CSS – CIA 2 NOTES

UNIT 2 :

Policy Requirements

1. Users use only existing production programs.
2. Programmers develop/test on non-production systems.
3. Production data access via a special process.
4. Deployment to production follows a controlled process.
5. Deployment must be audited.
6. Managers/auditors get system state & logs access.

Integrity Policies : ensure data remains accurate and unaltered by unauthorized users.

1. Biba Integrity Model (Prevents data corruption)

- Simple Integrity Property: No read down (Higher-level subjects can't read lower-level data).
- Star (*) Integrity Property: No write up (Lower-level subjects can't modify higher-level data).

Example: A manager (high-level) can't read an intern's (low-level) draft, and an intern can't edit an executive's report.

2. Low-Water-Mark Model (Flexible version of Biba)

- Allows read down but lowers subject's integrity level if they do so.
- Prevents write up (similar to Biba).

Example: If a senior engineer reads an intern's code, their integrity level drops, restricting future actions.

3. Ring Policy Model (Less restrictive)

- Allows read down (unlike Biba).
- Subjects can only write at their own level.

Example: A software developer can read an intern's documentation but can only modify files at their assigned level.

4. Lipner's Integrity Matrix (Combines Biba + Bell-LaPadula)

- No Read Down (Prevents higher-level users from reading lower-integrity data).
- No Write Up (Prevents low-level users from corrupting high-integrity data).

Example: A financial officer can access payroll data but can't modify security system configurations.

Two Security Levels:

- Audit Manager → High integrity, controls security audits.
- System Low → Lower integrity, general operations.

Five Categories:

1. Development (D) – Application development environment.
2. Production Code (PC) – Finalized software ready for deployment.
3. Production Data (PD) – Real-time operational data.
4. System Development (SD) – System-level software development.
5. Software Tools (T) – Utilities used for development and maintenance.

Security levels for subjects

| Users | Clearance |
|---|---|
| Ordinary Users | (SL,{PC,PD}) |
| Application developers | (SL,{D,T}) |
| System programmers | (SL,{SD,T}) |
| System managers and auditors | (AM{D,PC,PD,SD,T}) |
| System controllers | (SL,{D,PC,PD,SD,T}) downgrade privilege |

Security levels for objects

Objects : Development code, production code, production data, software tools, system programs, system programs in modification, system and application logs

Security Requirements of the Model

- Prevents Unauthorized Modification: Only trusted users can modify high-integrity objects.
- Separation of Duties: Developers cannot directly modify production systems.
- Auditability: The Audit Manager monitors all integrity violations.

5. Lipner's Full Integrity Matrix (Combines Biba's Integrity Model for integrity & Bell-LaPadula Model for confidentiality)

- Ensure Data Integrity: Prevent unauthorized or accidental modifications.
- Separate Development & Production: Developers can't alter production data.
- Maintain Auditability: Security logs must be protected from modification.

Lipner's Full Matrix Integrity Model

| Category | ISP (System Program) | IO (Operational) | ISL (System Low) |
|---|---|---|---|
| ID (Development) | SSD (System Development) | SD (Development) | SD (Development) |
| IP (Production) | SP (Production) | SP (Production) | SD (Development) |

Key Points:

- Integrity Levels:
  - ID (Development) → Lower integrity (less trusted).
  - IP (Production) → Higher integrity (more trusted).

- Access Control:
    - System Development (SSD) → Modifies system programs.
    - Development (SD) → Can access but not modify production.
    - Production (SP) → Can operate and modify production.
- Ensures:
    - Separation of duties.
    - Protection of production data.
    - Controlled development and deployment.

Example Scenario:

- A developer writes software in Development (D).
- The code is reviewed & approved, then moved to Production Code (PC).
- A user runs the program, which accesses Production Data (PD), but cannot modify it.
- The Audit Manager monitors logs for any violations.

6.Clark-Wilson Integrity Model

- Key Concepts:
    - CDIs (Constrained Data Items): Data with integrity controls.
    - UDIs (Unconstrained Data Items): Data without integrity controls.
    - IVPs (Integrity Verification Procedures): Ensure CDIs are valid.
    - TPs (Transaction Procedures): Move system from one valid state to another.
- Certification Rules (CR):
    - CR1: IVPs must ensure all CDIs are valid.
    - CR2: TPs must transform CDIs from one valid state to another.
    - CR3: Enforces separation of duties.
    - CR4: TPs must log operations.
    - CR5: TPs must validate UDIs before converting them into CDIs.
- Enforcement Rules (ER):
    - ER1: Only certified TPs can modify CDIs.
    - ER2: Users must be associated with specific TPs and CDIs.
    - ER3: Users must be authenticated before executing TPs.
    - ER4: Certifiers of TPs cannot execute them, enforcing separation of duty.
- Comparison with Biba Model:
    - Biba: Multilevel integrity, no explicit certification.

- Clark-Wilson: Focuses on transactions, separation of duties, and certified transformations.

Ensures data integrity via well-defined processes, transaction validation, and separation of duties.

HYBRID POLICIES : Overview of Security Models

Chinese Wall Model

- Goal: Prevent conflicts of interest in business consulting and financial sectors.
- Solution: Group companies into Conflict of Interest (COI) classes and control access.
- Example:
  - Tony advises Bank A; he cannot advise Bank B if they are in the same COI class.

Rules:

1. CW-Simple Security Rule (Read Rule)
   - A user can read only within a company dataset (CD) they already accessed OR
   - They must not have accessed any competing company's dataset.
2. **CW-Property (Write Rule)*
   - A user can write only if all the datasets they read belong to the same company.
3. Sanitization:
   - Public (sanitized) data is freely accessible.

Comparison to Other Models

- Bell-LaPadula: CW has no security labels, but tracks past access history.
- Clark-Wilson: If a person uses different processes, CW rules may be bypassed.

---

Clinical Information Security (CISS Policy)

- Focus: Integrity & confidentiality in medical records.

Key Principles:

1. Access Control List (ACL): Only approved clinicians can read/write records.
2. Responsible Clinician: Can add others to ACL and must notify patients.
3. Audit Logs: Track who accessed what, when, and why.
4. Data Deletion: Records cannot be deleted until the appropriate time.
5. Data Sharing (Confinement Principle): Information can be copied only if the new record's ACL is a subset of the original.

6. Aggregation Protection: Patients must be notified if someone gains access to many records (prevents misuse).
7. Enforcement & Auditing: Must be independently evaluated.

Comparison to Other Models

- Bell-LaPadula: Similar confinement rules, but B-LP focuses on subjects, while CISS focuses on objects (medical records).
- Clark-Wilson:
    - Medical records = CDIs (Constrained Data Items).
    - Updates = TPs (Transformation Procedures).
    - Integrity checks ensure only clinicians validate records.
    - Auditing ensures records are append-only.

 Chinese Wall Model prevents conflicts of interest in business.

 CISS Policy ensures medical record security through access control and auditing.

 RBAC & ORCON provide flexible access control mechanisms.

 Bell-LaPadula vs. Clark-Wilson:

- B-LP controls who can access data.
- C-W controls how data is modified and ensures integrity.

ORCON (Originator Controlled Access Control)

- Goal: Control how information is shared and copied after creation.
- Example: A government official writes a memo and must approve any further sharing.

Rules:

1. Originator (X) controls dissemination.
2. No sharing with new organizations without originator's approval.
3. Copies inherit the same access restrictions.

Why MAC & DAC Fail?

- DAC issue: The owner can freely change permissions, violating rule #2.
- MAC issue: Requires creating new categories for every access scenario, making it impractical.

Solution: Hybrid Approach

- MAC: Access restrictions cannot be changed by the owner.
- DAC: Originator can grant or deny permissions per subject/object.

RBAC (Role-Based Access Control)

- Goal: Access depends on job role, not individual identity.
- Example:
    - Allison (Math Dept bookkeeper) can access financial records.
    - When Betty replaces her, Betty gets access, not Allison.

Key Concepts:

- Role (r): A set of job functions.
- Transactions (trans(r)): Actions allowed for a role.
- Active Role (actr(s)): Role a subject is currently using.
- Authorized Roles (authr(s)): Roles a subject can take.

Rules:

1. Role Assignment: A subject must have a role to execute a transaction.
2. Role Authorization: A subject can only assume an assigned role.
3. Transaction Authorization: A subject can only perform actions allowed for its role.

Advanced Features:

- Role Containment: Higher roles include permissions of lower roles (Trainer > Trainee).
- Separation of Duty: Some roles must be mutually exclusive (e.g., a person cannot be both an auditor and an accountant).

- ORCON: Controls who can share copies of information (Hybrid of MAC & DAC).
- RBAC: Controls who can perform which actions based on job roles.
- Separation of Duty: Prevents conflicts by restricting multiple roles for a single user.

NON-INTERFERENCE and POLICY COMPOSITION

Composition Problem in security policies:

- Composition problem: How to merge multiple security policies into one coherent and secure policy.

- Important for distributed systems, merged organizations, and integrated networks.

- Challenges arise when policies conflict or interact unexpectedly.

1. Problem Statement

- Two organizations with different security policies merge.

- How do we create a unified, consistent security policy?

- Can the new policy maintain security guarantees of individual policies?

2. Policy Composition Challenges

Noninterference

- Definition: A high-security level should not affect a low-security level's outputs.

- Example: A government system should prevent classified info (HIGH) from influencing unclassified outputs (LOW).

HIGH Inputs Affect LOW Outputs

- Risk: If high-level actions cause changes in observable low-level behavior, a security breach occurs.

- Example: A CPU scheduling difference due to a classified process may leak information to an unclassified observer.

Nondeducibility

- Definition: A LOW-level observer should not be able to infer HIGH-level inputs from observable outputs.

- Example: If a stock market system delays updates based on insider trades (HIGH), external traders (LOW) might infer secret transactions.

Restrictiveness

- When can policies be successfully merged?

1. Autonomy Principle: If access is allowed in any original policy, it should be allowed in the merged policy.

2. Security Principle: If access is forbidden in any original policy, it must be forbidden in the merged policy.

3. Policy Composition Process

Composition of Bell-LaPadula

- Why? Some security standards require secure components to form a secure system.

- Key Question: Under what conditions is this secure?

Assumptions

- Systems implement security policies precisely.

- Must compose security lattices to merge labels and access rules.

Lattice Composition Example

- Assume clearances: LOW < S < HIGH < TS.

- Assume categories: SOUTH, EAST, WEST.

- Merged system must reflect relationships among different levels and categories.

Different Policies

- What does “secure” mean in the merged system?

- Which component's policy takes priority?

- Solutions:

1. **Autonomy** → Allow access if any original policy allowed it.

2. **Security** → Forbid access if any original policy forbade it.

4. Policy Composition Example (Gong & Qian)

- **System X**: Bob cannot access Alice’s files.

- **System Y**: Eve and Lilith can access each other’s files.

- **Merged Policy**: Bob can access Eve’s files, Lilith can access Alice’s files.

Key Question: Can Bob access Lilith’s files?

Solution (Gong & Qian Approach)

1. Define Access Sets (AS):

- AS(X) = ∅

- AS(Y) = {(Eve, Lilith), (Lilith, Eve)}

- AS(X∪Y) = {(Bob, Eve), (Lilith, Alice), (Eve, Lilith), (Lilith, Eve)}

2. Compute Transitive Closure (All Implied Accesses):

- AS(X∪Y)+ = {(Bob, Eve), (Bob, Lilith), (Bob, Alice), (Eve, Lilith), (Eve, Alice), (Lilith, Eve), (Lilith, Alice)}

3. Delete Conflicting Accesses: Remove (Bob, Alice) since it conflicts with System X.

4. Result: Bob can access Lilith’s files.

- Composition can **introduce unintended accesses**.

- Transitive closure **must be computed**.

- Forbidden accesses must be **explicitly removed**.

5. Interference & Covert Channels

Example: Covert Channel via CPU Usage

- Scenario:

- Holly (HIGH) & Lara (LOW) use separate virtual machines.

- They share the same CPU, but load-based scheduling creates an **observable pattern**.

- Holly runs a program to create high CPU usage → Lara detects it → Secret information leaks.

- Solution Considerations:

  - Prevent covert channels by strict resource allocation.

  - Modify policy definitions to include unintended interference.

6. Formal Model for Security Composition

- System as State Machine:

  - Subjects S = {si}

  - States Σ = {σi}

  - Commands Z = {zi}

  - Outputs O = {oi}

- Functions:

  - State Transition Function: T: C × Σ → Σ (Defines system evolution)

  - Output Function: P: C × Σ → O (Defines what is visible)

- Projection & Purge Mechanism:

  - Used to remove unauthorized outputs when merging policies.

7. Key Insights

- Composition is Hard:

  - Combining security policies often allows unintended behaviors.

  - Some policies may dominate others, affecting security guarantees.

- Mathematical Modeling is Key: Using state machines, projections, and closures helps reason about composition security.

NON INTERFERNCE :

Noninterference

- intuition: outputs visible to one user should only depend on inputs they can see
- formal definition: if a group G executes commands in set A, they are noninterfering with another group G' if outputs for G' remain unchanged
- notation: A,G :| G'

Example:

- initial state σ0 = (0,1), commands = [(Heidi, xor0), (Lucy, xor1), (Heidi, xor1)]
- checking projection for Lucy shows Heidi's commands affect her outputs, so noninterference is false
- modifying Heidi's commands to affect only H-bit ensures noninterference holds

Security policy

- defines authorized and unauthorized states
- ensures no forbidden interferences occur
- can be expressed as a set of noninterference assertions

Output-consistency

- states are output-consistent if, for a given command, they produce the same output for subjects in its domain
- ensures that executing a command does not leak information to unauthorized domains

Protection domains

- system partitions into protection domains D = {d1, d2, ..., dn}
- relation r defines information flow rules between domains

Projection function

- π′ removes commands that do not interfere with a domain
- if executing a command interferes with domain d, it remains visible in π′d

Noninterference security

- a system is noninterference-secure if executing commands produces the same output as their projection
- ensures no unauthorized information flow

Lemma and proof

- if state transitions remain equivalent after projection and output-consistency holds, system is noninterference-secure
- follows from the definition of output-consistency and noninterference

UNWINDING THEOREM :

- Links security of sequences of state transitions to security of individual transitions
- Helps prove ML security by checking if system meets conditions from known lemmata
- Does not guarantee system security due to implementation and operational issues locally respects
- system X locally respects policy r if applying a command c does not affect domain d when it should not
- ensures no unintended side effects on unrelated domains

Transition-consistent

- system is transition-consistent if applying command c does not change the equivalence of states under r
- ensures consistency in how commands affect the system

Lemma

- if two commands c1 and c2 allow information flow into d, their execution order does not matter
- T*(c1c2, σ) = T(c1, T(c2, σ)) = T(c2, T(c1, σ))

Theorem

- if a system is output-consistent, transition-consistent, and locally respects r, then it is noninterference-secure under r
- provides a basis for verifying system designs against noninterference policies

Proof outline

1. use induction on the sequence of commands cs
2. base case: cs is empty, holds trivially
3. assume hypothesis holds for cs
4. for cs followed by cn+1, consider two cases:
    - dom(cn+1)rd holds: apply transition consistency and induction hypothesis
    - dom(cn+1)rd does not hold: apply local respect property
5. conclude that T*(cs, σ0) ~d T*(π'd(cs), σ0)
6. since X is output-consistent, it follows that X is noninterference-secure

Access Control Matrix

Example of Interpretation

- Given: Access control information
- Question: Are given conditions enough to provide noninterference security?
- Assume: System in a particular state
- Encapsulates: Values in ACM

ACM Model

- Objects: L = { $l_1$, ..., $l_m$ } (Locations in memory)
- Values: V = { $v_1$, ..., $v_n$ } (Values that L can assume)
- Set of States: Σ = { $\sigma_1$, ..., $\sigma_k$ }
- Set of Protection Domains: D = { $d_1$, ..., $d_j$ }

Functions

- value: $L \times \Sigma \rightarrow V \rightarrow$ Returns value *v* stored in location *l* when system in state $\sigma$
- read: $D \rightarrow 2^L \rightarrow$ Returns set of objects observable from domain *d*
- write: $D \rightarrow 2^L \rightarrow$ Returns set of objects writable from domain *d*

Interpretation of ACM

- Functions represent ACM
- Subject *s* in domain *d* = *dom(c)*, object *o*
    - $r \in A[s, o] \Leftrightarrow o \in read(d)$
    - $w \in A[s, o] \Leftrightarrow o \in write(d)$

Equivalence Relation

$\sigma_a \sim_{dom(c)} \sigma\_b \Leftrightarrow \forall l_i \in read(d)$
$value(l_i, \sigma_a) = value(l_i, \sigma\_b)$

- You can read exactly the same locations in both states, where $\sigma\_b = T(c, \sigma_a)$

Enforcing Policy *r*

Five Requirements

- Three general ones: Describe dependence of commands on rights over input and output
    - Hold for all ACMs and policies
- Two specific ones: Apply to some security policies
    - Hold for most policies

First Requirement

- Output of command *c* executed in *dom(c)* depends only on values for which subjects in *dom(c)* have read access
- $\sigma_a \sim_{dom(c)} \sigma\_b \Rightarrow P(c, \sigma_a) = P(c, \sigma\_b)$

Second Requirement

- If *c* changes $l_i$, then *c* can only use values of objects in read(dom(c)) to determine the new value
- If $l_i$ can't be read in *dom(c)*, then values in $l_i$ may differ after *c* is applied to $\sigma_a$ or $\sigma\_b$

Third Requirement

- If *c* changes $l_i$, then *dom(c)* provides subject executing *c* with write access to $l_i$
- $value(l_i, T(c, \sigma_a)) \neq value(l_i, \sigma_a) \Rightarrow l_i \in write(dom(c))$

Fourth Requirement

- If domain *u* can interfere with domain *v*, then every object that can be read in *u* can also be read in *v*
- u r v ⇒ read(u) ⊆ read(v)

Fifth Requirement

- If subject *s* can write object *o* in *v* and subject *s'* can read *o* in *u*, then domain *v* can interfere with *u*
- $l_i \in$ read(u) and $l_i \in$ write(v) ⇒ v r u

Theorem

- If X satisfies the five conditions, then X is noninterference-secure with respect to *r*
- Proof: Must show X is:
    1. Output-consistent
    2. Locally respects *r*
    3. Transition-consistent
- Then, by the Unwinding Theorem, the theorem holds

Output-Consistency

- Take equivalence relation ~d, first condition is the definition of output-consistent

Local Respect for *r*

- Proof by contradiction
    - Assume (dom(c), d) ∉ *r* but $\sigma_a$ ~d T(c, $\sigma_a$) does not hold
    - Some object has value changed by *c*
    - Condition 3: $l_i \in$ write(d)
    - Condition 5: dom(c) r d, contradiction
    - So, $\sigma_a$ ~d T(c, $\sigma_a$) holds, meaning X locally respects *r*

Transition-Consistency

- Assume $\sigma_a$ ~d σ_b
- Must show value($l_i$, T(c, $\sigma_a$)) = value($l_i$, T(c, σ_b)) for $l_i \in$ read(d)

Case 1

- value($l_i$, T(c, $\sigma_a$)) ≠ value($l_i$, $\sigma_a$)
- Condition 3: $l_i \in$ write(dom(c))
- Condition 5: dom(c) r d
- Condition 4: read(dom(c)) ⊆ read(d)
- Apply Condition 2, so T(c, $\sigma_a$) ~dom(c) T(c, σ_b)

Case 2

- value($l_i$, T(c, $\sigma$_b)) $\neq$ value($l_i$, $\sigma$_b)
- Follows the same steps as Case 1

Case 3

- Neither of the previous two cases
- value($l_i$, T(c, $\sigma_a$)) = value($l_i$, $\sigma_a$)
- value($l_i$, T(c, $\sigma$_b)) = value($l_i$, $\sigma$_b)
- Then T(c, $\sigma_a$) ~d T(c, $\sigma$_b)

Conclusion: In all three cases, X is transition-consistent

Policies Changing Over Time

- Problem: Previous analysis assumes a static system
- Reality: ACM changes as system commands are issued
- Solution: Condition noninterference on cando(w, s, z)
- Example: If ¬cando(w, Lara, "write f"), then Lara can't interfere with any user by writing f

Generalized Noninterference

- G $\subseteq$ S (Group of subjects)
- A $\subseteq$ Z (Set of commands)
- p (Predicate over elements of C* )
- $\pi''$: Removes commands from A executed by G when p holds

Intuition

- $\pi''$(cs) = cs, except when p holds
- If p holds, commands from A by members of G are replaced with v
- Effect: Similar to deleting entries from cs

Noninterference Definition

- Users in G executing A are noninterfering with G′ under condition p iff:
    - $\forall$cs $\in$ C*, $\forall$s $\in$ G′, proj(s, cs, $\sigma_i$) = proj(s, $\pi''$(cs), $\sigma_i$)
- Notation: A, G :| G′ if *p*

Example: Security Policy

- If s can't execute z, then s can't use z to interfere
- $\forall$(s $\in$ S) $\forall$(z $\in$ Z) [ {z}, {s} :| S if ¬cando(w, s, z) ]

## Noninterference with Right Transfers

- Problem: Rights can be passed dynamically
- Example: *pass(s, z)* gives *s* right to execute *z*
- Policy:
    - No subject *s* can use *z* to interfere if, in the previous state, *s* did not have the right to *z*, and no subject gave it to *s*
- Effect:
    - Ensures that a subject's first execution of *z* does not affect other subjects' observation of the system

## Key Ideas in Transition Consistency and Noninterference

### Transition Consistency

- If two states, σa and σb, are indistinguishable under a dependency d, their transitions should also be indistinguishable.
- This is shown by checking three cases where a transition T(c, σ) affects a variable li:
    1. If li changes → Conditions ensure σa and σb remain consistent.
    2. If li changes in the other state → Similar reasoning applies.
    3. If li doesn't change → The property holds naturally.

### Policies Changing Over Time

- Real systems update access controls dynamically, unlike static models.
- Example: A user w gets new permissions, affecting their ability to execute commands.
- Noninterference Rule: If a user can't execute a command, they can't interfere with others.

### Noninterference & Security Policies

- Generalized Noninterference: Users in G executing commands in A don't affect users in G', ensuring security.
- Example: If a user lacks write access, their actions shouldn't impact other users.
- Extending to Dynamic Rights: If a user gains permission at time t, it must be verified how they got it.

### Security and Composition

- Combining secure systems can break security if timing issues arise.
- Example: A low-level observer (Lara) might infer high-level actions (Heidi) from indirect effects.

- Feedback-Free Systems: If a system has no loops, combining secure systems remains secure.

Nondeducibility vs. Strong Noninterference

- Nondeducibility: Low-level users can't infer high-level inputs.
- Strong Noninterference: Even the timing of high-level events can't be observed.
- Example: If a system leaks information via subtle timing differences, it fails strong noninterference.

Conclusion

- Security models ensure users can't infer hidden data.
- Dynamic policies and composition of systems introduce challenges.
- Solutions: Feedback-free structures, delayed updates, and strict access rules maintain security.

Restrictiveness & Composition in Security Systems

Restrictiveness: A system is restrictive if it satisfies four key properties (previously discussed).

Composition

- Idea: If two restrictive systems are combined, the result should also be restrictive.
- Why? By properties 3 and 4, a high-level output followed by a low-level output behaves like a low-level input, so security should hold.

Composite System

- System M1's outputs become M2's inputs.
- A composite system consists of paired states from M1 and M2:
    - $\mu 1i, \mu 2i \rightarrow$ States of M1, M2.
    - $(\mu 1i, \mu 2i) \rightarrow$ A state in the composite system.

Transition Conditions

An event e causes a transition in the composite system if any of these hold:

1. M1 changes: M1 moves from $\mu 1a \rightarrow \mu 1b$, e doesn't affect M2 ($\mu 2a = \mu 2b$).
2. M2 changes: M2 moves from $\mu 2a \rightarrow \mu 2b$, e doesn't affect M1 ($\mu 1a = \mu 1b$).
3. Both change: M1 moves $\mu 1a \rightarrow \mu 1b$ and M2 moves $\mu 2a \rightarrow \mu 2b$, where e is an input to one and an output from the other.

Intuition

- Any event in the composite system must trigger at least one component's transition.

- If only one component changes, then the event must be irrelevant to the other component when they are not connected.

Equivalence for Composite Systems

- Define an equivalence relation:
    - $(\sigma a, \sigma b) \equiv_C (\sigma c, \sigma d)$ if $\sigma a \equiv \sigma c$ and $\sigma b \equiv \sigma d$.
- This matches the equivalence relation in Property 2 for individual components.

Conclusion

- Restrictive systems remain restrictive when composed properly.
- Transitions follow predictable rules, ensuring secure information flow.