CSE 409, Fall 2011, Rob Johnson, http://www.cs.stonybrook.edu/~rob/teaching/cse409-fa11/
Alin Tomescu, September 16th, 2011

More access control lists

Windows ACLs and tokens

Motivation: Sometimes when you launch a program, you don't want to give it all of your privileges. W

Windows has tokens, a summary of your processes access rights.

A token will have a user ID and some group IDs with the following information associated for each user ID and group ID:

- ignore (this credential does not contribute to your ACL decision at all)
- normal (this credential can be matched against a positive ACL or a deny ACL)
- deny only (it can only be used to match ACL deny rules)

Windows has a system of API calls to allow you to restrict program access:

- `CreateRestrictedToken` (http://msdn.microsoft.com/en-us/library/aa446583(v=VS.85).aspx)
- `AdjustTokenPrivileges` (http://msdn.microsoft.com/en-us/library/aa375202(v=VS.85).aspx)

You can get a handle to your initial security token and then create a restricted token with deny attributes and deny conversions by changing some entries in the token from IGNORE or NORMAL to DENY or by adding new DENY entries.

ACL monotonicity: Adding entries to ACL only increases set of processes that can access file

- Windows does not have ACL monotonicity since it has deny entries.

Process token monotonicity: Adding credentials to a process can only increase its access (also, groups added to the process GID list should only increase its access)

- Again not true for Windows since adding a group to a process can restrict its access to files and directories which have deny entries for that group.

Access control matrices

Access control matrix has subjects along the rows and objects on the columns.

Objects / Subjects	File 1	File 2	Log file	Some directory
user1	rw		append	look up
user2		w		list files
user3				
user4				

An access control matrix can be defined at any level you want. For instance, if you have a log file, you can grant a user the ability to append to it.

If you read an ACM by columns, you get an ACL in each column. Ultimately, any scheme of representing permissions will bottle down to an ACM. The problem is of minimizing the size of this matrix and having the ability to easily edit it.

CSE 409, Fall 2011, Rob Johnson, http://www.cs.stonybrook.edu/~rob/teaching/cse409-fa11/
Alin Tomescu, September 16th, 2011

Discretionary versus Mandatory Access Control (DAC vs. MAC)

In **Discretionary access control (DAC) systems**, the user of the system can modify ACM entries in some ways. Not arbitrarily though. I can't go and grab anyone's files when I feel like it. The system imposes some restrictions.

In **Mandatory access control (MAC) systems**, access control is enforced by system policy. **Example:** Imagine you are working on a computer at the NSA and you are editing a top secret document. Do you get to decide whether the new saved file is made public? No. The system is going to force you to label the file you write out as top-secret, whether you like it or not.

The Bell-LaPadula access control model

The **Bell-LaPadaula (BLP)** access control model defines **security labels** (top-secret, secret, public) for objects and **clearances** (JFK, Aliens) for subjects. In addition objects and subjects can be assigned **compartments.**

A **security level** for an object is the object's security label plus its set of compartments.

You might have a **compartment** about *the JFK assassination* and one about *aliens*. Even if you have top-level clearance you won't be able to see documents which are in the JFK compartment unless you are also in the JFK compartment.

Some examples:

File security level	Your security level	Access granted?
Top secret	Top secret	**Yes**
Top secret, { Aliens }	Top secret	**No**, you are not in the Aliens compartment
Top secret, { JFK }	Top secret, { JFK }	**Yes**, you have top-secret clearance and JFK access
Top secret, { Aliens }	Top secret, { JFK }	**No**, you are not in the Aliens compartment
Top secret, { Aliens, JFK }	Top secret, { Aliens, UFOs }	**No**, you are not in the JFK compartment

How is access control determined?

In general, users are only allowed to **read down** and to **write up** relative to his security level. For instance, if a user has only secret clearance, he can only read (down) documents that are either public or secret. He can never read top-secret documents. Similarly, he could never write to files below his security level because he could expose secret information by doing so. Thus, he is only allowed to write (up) to files that are either secret or top-secret. *(See any problems with that?)*

Formally, **read access** to file f is granted to user x if:

$$security\_level_x \geq security\_level_f \; \boldsymbol{AND} \; compartments_f \subseteq compartments_x$$

If we were to apply this rule for **writing**, things would go bad. Imagine there's a file `index.html` who is labeled as public, and is accessible at http://www.nsa.gov. Then processes that could read all the top-secret files could write all the information to the `index.html` file, exposing the truth about aliens, UFOs and who killed JFK to the world. **NO NO!**

Therefore, **write access** file f is granted to user x if:

$$security\_level_x \leq security\_level_f \; \boldsymbol{AND} \; compartments_x \subseteq compartments_f$$

CSE 409, Fall 2011, Rob Johnson, http://www.cs.stonybrook.edu/~rob/teaching/cse409-fa11/
Alin Tomescu, September 16th, 2011

The reason behind these rules is to **prevent information flow from higher security levels to lower security levels.** It protects **confidentiality**, as illustrated below:

$$[top\ secret\ file] \underset{file_{level} \leq p_{level}}{\overset{secret\ info}{===\Rightarrow}} [process\ p] \underset{\boldsymbol{p_{level} \geq public\ file_{level}}}{\overset{secret\ info}{===\Rightarrow}} \rightarrow [public\ file]$$

This is why, when $process_{level} \geq file_{level}$, the process is not allowed by the Bell-LaPadula scheme to write to $file$.

But also we have to maintain **integrity.** With Bell-LaPadula, the following **bad** flow is possible:

$$[some\ secret\ file_1] \underset{file_{level} \leq p_{level}}{\overset{deceiving\ info}{===\Rightarrow}} [process\ p] \underset{p_{level} \leq president\ file_{level}}{\overset{deceiving\ info}{===\Rightarrow}} [presidential\ top\ secret\ file]$$

Obama: "Hm… it says here I should bomb Iraq."

Biba model for integrity

The **Biba model** solves the integrity problem.

Each object (files) and subject (user) has an **integrity level** associated with it which consists of an **integrity label** and some other information.

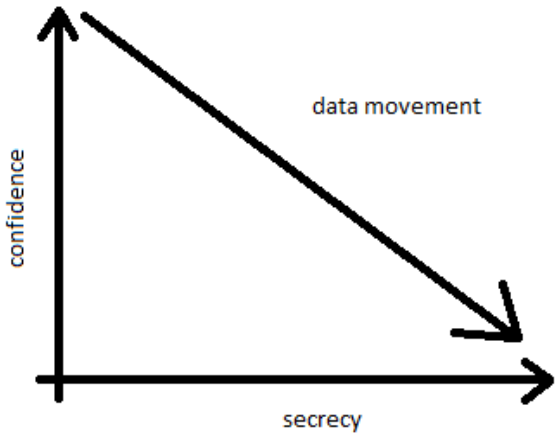
There are three possible integrity labels:

- First-hand information (stuff you have learned first-hand, this is what you trust the most)
- Second-hand information (something you learn from a friend, is second hand)
- Internet information (the internet is the most unreliable source)

The Biba model works by allowing users to **write-down** and **read-up,** exactly opposite to Bell-LaPadula.

- Read up from files of higher integrity levels
- Write down to files of lower integrity levels

Combining the Bell-LaPadula model and the Biba model, you will get a 2-dimnesional model which provides **secrecy** (top, secret, public) and **confidence** or **integrity** (first hand, second hand, internet).



CSE 409, Fall 2011, Rob Johnson, http://www.cs.stonybrook.edu/~rob/teaching/cse409-fa11/
Alin Tomescu, September 16th, 2011

Static versus floating Bell-LaPadula

In **static** mode, files have a label and processes have a label and these are **fixed labels**.

When the process starts up, its label is set based on the user running the process. If the user has top-secret access so will the process. This way, the process will not be able to write down top secret info.

But what if a top-secret user would like to read & modify just a secret document? In **static** mode he would never be able to do that because that would mean writing down. **Floating** mode fixes this issue.

In **floating** mode, the process receives the public label, and then it can read in many files. As it reads in each file, the label of the process gets upgraded to the file's label (floats up), if it is necessary. This way, when the process writes out, the file will be given the label of the process.

Interesting thing: Note that while Bell-LaPadula does an excellent job at restricting information flow in the system, it can still be tricked. In **floating** mode, a user can launch two processes, **a reader** which reads in a top-secret file and **a writer** which will write out to a public file. These two processes can communicate with each other through some covert channel. This way, the reader can send the file (slowly) to the writer.

Role based access control

Role based access control (RBAC) is very useful in large organizations where there are a lot of people doing the same thing.

RBAC works very simply. Users are assigned roles and roles are granted permissions.

$$user \rightarrow role \rightarrow permission$$

Bank example:

- **Roles:** teller, branch manager, security guard
  - **Tellers:** John, Frank, Susan
  - **Branch manager:** Jimmy
  - **Security guards:** Rufus, Ralph
- **Permissions:**
  - Open cash door – teller, branch manager
  - Open safe – branch manager
  - Open doors – security guard

You might also need a location parameter that ensures the manager of the 25A branch can't open the 347 branch's door.

The Chinese wall access control

There may be objects that are mutually exclusive in terms of who can access them.

Examples: Editorial vs. Advertising, Client A vs. Client B, Local files vs. Network