CSE 127: Computer Security

Multilevel Security

Kirill Levchenko

October 14, 2014

Access Control

- **Access Control:** Restriction of access to information or function
  - **Access:** ability to see, use or modify
- Access control policy is part of security policy
  - In many settings access control is same as security
  - Not all security issues are naturally formulated as access
    - E.g.: availability

Access Control

- **Discretionary Access Control:**
  Access controlled by users
  - **New objects:** users (e.g. *owner*) decides who can access
  - Security *depends on users*
- **Mandatory Access Control:**
  Access not controlled by users
  - **New objects:** system determines who can access
  - Most systems conservative in assigning new object access

Examples

- **Discretionary Access Control:**
  Access controlled by users
  - Unix access control
  - SQL access control
- **Mandatory Access Control:**
  Access not controlled by users
  - Information flow tracking systems
  - AppArmor and SELinux

Non-Computer Examples

- Discretionary access control:
  - Physical keys (capabilities!)
  - Vampires (must be invite into home)
- Mandatory access control:
  - Child-proof medicine bottles
  - Infectious disease quarantine

Multilevel Security

- All objects have a security label
  - Labels form a lattice (partial order with supremum and infimum)
- Subjects limited to certain labels
  - Usually defined in terms of maximal accessible labels

Traditional Multilevel Security

- Labels
  - **Level:** *top secret* > *secret* > *confidential* > *unclassified*
  - **Category:** unordered set (e.g. ULTRA)
  - Other restrictions such as NOFORN (not foreign)
- Subjects cleared to certain level and category
- Subject must have all category clearance and same or higher level as object being accessed

Noninterference

- Want to stop information from leaking from higher levels
- How to formalize this? **Noninterference.**
- Information at higher levels can have *no effect* on information at lower levels
  - Prevent side-channels within system
  - Control classification of new information
    - Low-clearance information must not depend on higher-clearance information

Bell-LaPadula

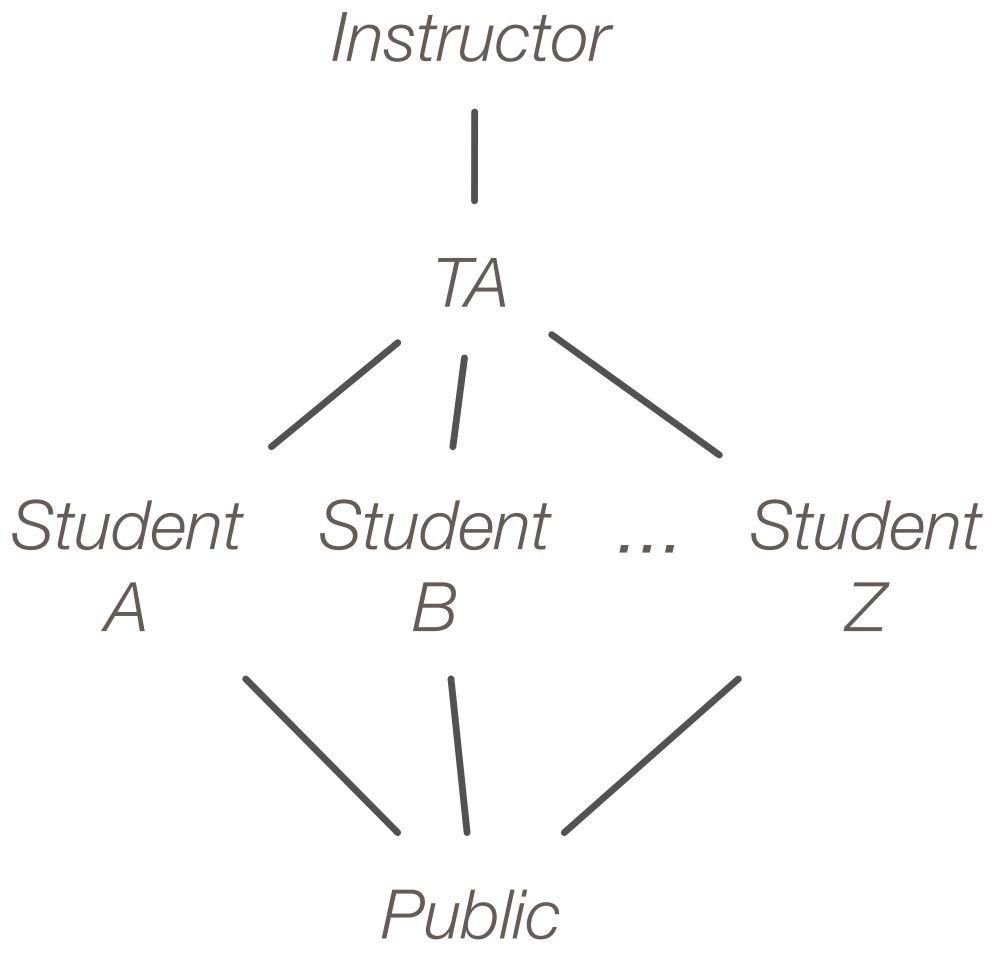
- Model for protecting *secrecy* of information
- Each process (running on behalf of user) **has a clearance**
- **No read up:** can't read data from higher level
- **No write down:** can't write data to lower level

Bell-LaPadula with HWM

- Model for protecting *secrecy* of information
- Each process (running on behalf of user) has a *maximum* clearance and a *current* clearance
- **No read up:** can't read data from higher level
- **No write down:** can't write data to lower level
- **High water mark principle:** Process starts at lowest level and raises current clearance (up to maximum) to read data with label above current clearance

CSE 127 Lattice

- **Instructor:** can be viewed by instructor only
- **TA:** Can be viewed by TAs and instructor only
- **Student *X*:** Can be viewed by Student *X*, TAs, and instructor only



CSE 127 Bell-LaPadula

- Grade of student A classified *Student A*
  - Can be read by student A, TAs, instructor, administrator
- Homework answer key classified *TA*
  - Can be read by TAs and instructor, administrator

CSE 127 Bell-LaPadula

- **No write down implies:**
  - Student can write email for instructor's eyes only
  - I can only write to *Instructor* level, which you can't read!
- Need high water mark principle
- Need better analogy
  - In the above: *process = person*

CSE 127 Bell-LaPadula

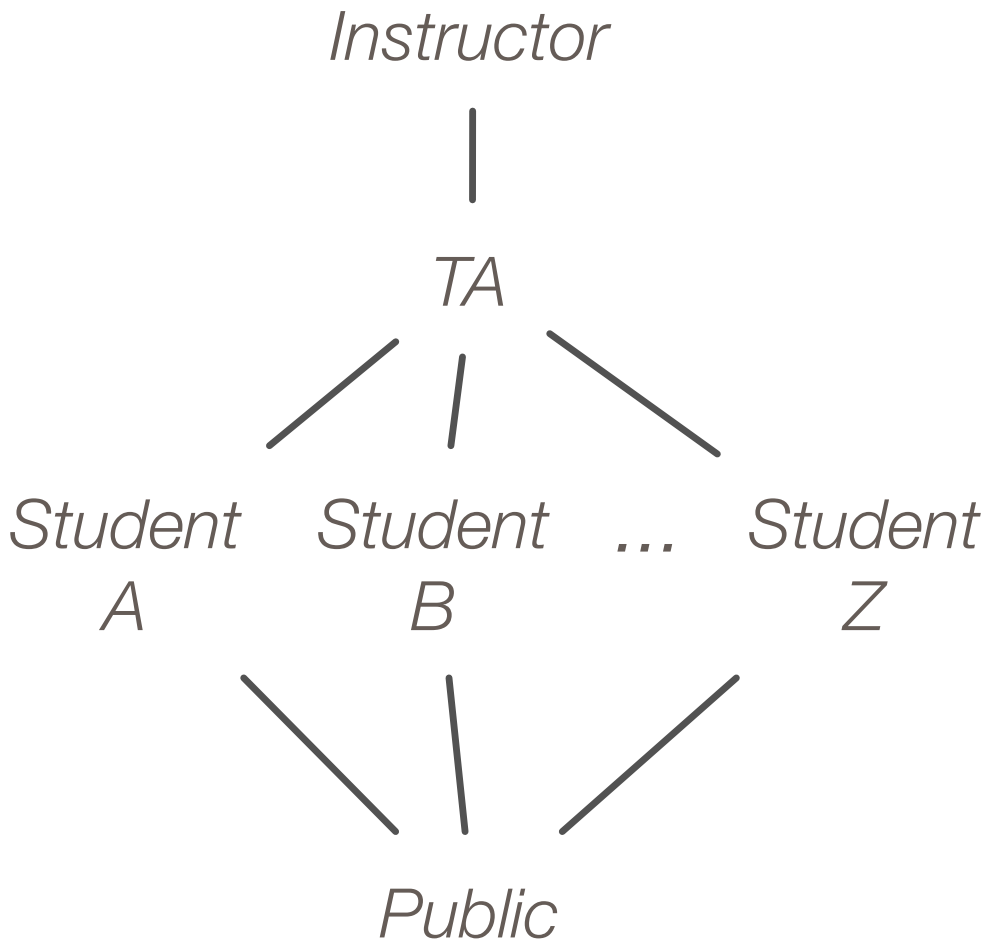
- ***Let's say:* New process when I sit down at a desk**
  - Short-term *Memento*-style amnesia
- I start at *Public* clearance
  - Can write up to any level
  - Can only read *Public* at current level
- I read student *X*'s homework labeled *Student A*
  - Read raises up my current process clearance to *Student A*
  - Can write up to *Student A*, *TA*, *Instructor*, but **not** *Public*

CSE 127 Bell-LaPadula

- I read student *A*'s homework labeled *Student A*
- I am now **tainted** by the knowledge of *Student A* info
- Anything I write may be derived from knowledge of *Student A* level material
- I can't write anything to level *Public* because it might leak *Student A* level information to *Public*

CSE 127 Bell-LaPadula

- Still sitting down, I read homework of student B
- What is my new current process clearance?
  - Must be at least as high as *Student A* and *Student B*
  - Process clearance now *TA*
  - Can only write to level *TA* and *Instructor*



CSE 127 Bell-LaPadula

- What happens when TA grades homework?
  - **Input:** student A's homework at level *Student A*
  - **Input:** Answer key at level *TA*
  - **Output:** Grade at level … ?
- Reading answer key increased TA's current process clearance to level *TA*
- Need way to *declassify* information

CSE 127 Bell-LaPadula

- Special user or process can declassify information
- *Strain the analogy further:* I can declassify information when I am wearing Hammer pants
  - Grade: *TA* to *Student A*

Bell-LaPadula Summary

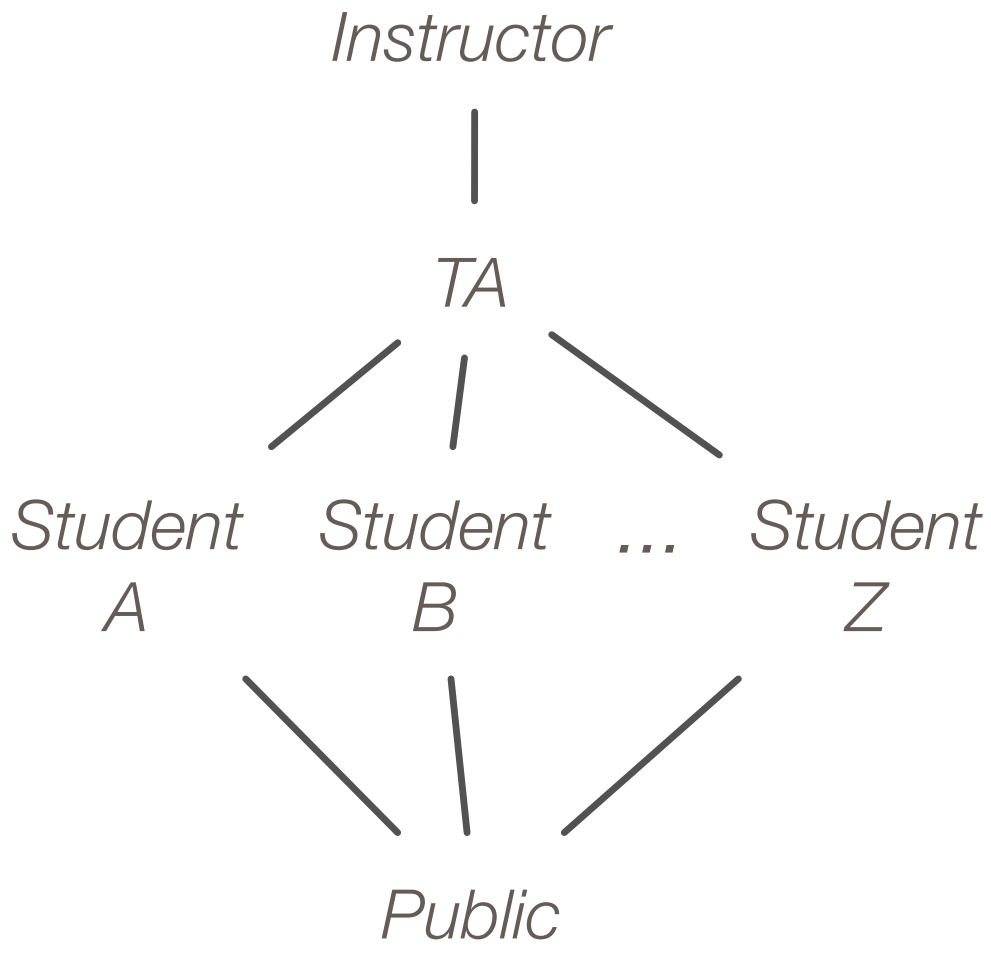
- BLP only protects *secrecy* of information
  - Integrity must be handled some other way
- System must label every piece of information
  - Recall *Complete Mediation* design principle
  - Requires a closed system
- What happens when data leaves system?

Biba Model

- Model for protecting *integrity* of information
- Each process (running on behalf of user) **has a level**
- **No read down:** can't read data from lower level
- **No write up:** can't write data to higher level
- **Low water mark:** start at highest level, decrease as necessary on read operation

CSE 127 Lattice

- Lattice now describes integrity (accuracy) of information
- **Instructor:** highest
- **Public:** lowest



CSE 127 Biba: without LWM

- I will only believe what I wrote myself
- TAs will only believe what they wrote or I wrote
- Each student only believe his/her own documents, as well as documents of TAs and instructor
- So when is low water mark policy useful?

Don't Talk About It

- **Two levels:** *Not Seen Fight Club* > *Seen Fight Club*
- Everyone starts at *Not Seen Fight Club*
- After seeing *Fight Club* level becomes *Seen Fight Club*
- Everything you write or say is labeled with your level
- If you have not seen Fight Club and read something written by someone who has, you're **tainted**
  - I will not talk to you because what you know might spoil movie

BLP vs Biba

- Which is better?
- When should you use BLP?
- When should you use Biba?

Multilevel Security

- BLP and Biba operate on processes and files
  - This is too coarse: processes get tainted easily
- What if we label *every byte of data on the system?*
- "Process" is now a single instruction
  - BLP: result of operation the higher of each operand's labels
  - Biba: result of operation the lower of each operand's labels

Multilevel Security

- **Information Flow Tracking:**
  Fine-grained label tracking
- *Static:* Label variables in a program

Jif: Java + information flow

Download Jif 3.4

Jif is a security-typed programming language that extends Java with support for information flow control and access control, enforced at both compile time and run time. The source code for the Jif compiler and run-time system is now available for download. Jif is written in Java and is built using the Polyglot extensible Java compiler framework.

Static information flow control can protect the confidentiality and integrity of information manipulated by computing systems. The compiler tracks the correspondence between information the policies that restrict its use, enforcing security properties end-to-end within the system. After checking information flow within Jif programs, the Jif compiler translates them to Java programs and uses an ordinary Java compiler to produce secure executable programs.

Jif extends Java by adding labels that express restrictions on how information may be used. For example, the following variable declaration declares not only that the variable `x` is an `int`, but also that the information in `x` is governed by a security policy:

```
int {Alice→Bob} x;
```

In this case, the security policy says that the information in `x` is controlled by the principal Alice, and that Alice permits this information to be seen by the principal Bob. The policy `{Alice←Bob}` means that information is owned by Alice, and that Alice permits it to be affected by Bob. Based on label annotations like these, the Jif compiler analyzes information flows within programs, to determine whether they enforce the confidentiality and integrity of information.

Documentation

Jif reference manual
Compiler API
Change log

Current developers

Danfeng Zhang
Owen Arden
Jed Liu
K. Vikram
Stephen Chong
Andrew Myers

Past contributors

Nate Nystrom

Multilevel Security

- **Information Flow Tracking** (IFT)**:**
  Fine-grained label tracking
- *Static:* Label variables in a program
- *Static:* Recompile binary executables to use labels
- *Dynamic:* Track byte labels at instruction label

MLS vs IFT

Multilevel Security	**Information Flow Tracking**
Process	Operation (add, sub, xor, *etc.*)
Label	Taint tag
File or input device	File or input device
High/low water marking	Bitwise OR of taint tag

TaintDroid

- Track data labels at variable granularity by modifying Android's Dalvik virtual machine
- Labels are 32 bits
- The label of result of every operation is bitwise-OR of labels of operands
- What do the labels represent in TaintDroid?

TaintDroid

- Track data labels at variable granularity by modifying Android's Dalvik virtual machine
- Labels are 32 bits
- The label of result of every operation is bitwise-OR of labels of operands
- What do the labels represent in TaintDroid?
  - Source of sensitive data (e.g. GPS or microphone)
  - Data the user may not want to leave the device

TaintDroid

- Is TaintDroid trying to protect *secrecy* or *integrity?*
  - *Secrecy*, although mechanism can be used for both
- TaintDroid records when an app sends tainted data out
  - Can be used to prevent such data from leaving phone

IFT for Integrity

- How can you use IFT to protect integrity?
  - *High:* trusted data, *Low:* untrusted data
  - Do not execute *Low* code or use *Low* address in jump
- Protect files, directories, processes from *Low* data
- What are the limits?

IFT Achille's Heel

What do you do when tainted data used as condition of jump?

IFT Achille's Heel


```
if (b) {
        a = 1;
        x = y;
}
```

IFT Achille's Heel

- Option 1: Ignore control dependencies
  - Only taint arithmetic and memory access operations
- Option 2: Taint all variables *inside* then and *else* clauses
- Neither one is ideal:
  - Option 1 leads to incomplete taint
  - Option 2 leads to over-tainting

Summary

- Multilayer security guarantees information protection
- Implementation must be correct
- Declassification must be correct
- Side-channels may still exist

For Next Class

- **Generate one random bit**
  - *Remember what it is!*
- Serial number of the first banknote you find modulo 2
- Modulo 2 sum of the last 4 digits of your credit card number or student id:
  - “5432” → 5 + 4 + 3 + 2 = 14 → 0
- Use `random.org`