

Access control structures

- How are access control rights defined? Many different schemes, but ultimately can be modelled thus:
  - A set S of subjects, a set O of objects
  - A set A of operations (modelled by access rights), we’ll consider A = {exec, read, append, write}.
  - An access control matrix
    \[ M = (M_{so}) \in S \times O \]
    where each entry \( M_{so} \) defines rights for \( s \) to access \( o \).
- Example matrix for \( S = \{Alice, Bob\} \) and three objects:

<table>
<thead>
<tr>
<th>Bob</th>
<th>Alice</th>
</tr>
</thead>
<tbody>
<tr>
<td>(exec, read)</td>
<td>(exec)</td>
</tr>
<tr>
<td>(exec)</td>
<td>(exec, read, write)</td>
</tr>
</tbody>
</table>

Access operations

- We can consider some fundamental access modes. Typically:
  - observe examine contents of an object
  - alter change contents of an object
- Next we define access rights and their profiles:

<table>
<thead>
<tr>
<th>Access Right</th>
<th>exec</th>
<th>read</th>
<th>append</th>
<th>write</th>
</tr>
</thead>
<tbody>
<tr>
<td>observe</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>alter</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

These are the access rights of the influential Bell-LaPadula (BLP) model. Access rights are the model’s level of granularity for defining security policy. Each real operation requires particular access rights.

- The profiles shown (and names of rights) differ between systems, and even between different kinds of subject on the same system. For example, some systems have a delete. In Unix, exec for directories indicates the ability to read the directory.
Implementing $M$ directly is impractical, so different schemes are used. Two complementary possibilities, either using capabilities (store $M$ by rows) or using access control lists (store $M$ by columns).

A capability is an unforgeable token that specifies a subject’s access rights. Pros: can pass around capabilities; good fit with discr. AC. Cons: difficult to revoke, or find out who has, access to a particular resource (must examine all capabilities). Interest reinstated recently with distributed and mobile computation.

An access control list (ACL) stores the access rights to an object with the object itself. Pros: good fit with object biased OSes. Cons: difficult to revoke, or find out, permissions of a particular subject (must search all ACLs).

Security levels

- Ideas for multi-level security (MLS) systems originated in the military. A security level is another kind of security attribute used to label subjects and objects, to describe security policies.
- Security levels (like protection rings) come with an ordering. Typical example: unclassified $\leq$ confidential $\leq$ secret $\leq$ topsecret.
- The ordering can express policies like “no write-down” which means that a high-level subject cannot write down to a low-level object. For example, a user with confidential clearance cannot write an unclassified file (it might contain confidential information which was read earlier).
- In practice, more flexible security levels are needed than the simple linearly-ordered document classifications shown above. Typically we want to introduce categorizations as well, for example, describing departments or divisions in an organization. This means that individual levels may not be comparable.

Security lattices

- A lattice is a set $L$ equipped with a partial ordering $\leq$ such every two elements $a, b \in L$ has a least upper bound $a \lor b$ and a greatest lower bound $a \land b$. A finite lattice must have top and bottom elements.
- In security, if $a \leq b$, we say that $b$ dominates $a$. The bottom level dominated by all others is system low; the top level which dominates all others is system high.
- Lattices were proposed to model MLS policies because they allow an ordering of security levels such that:
  - Given two objects at different levels $a$ and $b$, there is a minimal security level $a \lor b$ needed to access both $a$ and $b$;
  - Given two subjects at different levels $a$ and $b$, there is a maximal security level $a \land b$ for an object which must be readable by both.

Intermediate controls

Apart from subject-biased or object-biased representation of access control policy, there are many mechanisms using intermediate layers:

- **Groups**: assign subjects to one or more groups, and define common permissions group-wise. (e.g., Unix ACL is perms for subject, group, other).
- **Negative permissions**: often a natural way of defining the security policy, to say who should not have access to some resource. A negative permission may override a permission obtained from a group permission, for example.
- **Protection rings**: assign subjects and objects to inhabit one ring in a concentric series numbered 0,1,2,3. If a subject tries to access an object, compare the rings they occupy (e.g., might only allow “outward” accesses; ring number 0 has the highest protection). Used in Oles and processor hardware, for integrity assurances.

- Privileges: instead of considering objects, privileges focus on operations that subjects may perform. Privileges might include activities like syscall, mail access, network access (typically OS-defined). These activities may be considered as a higher-level of access controls.
- Role-based access control (RBAC): instead of considering subjects, RBAC focuses on roles that a subject (user) may perform. A role is defined as a collection of procedures at the application level. Users may have several different roles, and may change roles. We now have a hierarchy of access control mechanisms:

  **Role** A collection of procedures assigned to users.

  **Procedure** A controlled operation with a finer grained meaning than just observe and alter. Operates on data of particular datatypes.

  **Datatype** Programming-language level abstraction used to enforce integrity: each object belongs to a certain datatype and may only be accessed through methods defined for that type.

An Example Lattice [Gol99]

A standard construction is to take a set of classifications $H$, with a linear ordering $\leq$, together with a set $C$ of categories. Define a compartment as a set of categories, and then a security level as a pair $(h,c)$ where $h \in H$ and $c \in C$. Then the ordering $(h_1,c_1) \leq (h_2,c_2) \iff h_1 \leq h_2, c_1 \leq c_2$ defines a lattice.
Bell-LaPadula Model [BL96]

- **BLP (1973)** is state machine model designed to model confidentiality. Permissions use an AC matrix and **security levels**. The **security policy** prevents information flowing from a high level to a lower level.
- Assume subjects S, objects O, accesses A, and levels (L, s) as before.
- The state set \( B \times M \times F \) captures current permissions and subjects accessing objects. It has these parts:
  - \( B = p(S \times O \times A) \) is the set of all possible current accesses. An element \( b \in B \) is a set of tuples \((s,o,a)\) indicating that \( s \) is currently performing operation \( a \) on object \( o \).
  - \( M \) is the set of access permission matrices \( M = (M_{st})_{s \in S; t \in T} \)
  - \( F \subseteq L^3 \times L^3 \times L^2 \) is the set of security level assignments. An element \( f \in F \) is a triple \((f_s, f_c, f_o)\) where \( f_s: S \rightarrow L \) gives the **maximal security level** each subject can have; \( f_c: S \rightarrow L \) gives the **current security level** of each subject (state \( f_c \leq f_s \)), and \( f_o: O \rightarrow L \) gives the **classification** of all objects.

Basic security theorem

- A transition from state \( v_1 \) to \( v_2 \) is **secure** simply if both states \( v_1 \) and \( v_2 \) are secure.
- This leads to a rather simple and general theorem:
  
  **Basic security theorem.** If all state transitions in a system are secure and the initial state of the system is secure, then every subsequent state is also secure.

  (NB: this follows immediately by induction on the length of transitions, it has nothing to do with the properties of BLP!)

- So a system is **secure** according to BLP if the initial state of the system is secure, and if all transitions it may execute preserve security.
- The point is that we can reduce the checking system for all possible inputs, to checking that each kind of state transition which may occur preserves security. Of course, to do this we need a concrete instance of the model which describes the transitions that may happen.

Security properties in BLP

Consider a BLP state \((b, M, f)\) where \( b \) is the set of current accesses.

**ss-property** For each access \((s,o,a) \in b\) where \( a \in \{\text{read}, \text{write}\}, f_o(o) \leq f_s(s) \) (no read-up).

**s-property** For each access \((s,o,a) \in b\) where \( a \in \{\text{append}, \text{write}\}, f_c(s) \leq f_o(o) \) (no write-down). Furthermore, for such an access, we must have \( f_o(o') \leq f_o(o) \) for all \( o' \) with \((s,o',a') \in b\) and \( a' \in \{\text{read}, \text{write}\} \) (\( o \) must dominate any other object \( o' \) it can read).

These two properties describe the mandatory access control policy. The access control matrix \( M \) allows DAC as well.

**ds-property** For each access \((s,o,a) \in b\), we have that \( a \in M_{st} \) (discretionary access controls are obeyed).

The state \((b, M, f)\) is **secure** if these three properties are satisfied. (Note that BLP’s notion of security is entirely captured in the current state).

Current clearance level

- Unfortunately, the **s-property** means that there is no way for a high-level subject to send messages to a low-level subject.
- There are two ways out:
  1. temporarily downgrade a high-level subject, which is why the model includes the current **clearance level** setting \( f_c \), or
  2. identify a set of **trusted subjects** allowed to violate the **s-property**.

  Approach 1 works because BLP considers the current state to describe exactly what each subject can know. So if a subject (think of it as a process) is downgraded, it cannot access higher-level material, so may safely write at any lower level than its maximum.

- Considering subjects who are people with high-level clearances, the second approach is more realistic: they may be trusted to violate the property required of the model, for example, when deciding to make some part of a secret document available at a lower level.

Harrison-Ruzzo-Ullman Model [HRU76]

- The HRU model (1976) captures the notion of changing access rights, and adding and removing subjects and objects.
- It defines **authorization systems** using a language of primitive operations for manipulating a state \((S, O, M)\). Commands are guarded sequences of operations, which model, for example, the creation of a file by a subject. These are the state transitions of the model.

- A security policy in HRU can regulate the allocation of access rights. The security property for a state \( M \) expresses that no reachable state \( M' \) can allow a command to add an illegal right \( r \) to \( M' \).

- Verifying security of an arbitrary \( M \) and \( r \) turns out to be **undecidable**. It is **decidable** if commands contain a single operation, or if \( S \) is finite.

- Goldmann (Gold99) says this proximity to undecidability suggests that we should be careful to limit (conceptual) complexity in our formal models, as well as in implementations.
Clark-Wilson model [CW87]

- The Clark-Wilson model (1987) attempts to capture security requirements for commercial applications. The authors contrasted against BLP focusing on integrity, whereas BLP addresses confidentiality.
- Clark and Wilson take a broad view of integrity, making the distinction: internal consistency property of system state, enforced by system; external consistency relation with real-world, enforced externally.
- They suggest two general mechanisms to enforce integrity: well-formed transactions: data accessed only by specified programs; separation of duties: users must collaborate to manipulate data.

The first of these is familiar in current ideas of ADTs and OOP.
The second principle is also well known (for example, using separate people to implement, test, and certify a system). Separation of duties is also well known from ideas of book-keeping. Apart from increased confidence through double-checking, it means that to break a system, collusion between more than one person is needed.

Implementations

- Security models with MACs were implemented in Multics and some commercial versions of Unix (e.g., defunct Data General DG/UX).
- In 2000, the NSA introduced Security-Enhanced Linux, open-source kernel patch and utilities for Linux, based on their Flask architecture:
  - A security server provides policy decisions to object managers (managing processes, files, and sockets). Policies describe security contexts; the security server maps integer security identifiers (SIDs) to security contexts.
  - Allows Type Enforcement, RBAC, MLS to be specified in policies.
  - A program checks policy compiles a policy into a binary form read at boot-up; a kernel call is provided to update the policy at runtime.
- The TrustedBSD project (http://www.trustedbsd.org) is a related effort for FreeBSD which provides MAC support and a port of the NSA Linux Flask architecture.

Where does the trust lie?

- We’ve seen trusted principals both in BLP and in protocols. Something trusted is something that can hurt you, because you trust it blindly.
- By contrast, a trustworthy something is something that you have decided to trust, hopefully with good grounds, for some purpose.
- A design principle is to limit the trusted portion. Perhaps with a small security kernel; if it enforces our security policy, then any code running on top is automatically secure, and won’t have to be trusted.
- Unfortunately, it’s not possible or meaningful to express high-level security requirements at the lowest level. Instead, the onion model of protection mechanisms describes each layer (apps, middleware, OS, kernel, hardware) as trusting the layer below it, and implementing its own security concepts, controlling access to the layer below.
- The trusted code from top to bottom (on the security critical path) is known as the trusted computing base (TCB).

References

See Chapters 3–5 of Gollmann [Gol99] for a good overview of access control and security models. For further reading, see Chapters 1–2 of Anderson [And01] and Parts 2–3 of Bishop [Bir03].