Multi-Level Security & Information Flow Control

Software Security

Steffen Helke

Chair of Software Engineering

21st November 2018

Brandenburgische Technische Universität Cottbus - Senftenberg

Objectives of today's lecture

- ➜ Repetition: What are the main rules of the *Bell-LaPadula model*?
- ➜ Getting to know other security models and design principles of a *Trusted Computing Base*
- ➜ Understanding and applying analytical techniques to detect undesirable information flows in a program code
- ➜ Being able to implement security policies for small code examples based on JiF (Java + Information Flow)

Repetition: Bell-LaPadula Model (BLP)

Repetition: Bell-LaPadula Model (BLP)

1. Which protection goal is implemented by BLP?

➜ Confidentiality

2. What are the most important rules of BLP?

➜ No Read Up: Subjects are not allowed to read an object of a higher security class

➜ No Write Down: Subjects are not allowed to write an object of a lower security class

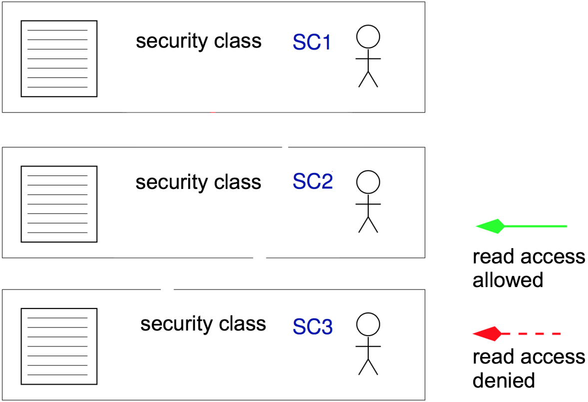
3. How are security classes represented and how are they ordered?

➜ Represented as pairs (A, C), where A is a *sensitivity level* and C is a *set of compartments*

➜ Information from (A, C) to (A', C') shall flow iff $A \leq A'$ and $C \subseteq C'$

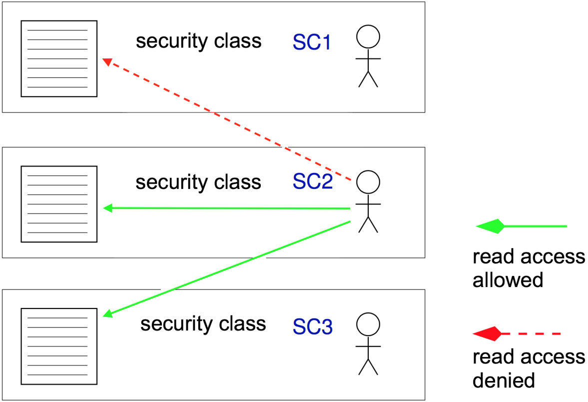
No Read Up (Simple Security Property)

... assumed if $SC3 \leq SC2 < SC1$, then what is allowed?



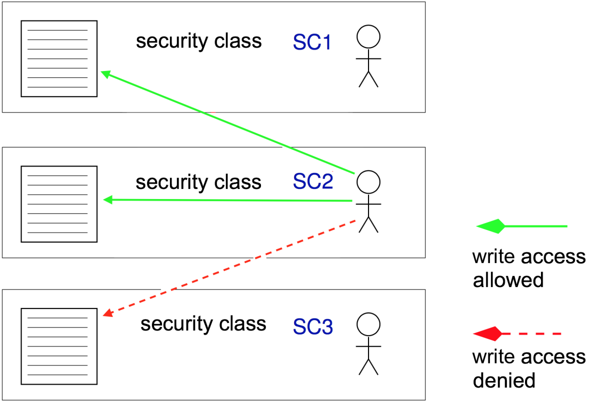
No Read Up (Simple Security Property)

... assumed if SC3 $\leq$ SC2 $<$ SC1, then what is allowed?

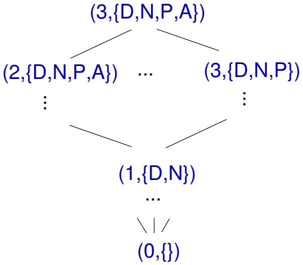


No Write Down (*-Property)

... assumed if $SC3 < SC2 \leq SC1$, then what is allowed?



Example: Hospital Scenario



Note:

- all compartments of a security class must be linked by conjunction
- e.g. $(\_, \{D, N\})$ means, reading is only allowed if the subject is authorized for both compartments doctor <u>and</u> nurse

1. Will it be possible for a person or process P with $SC = (1, \{D\})$ to read a document with $SC = (0, \{D\})$? yes
2. Is it possible for P to expand/write a document with $SC = (2, \{D, N\})$? yes
3. Are there documents that P is allowed to read and write at the same time? yes, documents with $SC = (1, \{D\})$!

Generalization by Dorothy Denning

Complete Lattice $((A, C);\ \leq\ ;\ \oplus;\ \otimes)$

with $(A, C) \oplus (A', C') = (\max\ (A, A'), C \cup C')$

$(A, C) \otimes (A', C') = (\min\ (A, A'), C \cap C')$

$Low = (0, \varnothing)$

$High = (n, c)$

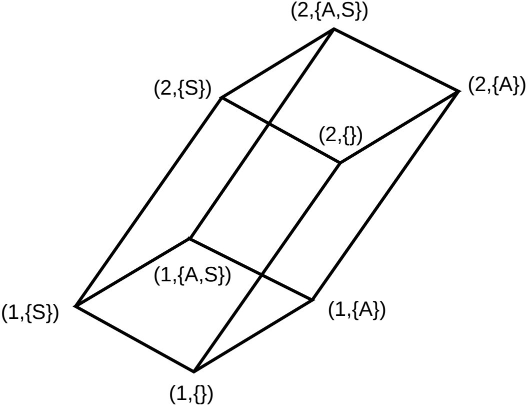
where n: highest security level

c: universal (largest) set of compartments

Note: Completeness follows from finiteness of A and C!

Visualisation by Hasse Diagram

- Often used to represent a finite partially ordered set
- Each edge links smaller and larger elements
- The lower end of an edge represents the smaller element



Benefits of the Generalization

Idea

- Information flows of a program are generated by program statements that combine data or processes from different security classes
- Denning's formalization makes it possible to describe information flows with the help of the lattice operators

Example $Y := X_1 + X_2 * X_3$

induces the information flows

$X_1 \rightarrow Y$, $X_2 \rightarrow Y$ und $X_3 \rightarrow Y$

Using the operator $\oplus$, the test can be reduced to the following check

$$SC(X_1) \oplus SC(X_2) \oplus SC(X_3) \leq SC(Y)$$

Limitations of the Bell-LaPadula Model

Advantages

- Comprehensible theory and an excellent mathematical basis

Drawbacks

- Rules in practice often not sufficient and not flexible enough
- Consequence: User prompts Admin to temporarily classify documents in a lower security class

Solution

- Use of Tranquility Properties
  ➜ Security classes of objects or subject are allowed to change under certain rules
- Assumption is that the classification of an object does not change as long as it is still referenced

Use of Tranquility Properties

What is the difference between a strong and a week tranquility property?

Strong

- Subjects and objects do not change security classes during the lifetime of the system

How to make BLP more flexible? What do you know about the high watermark principle?

Weak

If user A is writing a CONFIDENTIAL document, and checks the unclassified dictionary, the dictionary becomes CONFIDENTIAL

- Security classes are only modified in conformance with the specified security policy model
  ⇒ High Watermark Principle

(compliance with standards, rules, or laws.)

- e.g. process starts with low security class and is upgraded when accessing objects of higher security classes
- An upgrade is only possible if the security model is not violated

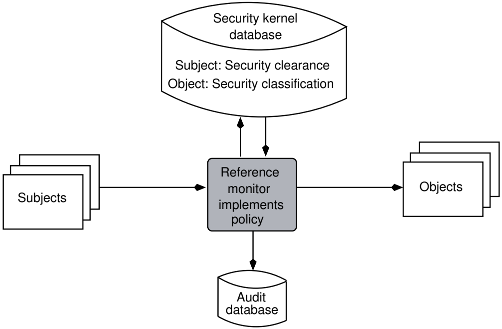
Under high-water mark, any object less than the user's security level can be opened, but the object is relabeled to reflect the highest security level currently open, hence the name.

Limits of BLP & High Watermark Principle

- Standard software often doesn't run without errors, because temporary files created before an upgrade can't be written afterwards
- WriteUp must be *blind*, because confirming the successful writing would open an information channel about the state of the higher class
- How to create excerpts from documents for publication (downgrading)?
- How do you classify certain documents that are only to be protected in combination?
- Conclusion: The assumption of the Tranquility Property is often too strong in practice!

Trusted Computing Base (TCB)

- TCB covers all hardware and software components for implementing the security concepts
- Implementable as a *reference monitor* or as a part of an operating system core



Properties of a Trusted Computing Base

What are the design principles of a Trusted Computing Base (TCB)?

Design Principles

- TCB should be as *small as possible* and consist of just a few components
- Reference monitor and databases have to be protected from unauthorized access
- Verify the correctness of all TCB components if possible
- Every access to the system must be controlled by the TCB!

Covered Channels

If a Trojan horse enters a high security class level and transmits information to lower security class level by bypassing the protection mechanism, a hidden channel is present

How to bypass BLP with the help of covered channels?

Such information flows can be implemented via ...

➜ Storage Channels:

- Process of high security class transmits confidential information by actions with the hard disk drive
- e.g. modify the position of the hard disk's read head or open, close or lock files
- Process of low security class is able to monitor these actions

➜ Timing Channels:

- Information transmission by measuring the runtime of processes

Other Security Policy Models

What are the differences between the BIBA and BLP model?

Bell-LaPadula-Model (BLP, 1969/73)

- ➜ Protection goal: Confidentiality of data
- ➜ Rules: *No-Read-Up* and *No-Write-Down*

Another analogy to consider is that of the military chain of command. A General may write orders to a Colonel, who can issue these orders to a Major. In this fashion, the General's original orders are kept intact and the mission of the military is protected (thus, "read up" integrity). Conversely, a Private can never issue orders to his Sergeant, who may never issue orders to a Lieutenant, also protecting the integrity of the mission ("write down").

Biba-Model (1977)

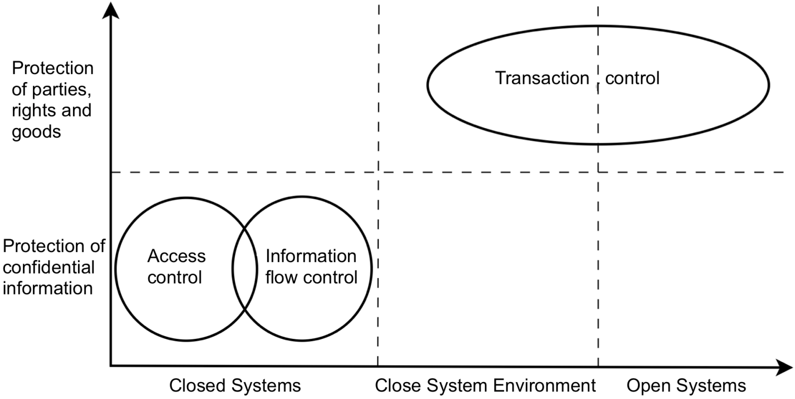
- ➜ Protection goal: Integrity of data
- ➜ Biba is dual of BLP, i.e.
  Rules: *No-Read-Down* and *No-Write-Up*

Low-Watermark Mandatory Access Control (LoMAC)

- ➜ Variant of the Biba model, allows read access for objects with high integrity to objects of low integrity
- ➜ Reading subject is then downgraded to a lower integrity level

The Simple Integrity Property states that a subject at a given level of integrity must not read data at a lower integrity level (read up). The * (star) Integrity Property states that a subject at a given level of integrity must not write to data at a higher level of integrity (write down).

Relationship between Protection Goals and Operational Environment



Source: Vitali Keppel: Klassifikation und Analyse von IT-Sicherheitsmodellen, Masterarbeit, Uni Koblenz-Landau, Juni 2013.

Implementation of MLS Security Models

Hersteller/Implementation	Typus	System	Akkreditiert
NSA/Red Hat - SELinux	Variante von Bell-LaPadula	Red Hat, Gentoo Linux, SUSE, Debian, Darwin	-
TrustedBSD	Biba, LoMAC	TrustedBSD, Mac OS X, Darwin	-
Novell AppArmor	-	Ubuntu, SUSE	-
Rule Set Based Access Control (RSBAC)	Variante von Bell-LaPadula	Gentoo Linux, Debian, Fedora	-
Sun Microsystems	Variante von Bell-LaPadula	Sun Trusted Solaris	-
Microsoft	Biba	Windows Vista	-
Unisys	Biba, Bell-LaPadula, Lattice (compartment), Clark-Wilson	OS2200	TCSEC B1
Argus Systems Group PitBull LX	Lattice (compartment)	AIX, Sun Solaris, Linux	ITSEC F-B1, E3

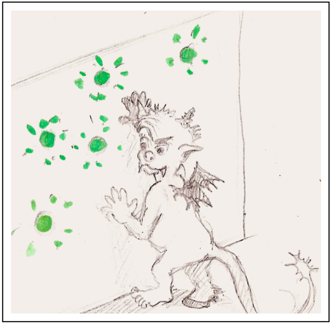
Source: `http://de.wikipedia.org/wiki/Mandatory_Access_Control`

JIF: Java + Information Flow

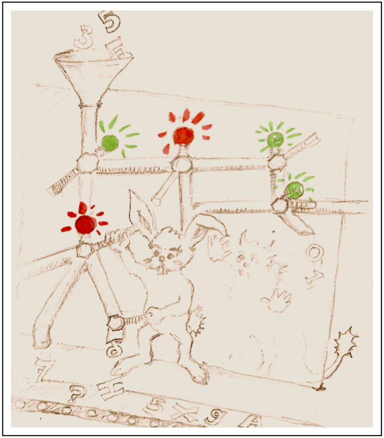
Information Flow Security

➜ Attacker should not learn confidential (red) information by observing visible (green) events/outputs

Attacker interface



Developer interface



Motivation for Information Flow Control

Problem

- Security engineering methods support the systematic derivation of security policies on an abstract level, e.g. information may only flow up ($L \rightarrow H$)
- Mapping security rules into the program code is challenging
- Specifying access modifiers (e.g. private, public etc.) is generally not powerful enough

Solution

- Labelling of the program code with security labels to express security policies more precisely
- Performing security analysis based on labels, e.g. by compilers

JIF: Java + Information Flow

General Remarks

- Development 1997 at the Cornell University
- Security-typed programming language that extends Java
- Based on a *Decentralized Label Model* (DLM)

Language Features

- Labels for *Integrity* & *Confidentiality*
- Policy specification using *Principals*
- Hierarchical relationships between principals can be specified (*acts-for relation*)
- Declassification of labeled objects with respect to confidentiality & integrity is supported

JIF-Homepage: `http://www.cs.cornell.edu/jif/`

Decentralized Model of Security Labels

How to specify a security policy for confidentiality or integrity using JiF?

Confidentiality Rules

- Notation: $u \to p$
- Owner u trusts the reader p not to give the information to unauthorized persons

Integrity Rules

- Notation: $u \leftarrow p$
- Owner u trusts the writer p not to destroy or damage the information

```
// only Alice and Bob are allowed to read a
int{Alice -> Bob} a;

// only Alice is allowed to write b
int{Alice <- Alice} b;

// combined read and write permissions
int{Alice -> Bob; Alice <- Alice} c;
```

Example: Assignment Operator of JIF

Example for Confidentiality

```
// only Alice and Bob are allowed to read a
int{Alice -> Bob} a;

// only Alice is allowed to read b
int{Alice -> Alice} b;

a = b;
b = a;
```

Example: Assignment Operator of JIF

Example for Confidentiality

```
// only Alice and Bob are allowed to read a
int{Alice -> Bob} a;

// only Alice is allowed to read b
int{Alice -> Alice} b;

a = b; // not allowed!
b = a; // allowed, because readers(b) ⊆ readers(a)
```

Example: Assignment Operator of JIF

Example for Integrity

```
// only Alice and Bob are allowed to write a
int{Alice <- Bob} a;

// only Alice is allowed to write b
int{Alice <- Alice} b;

a = b;
b = a;
```

Example: Assignment Operator of JIF

Example for Integrity

```
// only Alice and Bob are allowed to write a
int{Alice <- Bob} a;

// only Alice is allowed to write b
int{Alice <- Alice} b;

a = b; // allowed, because writers(b) ⊆ writers(a)
b = a; // not allowed!
```

How principals are created?

Concept

- *Principals* are entities with some power to observe and change certain aspects of the system
- Useful to model *users*, *processes*, *user groups*, or other application-specific entities, like *nodes of a netwerk*

Example

- There are several ways to create principals, e.g.

```
import jif.util.*;
class Test {
 public static void main (String args[])   {
  String sAlice = "Alice";
  final principal Alice = new ExternalPrincipal(sAlice);...
}}
```

Note: When defining labels, the bottom and top can be represented by * (*no principals*) and _ (*all principals*). These characters represent only principal sets and not objects of the class Principal

How principals are created?

- ➜ Principals can also be created by independent classes with static attributes *using a singleton pattern*
- ➜ Advantage is that e.g. the principal *Steffen* is created only once and can be used by other classes of the package

```
package jif.principals;

public class Steffen extends ExternalPrincipal {
    public Steffen() {
        super("Steffen");
    }

    private static Steffen{*!:*} P;
    public static Principal getInstance{*!:*}() {
        if (P == null) {
            P = new Steffen();
        }
        return P;
    }}
```

Example: Defaultlabel of JIF

Security label without explicit information

➜ .. means that everyone is allowed to read or modify the variable

```
// everything is allowed
int{} a; // abbreviation for int {_->_ ; _<-_} a;

// only Alice is allowed to read b
int{Alice -> Alice} b;

a = b; // not allowed!
b = a; // allowed, because readers(b) ⊆ readers(a)
```

Under which condition can an arbitrary value be assigned?

➜ ... if the variable is *unreadable for anyone* and *writable for everyone*

```
int{* -> *; _<-_} a;

int{Alice -> Alice; Alice <- Bob} b;

a = b;
```

Generalization: Assignments

What does the JIF compiler check for the assignment

```
a = b;
```

?

What does the JIF compiler check for an assignment?

➜ Main target: $sc(b)$ is less restricted than $sc(a)$

i.e. $sc(b) \sqsubseteq sc(a) := sc(b) \sqsubseteq_{\mathbf{C}} sc(a) \wedge sc(b) \sqsubseteq_{\mathbf{I}} sc(a)$

➜ Consists of the following subgoals:

1. b has a greater or equal number of readers than a

   $sc(b) \sqsubseteq_{\mathbf{C}} sc(a) := readers(b) \supseteq readers(a)$,

2. b has a lower or equal number of writers than a

   $sc(b) \sqsubseteq_{\mathbf{I}} sc(a) := writers(b) \subseteq writers(a)$

➜ This results in the following partial order relation with a bottom and top element $\{\_ -> \_; * <- *\} \sqsubseteq \quad \ldots \quad \sqsubseteq \{* -> *; \_ <- \_\}$

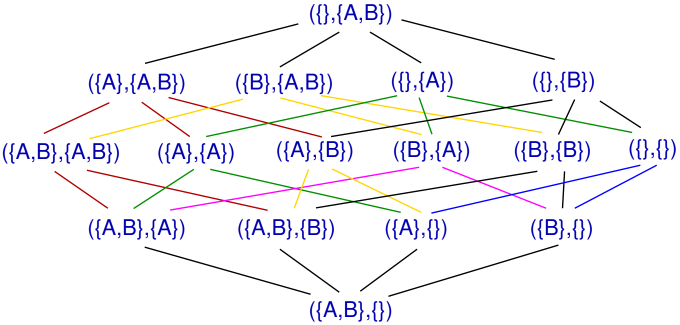
Note: The JIF-operator $*$ represents *no* principals and the JIF-operator $\_$ represents *all* principals, $sc(x)$ is the security label of the variable x

Complete Lattice for JiF-Labels

Assumptions

- ➜ The number of principals is limited to two with $P = \{A, B\}$
- ➜ Security labels are described by tuple (R, W), with $R =$ authorized readers and $W =$ authorized writers

Solution



How to deal with implicit information flows?

How are implicit information flows handled in JiF?

- Analysis of implicit information flows is implemented using a *pc label* (*program-counter label*)
- *pc labels* are initial empty

```
class Test {
    int {Bob -> Alice,Bob} a = 0;
    int {Bob -> Bob} b;

    public void f {} () {
      if (a == 0) {
          b = 4;  // check the pc label
      }
    }
}
```

➜ For each assignment the *pc label* has to be checked

➜ Proof obligation of the example for b = 4;

{Bob −> Alice,Bob} $\sqsubseteq$ {Bob −> Bob}

Implicit information flows and method calls

What is problematic with a method call? How the JiF-Compiler is able to solve this problem?

Question: What is problematic with a method call?

```
class Test {
   int {Bob -> Alice,Bob,Steffen} a = 0;
   int {Bob -> Bob} b;

   public void f {} () {
     if (a == 0)
         setB();
   }
   private void setB () {
     b = 4;
}}
```

➜ Value assignment in the method setB() must be checked for all possible contexts in which this method is potentially called

Implicit information flows and method calls

What is the meaning of begin and end-labels in JiF? How is it possible to support JiF-refactorings?

- Implicit information flows for method calls are handled in JIF by so-called *begin labels*
- A *begin label* defines the *upper bound* for all pc labels at which the method can be called

```
class Test {
   int {Bob -> Alice,Bob,Steffen} a = 0;
   int {Bob -> Bob} b;

   public void f {} () {
     if (a == 0)
         setB();
   }
   private void setB {Bob -> Bob,Steffen} () {
     b = 4;
}}
```

➜ Check at the method call of setB(): *pc-label* $\sqsubseteq$ *begin-label*
{Bob -> Alice , Bob , Steffen} $\sqsubseteq$ {Bob -> Bob , Steffen}