

Bell-LaPadula and Other Security Models

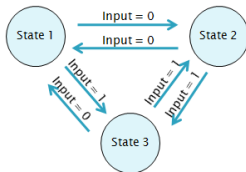
- ▶ Bell-LaPadula model
- ▶ Biba model
- ▶ Chinese Wall model
- ▶ (Clark-Wilson model)

Motivation

- ▶ Demonstrate how security policies can be expressed in a *formal* way.
- ▶ Give some history of computer security
- ▶ Understand the limitations of various models

State Machine Model

- ▶ **State** – Representation of the system at some given time
- ▶ **State transition** – next state depends on current state + input.
- ▶ **Idea:** If we start in a secure state and all state transitions preserve security, then the system will be secure.



Question:
What is a secure state?

Bell-LaPadula Model

- ▶ Most famous security model
- ▶ First developed around 1973
- ▶ "Unified exposition and Multics interpretation", 1976
- ▶ Focus on **confidentiality**, not integrity
- ▶ Based on state transitions
- ▶ Both **mandatory** and **discretionary** access control
 - Multilevel security
 - Access control matrix

Notation

- ▶ Set of subjects S
- ▶ Set of objects O
- ▶ Set of access operations A
 - execute, read, append, write

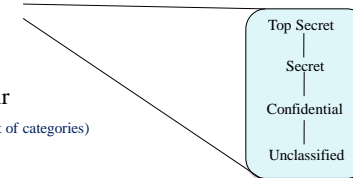
	Execute	Append	Read	Write
Observe			X	X
Alter		X		X

- ▶ Set of security levels L with ordering \leq
- ▶ Functions
 - $f_S: S \rightarrow L$, maximum security level
 - $f_C: S \rightarrow L$, current security level
 - $f_O: O \rightarrow L$, security level of object

Security Levels

- ▶ *Multilevel security*
- ▶ *Categories*
 - {Division A, Division B}
- ▶ *Security level given by pair*

(Classification, Set of categories)



- ▶ *Ordering:*

$$(h_1, c_1) \leq (h_2, c_2) \text{ if and only if } h_1 \leq h_2 \text{ and } c_1 \subseteq c_2$$

- ▶ Security level (h_2, c_2) dominates (h_1, c_1)

The State

The state consists of three parts

1. Current access given by a set of (s, o, a) tuples
 - An element of the powerset $P(S \times O \times A)$
 - Can be written as matrix b .
 - s is row, o is column, a is current access operation
 2. Access matrix given by M
 - Defines what is allowed
 3. Functions $f = (f_S, f_C, f_O)$
- ▶ State is given by (b, M, f)

State Example

- ▶ We have a system with 5 subjects and 5 objects, 2 classifications and 2 categories
 - **Subjects:** Alice, Bob, Charlie, David, Erika
 - **Objects:** file_a, file_b, file_c, file_d, file_e
 - **Classifications:** public, private
 - **Categories:** A, B

State Example

Current access set b
 (Alice, file_b, r)
 (David, file_c, w)
 (Erika, file_a, a)

Access Control Matrix M

	file_a	file_b	file_c	file_d	file_e
Alice		r,w,a		e	a
Bob			a	r,e	
Charlie	r	r			a
David			r,w,a		r,w,a
Erika	a			e	

Functions $f = (f_s, f_c, f_o)$

f_s :
 Alice: (private, {A}), Bob: (public, {A,B}), Charlie: (public, {B}), David: (private, {A,B}), Erika: (public, {A})
 f_c :
 Alice: (private, {A}), Bob: (public, {A,B}), Charlie: (public, {B}), David: (public, {A,B}), Erika: (public, {A})
 f_o :
 file_a: (private, {A}), file_b: (private, { \emptyset }), file_c: (public, {A,B}), file_d: (public, {A}), file_e: (private, {A,B})

State is given by (b,M,f)

ss-property

- Simple Security Property
- Mandatory access control

	Execute	Append	Read	Write
Observe			x	x
Alter		x		x

State (b,M,f) satisfies the *ss-property* if for each element $(s,o,a) \in b$ where the access operation a is *read* or *write*, the security level of s dominates security level of o , i.e., $f_o(o) \leq f_s(s)$

- No read-up – A user is not allowed read (observe) access to objects with higher security level

Example

Current access set b
 (Alice, file_b, r)
 (David, file_c, w)
 (Erika, file_a, a)

Access Control Matrix M

	file_a	file_b	file_c	file_d	file_e
Alice		r,w,a		e	a
Bob			a	r,e	
Charlie	r	r			a
David			r,w,a		r,w,a
Erika	a			e	

Functions $f = (f_s, f_c, f_o)$

f_s :
 Alice: (private, {A}), Bob: (public, {A,B}), Charlie: (public, {B}), David: (private, {A,B}), Erika: (public, {A})
 f_c :
 Alice: (private, {A}), Bob: (public, {A,B}), Charlie: (public, {B}), David: (public, {A,B}), Erika: (public, {A})
 f_o :
 file_a: (private, {A}), file_b: (private, { \emptyset }), file_c: (public, {A,B}), file_d: (public, {A}), file_e: (private, {A,B})

According to *ss-property*:

- Alice is allowed to read file_b since $f_o(\text{file}_b) \leq f_s(\text{Alice})$
- David is allowed write access to file_c since $f_o(\text{file}_c) \leq f_s(\text{David})$

Controlling Write Access

- A subject can append to any object with higher security level than the subject.
- We do not allow information to flow downwards.
 - Easy way: A subject can not send any information to an object with lower security level – Not practical
 - Better solution:
 - Possible to temporarily downgrade a subject – the reason to introduce $f_c(s)$
 - Let *trusted subjects* send information downwards
- Note that we assume that a subject does not have an internal memory – We have to see it as a process, not a human being
 - Only know the contents of the objects it is currently accessing

*-property (Incomplete Version)

- ▶ Star-Property
- ▶ **Mandatory access control**
- ▶ Does not apply to *trusted subjects*

	Execute	Append	Read	Write
Observe			X	X
Alter		X		X

State (b, M, f) satisfies the *-property if for each element $(s, o, a) \in b$ where the access operation a is *append* or *write*, the current security level of s is dominated by the security level of o , i.e., $f_C(s) \leq f_O(o)$

- ▶ No write-down – A user is not allowed write (alter) access to object with lower security level than the current security level of subject

Example

Current access set b
 (Alice, file_b, r)
 (David, file_c, w)
 (Erika, file_a, a)

Access Control Matrix M

	file_a	file_b	file_c	file_d	file_e
Alice		r,w,a		e	a
Bob			a	r,e	
Charlie	r	r			a
David			r,w,a		r,w,a
Erika	a			e	

Functions $f = (f_S, f_C, f_O)$
 f_S : Alice: (private, {A}), Bob: (public, {A,B}), Charlie: (public, {B}), David: (private, {A,B}), Erika: (public, {A})
 f_C : Alice: (private, {A}), Bob: (public, {A,B}), Charlie: (public, {B}), David: (public, {A,B}), Erika: (public, {A})
 f_O : file_a: (private, {A}), file_b: (private, {∅}), file_c: (public, {A,B}), file_d: (public, {A}), file_e: (private, {A,B})

According to *-property:

- ▶ David is allowed write access to file_c since $f_C(\text{David}) \leq f_O(\text{file}_c)$
- ▶ Erika is allowed append access to file_a since $f_C(\text{Erika}) \leq f_O(\text{file}_a)$

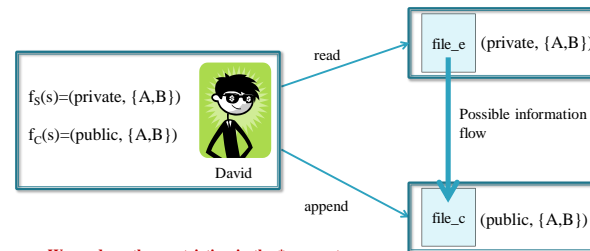
Problem

- ▶ ss-property considers maximum level of subject – $f_S(s)$
- ▶ *-property considers current level of subject $f_C(s)$
- ▶ David has append access to file_c, but he would also be granted read access to file_e

Functions $f = (f_S, f_C, f_O)$
 f_S : Alice: (private, {A}), Bob: (public, {A,B}), Charlie: (public, {B}), David: (private, {A,B}), Erika: (public, {A})
 f_C : Alice: (private, {A}), Bob: (public, {A,B}), Charlie: (public, {B}), David: (public, {A,B}), Erika: (public, {A})
 f_O : file_a: (private, {A}), file_b: (private, {∅}), file_c: (public, {A,B}), file_d: (public, {A}), file_e: (private, {A,B})

Problem

- ▶ **Result:** It would be possible for information to flow from file_e to file_c



→ We need another restriction in the *-property

*-property

State (b, M, f) satisfies the **-property* if for each element $(s, o, a) \in b$ where the access operation a is *append* or *write*, the current security level of s is dominated by the security level of o , i.e., $f_C(s) \leq f_C(o)$

Furthermore, if there exists an element $(s, o, a) \in b$ where the access operation a is *append* or *write*, then we must have $f_C(o') \leq f_C(o)$ for all objects o' with $(s, o', a') \in b$ and a' is *read* or *write*

- ▶ In other words: If a subject has access to several objects at the same time, all objects with observe access must have lower (or equal) security level than all objects with alter access
- ▶ Still does not apply to *trusted subjects*

ds-property

- ▶ Subject may pass an access permission on to other users.
- ▶ Discretionary access control

State (b, M, f) satisfies the *ds-property* if for each element $(s, o, a) \in b$ we have $a \in M_{SO}$

- ▶ Access rights given in access control matrix must also be followed

Example

Current access set b
 (Alice, file_b, $\text{\textcircled{r}}$)
 (David, file_c, $\text{\textcircled{w}}$)
 (Erika, file_a, $\text{\textcircled{a}}$)

Access Control Matrix M

	file_a	file_b	file_c	file_d	file_e
Alice		$\text{\textcircled{r,w,a}}$		e	a
Bob			a	r,e	
Charlie	r	r			a
David			$\text{\textcircled{r,w,a}}$		r,w,a
Erika	$\text{\textcircled{a}}$			e	

Functions $f = (f_S, f_C, f_O)$

f_S :
 Alice: (private, {A}), Bob: (public, {A,B}), Charlie: (public, {B}), David: (private, {A,B}), Erika: (public, {A})
 f_C :
 Alice: (private, {A}), Bob: (public, {A,B}), Charlie: (public, {B}), David: (public, {A,B}), Erika: (public, {A})
 f_O :
 file_a: (private, {A}), file_b: (private, { \emptyset }), file_c: (public, {A,B}), file_d: (public, {A}), file_e: (private, {A,B})

- ▶ All accesses given in b are allowed in the access control matrix M

Basic Security Theorem

- ▶ Current state is *secure* if and only if each $(s_i, o_i, a_i) \in b$ satisfies the three properties
 - ss-property, *-property and ds-property
- ▶ State of the system changes if any component in (b, M, f) changes
- ▶ As long as any state change does not violate any of the three properties, the system remains *secure*

If all state transitions are secure and if the initial state is secure then every subsequent state will be secure

State Transitions

- ▶ **Get access** – add triple (subject, object, access operation) to current access set b } Change b in state
- ▶ **Release access** – remove triple from b } Change b in state
- ▶ **Change object level** – change value of $f_O(o)$ for object o } Change f in state
- ▶ **Change current level** – change value of $f_C(s)$ for subject s } Change f in state
- ▶ **Give access permission** – add an access operation to M } Change M in state
- ▶ **Rescind access permission** – remove an access operation from M } Change M in state
- ▶ **Create object** – add an object to system } Not supported by "our" state
- ▶ **Remove object** – remove an object from system } Not supported by "our" state

Tranquility

McLean criticism:

Make a state transition that

- ▶ downgrade all subjects and objects to lowest security level
- ▶ enter all access rights in all entries of M
- Everyone can do everything – not secure

Bell standpoint:

- ▶ If such a transition is required, it should be ok.
- ▶ Otherwise, it should not be implemented.

Tranquility: security levels and access rights never change.

Drawbacks

- ▶ Only focus on confidentiality, not integrity
- ▶ Not addressing management of access control
- ▶ Contains *covert channels* – information flow not controlled by the security mechanisms.

Example:

- Low level subject creates file.txt at low level
- High level subject upgrades file.txt to higher level, or leaves it alone
- Low level subject tries to read file.txt

Example 2:

- If low-level subjects can read filenames at high levels the filename can also be used to send information from high-level subjects

Biba Model

- ▶ Focuses on integrity, i.e., unauthorized modification of data
- ▶ Proposed in 1977
- ▶ Similar to Bell-LaPadula in several ways
 - Based on multilevel security with a partial ordering
 - Based on subjects and objects
- ▶ Subjects and objects mapped to *integrity levels* forming a lattice
 - $f_S: S \rightarrow L$ subject integrity level
 - $f_O: O \rightarrow L$ object integrity level

Subjects and Objects

- ▶ Information flows **downward**
- ▶ High integrity subjects and objects are called **clean**, low integrity subjects and objects are called **dirty**
 - Clean objects are more accurate or reliable than dirty
 - We have more confidence in clean subjects to execute as expected or to validate input
- ▶ Clean objects cannot be contaminated by information from low-integrity processes (subjects)
- ▶ Clean subjects should not *read* dirty objects
- ▶ **Operations of interest:** Modify, Read, Invoke

Static Integrity Levels

- ▶ Simple integrity property
 - Corresponds to ss-property in Bell-LaPadula
 - If subject s can modify object o , then $f_o(o) \leq f_s(s)$.
 - no write-up
- ▶ Integrity *-property
 - Corresponds to *-property in Bell-LaPadula
 - A subject s can read an object o only if $f_s(s) \leq f_o(o)$
 - No read down

Dynamic Integrity Levels

- ▶ Integrity levels are automatically adjusted
- ▶ Subject low watermark property

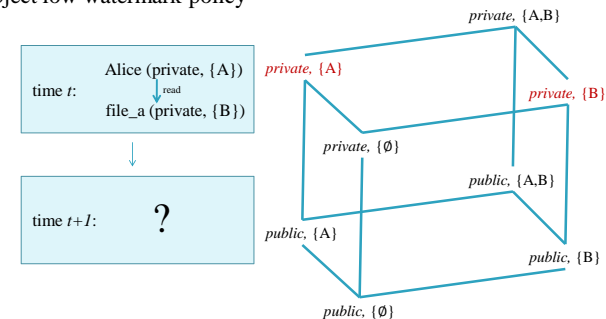
Subject s can read an object o at any integrity level. The new integrity level of the **subject** is the greatest lower bound of $f_s(s)$ and $f_o(o)$.

- ▶ Object low watermark property

Subject s can modify an object o at any integrity level. The new integrity level of the **object** is the greatest lower bound of $f_s(s)$ and $f_o(o)$.

Example

- ▶ Subject low watermark policy



Example

- Subject low watermark policy

time t : Alice (private, {A})
↓ read
file_a (private, {B})

time $t+I$: Alice (private, {∅})
↓ read
file_a (private, {B})

3D lattice nodes: $private, \{A,B\}$, $private, \{A\}$, $private, \{B\}$, $private, \{\emptyset\}$, $public, \{A,B\}$, $public, \{A\}$, $public, \{B\}$, $public, \{\emptyset\}$

EITA25 - Computer Security 29

Example

- Object low watermark policy

time t : Alice (public, {A})
↓ modify
file_a (private, {B})

time $t+I$: ?

3D lattice nodes: $private, \{A,B\}$, $private, \{A\}$, $private, \{B\}$, $private, \{\emptyset\}$, $public, \{A,B\}$, $public, \{A\}$, $public, \{B\}$, $public, \{\emptyset\}$

EITA25 - Computer Security 30

Example

- Object low watermark policy

time t : Alice (public, {A})
↓ modify
file_a (private, {B})

time $t+I$: Alice (public, {A})
↓ modify
file_a (public, {∅})

3D lattice nodes: $private, \{A,B\}$, $private, \{A\}$, $private, \{B\}$, $private, \{\emptyset\}$, $public, \{A,B\}$, $public, \{A\}$, $public, \{B\}$, $public, \{\emptyset\}$

EITA25 - Computer Security 31

Invocation

- A subject may invoke another subject to access an object
- Invoke property

Subject s_1 can invoke subject s_2 only if $f_S(s_2) \leq f_S(s_1)$
- Only invoke subjects (e.g., software tools) at lower levels
 - Otherwise dirty subjects could use clean tools to alter clean objects
 - But maybe this is what we want? **Controlled invocation!**
- Ring property

A subject s_1 can read all objects. It can only modify objects with $f_O(o) \leq f_S(s_1)$ and it can invoke subject s_2 only if $f_S(s_1) \leq f_S(s_2)$

EITA25 - Computer Security 32

Chinese Wall Model

- ▶ Proposed by Brewer and Nash, 1989
- ▶ Aimed at consultancy business
- ▶ Based on avoiding conflicts of interest
- ▶ Motivation:
 - A business consultant should not give advice to Volvo if he has insider knowledge about Scania.
 - A business consultant can give advice to both Volvo and H&M since they are not competitors.

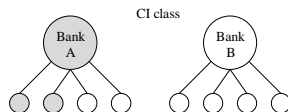
Elements of the Model

- ▶ **Company** denoted $c \in C$
- ▶ **Subjects** $s \in S$ are the analysts having access to company information
- ▶ **Objects** $o \in O$ are items of information, each belonging to a company
- ▶ **Company dataset** are all objects concerning a company
 - Function $y : O \rightarrow C$ maps object to its company dataset
- ▶ **Conflict of interest class** indicates which companies are in competition
 - Function $x : O \rightarrow P(C)$ maps object to its conflict of interest class, an element in the powerset of C
- ▶ **Security label** is a pair $(x(o), y(o))$
 - Label is $(\emptyset, y(o))$
- ▶ **Matrix** $N_{s,o}$ records history of subjects actions (true or false)

Prevent Direct Information Flow

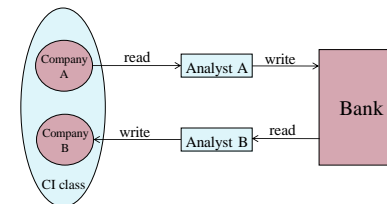
- ▶ Access granted only if object belongs to
 - A company data set already accessed by the subject **or**
 - A different conflict of interest class than previous objects
- ▶ ss-property

A subject s is permitted to access an object o only if for all objects o' with $N_{s,o'} = \text{true}$, $y(o) = y(o')$ or $y(o) \notin x(o')$



An analyst with access to grey shaded areas, will have access to other objects in Bank A data set, but not Bank B dataset

Indirect Information Flow



- ▶ Analyst A updates bank information about company A
- ▶ Analyst B can read this bank information and write to an object in company B

Avoiding Indirect Information Flow

- ▶ *-property regulates write access

A subject s can write to an object o , only if s has no read access to an object o' with $y(o) \neq y(o')$ and $x(o') \neq \emptyset$

Other data set

Unsanitized data

- ▶ Very restrictive: If you can read sensitive information in one company, you can not write to objects in any other company – ever

Clark-Wilson Model

- ▶ Developed in 1987
- ▶ Security (integrity) in commercial systems
 - Bank system will be used as example
- ▶ Differences between military and commercial applications (according to Clark and Wilson):

Military: Data item associated with a particular level.

Commercial: Data item associated by a set of programs permitted to manipulate it.

Military: Users constrained by what they can read or write.

Commercial: Users constrained by which programs they are allowed to execute.

Clark-Wilson Model

- ▶ Consistency – data is consistent if it satisfies some given properties
 - Balance day $i = \text{balance day } i-1 + \text{deposits} - \text{withdrawals}$
- ▶ Two important concepts
 - Well formed transactions – users can only change system through programs
 - Separation of duties – User can only use a certain set of programs
 - If you can create a well-formed transaction you may not be allowed to run it
 - Users have to collaborate to manipulate data

Parts of the System

- ▶ Two kinds of data items
 - **Constrained data items** (CDI) – data items subject to integrity control. E.g., account balances.
 - **Unconstrained data items** (UDI) – data items *not* subject to integrity control. E.g., unimportant text files
- ▶ **Integrity verification procedures** (IVP) – check the integrity of CDIs. E.g., check that account balance is what it should be
- ▶ **Transformation procedures** (TP) – changes the state of the system, i.e., manipulates CDIs. E.g., deposit money, withdraw money, transfer money
- ▶ **Certification rules** – How should the system behave
- ▶ **Enforcement rules** – How do we make the system behave the way we want

Rules

- ▶ **Certification rule 1:** IVPs must ensure that all CDIs are in a valid state
- ▶ **Certification rule 2:** A TP has an associated set of CDIs. TP must transform these CDIs from valid states to valid states.

CR2 implies that nonassociated CDIs can be corrupted by a TP.

- ▶ **Enforcement rule 1:** System must maintain list of CDIs associated to each TP. Only these CDIs can be manipulated by this TP.

Not everyone should be able to run any TP

- ▶ **Enforcement rule 2:** System must associate a list of TPs with each user. A TP can not be run by a user not associated with that TP.

Now we have a set of triples (*user, TP, {CDI set}*). These must be certified.

- ▶ **Certification rule 3:** All triples must meet the separation of duties requirements

EITA25 - Computer Security

41

Rules Continued

Only certain users are allowed to run a certain TP. How do we know the user is who he claims to be?

- ▶ **Enforcement rule 3:** The system must authenticate each user trying to execute a TP.

Each operation must be logged

- ▶ **Certification rule 4:** All TPs must log information about all operations

An UDI entering the system may not be trusted

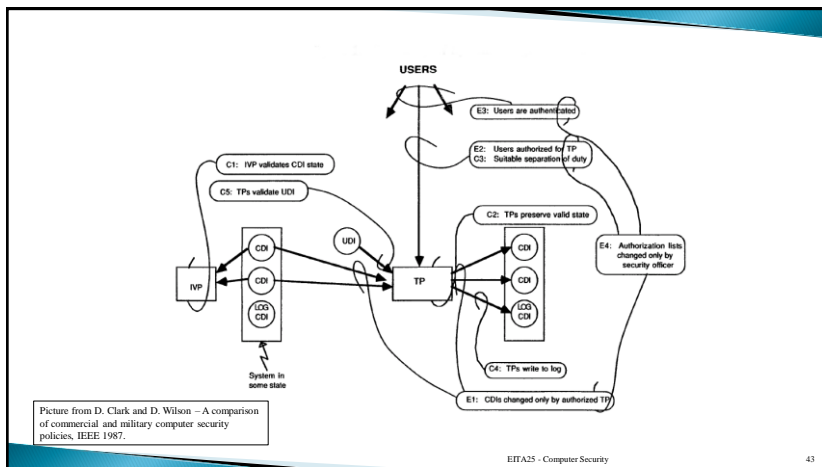
- ▶ **Certification rule 5:** A TP taking a UDI as input must either transform it to a valid CDI or reject it

No person should be able to both create and run a TP

- ▶ **Enforcement rule 4:** Only certifier of TP may change the list of entities associated with that TP. No certifier may ever execute the TP. (separation of duties)

EITA25 - Computer Security

42



Picture from D. Clark and D. Wilson - A comparison of commercial and military computer security policies. IEEE 1987.

EITA25 - Computer Security

43