CSE 127: Computer Security
Security Models
Kirill Levchenko

October 9, 2014
Saltzer & Schroeder’s Security Design Principles

- Economy of mechanism
- Fail-safe defaults
- Complete mediation
- Open design
- Separation of privilege
- Least privilege
- Least common mechanism
- Psychological acceptability
- Work factor
- Compromise recording
Security Model

Abstraction used by the security policy and mechanism
Security Model

- **Subjects**: Individuals or processes acting on their behalf
- **Objects**: Protected information or function
  - Objects often also include subjects
- **Subjects operate on objects**
  - System mediates and facilitates subject-object interaction
- **Policy**: what *action* is *subject* allowed to do with *object*?
  - And who can introduce new subjects and objects into system?
- Nearly all security models built on this idea
Access Control Matrix

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{allowed}</td>
</tr>
<tr>
<td></td>
<td>{actions}</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Access Control Matrix

<table>
<thead>
<tr>
<th></th>
<th>Broccoli</th>
<th>Fruit from Tree of Life</th>
<th>Fruit from Tree of Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adam</td>
<td>{see, eat}</td>
<td>{see, eat}</td>
<td>{see}</td>
</tr>
<tr>
<td>Eve</td>
<td>{see, eat}</td>
<td>{see, eat}</td>
<td>{see}</td>
</tr>
</tbody>
</table>
Access Control Lists (ACLs)

- An access control list of an object identifies which subjects can access the object and what they are allowed to do.

- ACLs are object-centric: access control is associated with objects in the system.

- Each access to object is checked against object’s ACL.

- Example: guest list at a night club.
a) “List-oriented” implementations, in which the guard holds a list of identifiers of authorized users, and the user carries a unique unforgeable identifier that must appear on the guard’s list for access to be permitted. A store clerk checking a list of credit customers is an example of a list-oriented implementation in practice. The individual might use his driver’s license as a unique unforgeable identifier.
A capability grants a subject permission to perform a certain action
• Unforgeable
• Usually transferrable

Capabilities are subject-centric: access control is associated with subjects in the system

Example: car key
Capabilities

b) “Ticket-oriented” implementations, in which the guard holds the description of a single identifier, and each user has a collection of unforgeable identifiers, or tickets\(^\text{17}\), corresponding to the objects to which he has been authorized access. A locked door that opens with a key is probably the most common example of a ticket-oriented mechanism; the guard is implemented as the hardware of the lock, and the matching key is the (presumably) unforgeable authorizing identifier.
## ACLs & Capabilities

- Columns of the Access Control Matrix define objects’ *ACLs*
- Rows of the Access Control Matrix define users’ *capabilities*

**Diagram:**

- **ACL**
  - Objects
  - Capabilities
  - Allowed actions

- **Subjects**
Unix Security Model

- **Subjects**: Users
- **Objects**: Files and directories
- **Actions**: read, write, execute
  - Execute a file means can call `exec()` on file
  - Directory “execute” means user can traverse it
- Unix is a simplified ACL system
  - Arbitrary ACLs not possible in traditional Unix
  - Modern Unix operating systems allow arbitrary ACLs
Unix Superuser

- **Superuser** allowed to do anything that is possible
- Called `root` and mapped to user id 0
- A superuser is a *role* rather than a particular user
- System administrators assume the superuser role to perform privileged actions
  - Good practice to assume superuser role only when necessary
Permissions

- Each file has an **owner** and a **group**
  - **Group**: named set of users

- File permissions specify what owner, group, and other (neither owner nor group) is allowed (read, write, exec)

```
-rwxrwxr--
```

- Other’s permissions
- Group’s permissions
- Owner’s permissions
Permissions

- Only owner and superuser can change permissions
- Only superuser can change owner
- Only owner and superuser can change group
  - Owner can only change to group she belongs to
- User’s allowed actions on file are:
  - Owner’s permissions if the user is the owner,
  - Group’s permissions if the user is in the group,
  - Other’s permissions otherwise
Permissions

- Users interact with system via processes acting on their behalf
- When you interact with system via terminal, command shell acts on your behalf
- Each process is associated with a user
Processes and User

- **ruid**: real user id
  - Inherited from parent process

- **euid**: effective user id
  - Inherited from parent process unless setuid bit set
  - Permissions checked against euid

- **suid**: saved user id
  - Can reset euid to suid
Processes and User

- getuid()
  - returns ruid

- setuid()
  - if root: set ruid, euid, suid
  - otherwise (POSIX): set euid to ruid or suid
  - otherwise (BSD): set ruid, euid, suid to ruid
Processes and User

- `geteuid()`
  - returns `euid`

- `seteuid()`
  - if `root`: set `euid`
  - otherwise: set `euid` to `ruid` or `suid`
Unix Groups

- A user may belong to several groups
- Processes have real, effective, and saved group id
  - `getgid()`, `setgid()`, `getegid()`, and `setegid()`.
- Processes also have *supplementary* groups
  - File access checked against all supplementary groups
- Processes also have *supplementary* groups
- How group ids affect what a process can do varies
Login

- When user connects to system via physical terminal, system runs `login` process as `root` to start session
  - Authenticates user using username and password
  - Changes its user id and group id to that of user
    - This is possible because superuser allowed to do anything
  - Executes user’s shell

- `sshd` performs similar actions

- Critical: dropping privileges from `root` to regular user
Changing Privilege

- Superuser can drop privilege to become regular user
- Want way to elevate privilege in controlled manner
- How?
It is easier to control who may use the descriptor, and a common scheme is to introduce an additional bit in the processor state. This bit is called the privileged state bit. All attempts to load the descriptor register are checked against the value of the privileged state bit; the privileged state bit must be ON for the register to be changed. One program (named the supervisor—program S in Fig. 1) runs with the privileged state bit ON, and controls the simulation of the virtual processors for the other programs. All that is needed to make the scheme complete is to ensure that the privileged state bit cannot be changed by the user programs except, perhaps, by an instruction that simultaneously transfers control to the supervisor program at a planned entry location.

The third mechanism protects the privileged state bit. It allows an executing program identified by the privileged state bit being OFF (a user program) to perform the single operation “turn privileged state bit ON and transfer to the supervisor program.” An executing program identified by the privileged state bit being ON is allowed to turn the bit OFF. This third mechanism is an embryonic form of the sophisticated protection mechanisms required to implement protected subsystems.
Elevating Privilege

- Executable files have a setuid and setgid bit
- If setuid is set, files is executed with privilege of owner
  - ruid is that of executing user, euid and suid that of owner
- The setgid bit does same for group
  - But supplementary groups remain that of executing user
- The passwd command is setuid and owned by root
  - Executes as superuser (root) — why?
PostgreSQL Security Model

- A relational database is a set of tables (relations) and a language for defining, querying, and modifying the tables.
- SQL is the standard language for relational databases.
- PostgreSQL implements SQL with modifications:
  - PostgreSQL is a relational database management system (RDBMS).
  - Each RDBMS implements standard slightly differently.
PostgreSQL Security Model

- **Subjects:** database users and groups (roles)
- **Objects:** databases, tables, views, functions, etc.
- **Actions:** use, query, update, insert, delete, etc.
- PostgreSQL also has superuser who can do anything
- SQL access control is ACL-based
GRANT Privilege

- Owner can grant other users or roles the ability to grant an access privilege to others
  - GRANT ... WITH GRANT OPTION;

- Is the ability to grant option also granted?
Access Control Checks

- Issuing a SELECT on a table requires SELECT privilege on same table
  - UPDATE requires UPDATE, and so on
- Joins and subqueries will require SELECT privileges
- User has privileges that are the union of her own and privileges of all groups to which she belongs
Views

- A **view** is a saved query whose results can be accessed as if they are a table
  - The underlying query is added as a subquery and re-executed
  - Contents of view may not be the same between queries
    - Depend on the contents of the underlying tables, which may change

- Views also have access controls
  - Can grant SELECT, UPDATE, DELETE, etc.
  - Data modification queries handled differently by each RDBMS
View Access Control

- Creator of view must be able to query underlying table
- When user queries a view, system checks her access privileges for the view only
- View is then expanded into its defining query and accessed with the privileges of view owner
- View allows controlled access to protected data
  - A kind of privilege elevation
Functions

- PostgreSQL supports user-defined functions
- User must have USAGE permission to use function
- Function executed with invoking user’s privileges
- SECURITY DEFINER functions execute with owner privileges
AppArmor Security Model

- AppArmor is an additional security mechanism available on some Unix systems
- **Subjects:** processes
- **Objects:** files, system functions
- **Actions:** read, write, execute (several flavors) and several types of capabilities (e.g. network)