## Chapter 1:

Three basic components of computer security:

*Confidentiality*: The concealment of information or resources.

*Integrity*: The trustworthiness of data or resources, preventing unauthorized changes.

*Availability:* The ability to use information or resources (reliability).

Threat Model:

Classifications: Deception (fake news), Disruption (prevent operation), Usurpation (unauthorized control), Disclosure.

Characterizations: Alteration, Spoofing, Repudiation, Denial of Receipt, Delay, Denial of Service.

Policy and Mechanism:

*Security Policy*: A statement of what is, and what is not allowed.

*Security Mechanism*: A method, tool or procedure for enforcing a security policy.

*Policy Model*: A model that represents a particular policy or class of policies.

Assumptions and Trust:

*Trust*: Your belief the system is trustworthy.

*Assurance*: Level at which the security mechanism implements the policy.

Let P be the set of all possible states, Q be the set of secure states, R be the set of states restricted by the security system.

A security mechanism is *secure* if R in Q, *precise* if R = Q, and *broad* if there are some states r not in Q.

*Specification*: A statement of the desired functioning of the system.

A system is said to *satisfy* a specification if the specification correctly states how the system will function.

*Design*: Translates the specifications into components that will implement them.

*Implementation*: Creates a system that satisfies the design.

A program is *correct* if its implementation performs as specified.

## Chapter 2: ACM

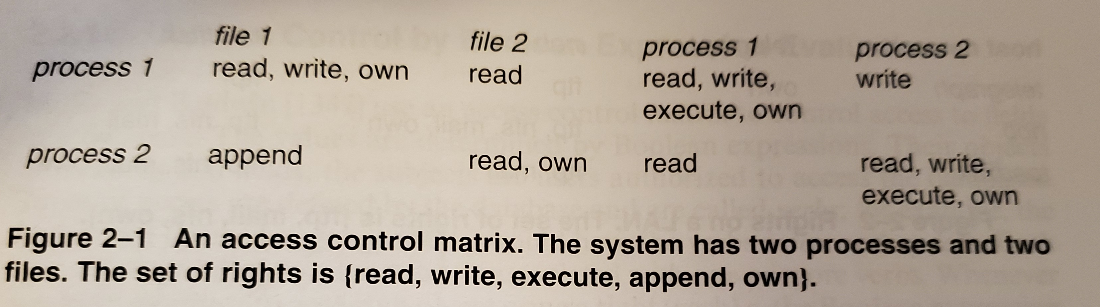
Protection State:

*State*: The collection of the current values in memory locations.

*Access Control Matrix Model*: A tool that can describe the current protection state. Which Describes the rights of subjects over all entities in the matrix.

Consider the set of protection states P. Some subset Q of P consists of exactly those states in which is the system is authorized to reside.

When a command changes the state of the system a *state transition* occurs.



Protection State Transitions: Let the intial state of the system be X\_0 = (S\_0, O\_0, A\_0) the set of state transitions is represented as a set of operations t,t2... successive states are represented as X\_1, X\_2,… where the notation X\_i (sideways T) \_(t\_i + 1) means that sate transition (t\_i + 1) moves the system from state X\_i to state X\_i + 1.

Harrison, Ruzzo, Ullman (HRU):

Primitive Commands:

1. Precondition: s not in S.
   1. Command: **create subject** s.
2. Precondition: o not in O.
   1. Command: **create object** o.
3. Precondition: s in S, o in O, r in R.
   1. Command: **enter** r **into** a[s,o].
4. Precondition: s in S, o in O, r in R.
   1. Command: **delete** r froma[s,o].
5. Precondition: s in S.
   1. Command: **destroy subject** s.
6. Precondition: o in O.
   1. Command: **destroy object** s.

Example:

**command** create\_file(p, f)

create **object** f;

**enter** *own***into** A[p,f];

**enter** *r***into** A[p,f];

**end**

**command** grant\_read\_file(p, f, q)

**if** *r* **in** A[p,f] **and** *g* **in** A[p,f]

**Then**

**enter** *r***into** A[p,f];

**end**

Note commands cannot have or negation like not in operators otherwise they would be two commands.

*Copy Right*: Often called the grant right allows the processor to grant rights to another.

*Own Right*: Enables the possessors to add or delete privileges.

*Principle of Attenuation of Privilege*: A subject may not increase its rights, nor grant rights that it does not possess to another subject.

## Chapter 3 (Foundational Results):

*Leaked*: When a generic right *r* is added to an element of the ACM that did not contain *r* initially, that right is said to be *leaked.*

*Safe System*: If a system can never leak the right *r*, the system is called *safe* with respect to right *r*. Alternatively is *unsafe with respect to right r.*

**Theorem 3.1:** There exists an algorithm that will determine whether a given mono-operation protection system with initial sate *s\_0* is safe with respect to generic right r.

You can omit the **delete, destroy, and create** commands beyond the first sequence.

**Theorem 3.1:** It is undecidable weather a given state of a given system is safe for a given generic right.

Proof by the Turing equivalent halting problem.

HRU in general undecidable – must try all possible command sequences.

Delete the **create** command, then the problem becomes complete in **P-SPACE.**

Delete the destroy, delete commands then the safety question is still undecidable.

Monotonic protection systems is decidable.

Mono conditional protection systems with **create, enter, delete (no destroy)** is decidable.

Take Grant Protection Model:

Subjects = Black Dot.

Objects = White Dot.

Subject or Object (doesn’t matter) = Dot with X filled in.

G |–x G' apply a rewriting rule x (witness) to G to get G'

G |–\* G' apply a sequence of rewriting rules (witness) to G to get G'

R = { t, g, r, w, ... } set of rights

These four rules are called the de jure rules:

* *take rule* allows a subject to take rights of another object (add an edge originating at the subject)
  + Preconditions:
    - subject *s* has the right Take for *o*.
    - object *o* has the right *r* on *p*.
* *grant rule* allows a subject to grant own rights to another object (add an edge terminating at the subject)
  + Preconditions:
    - subject *s* has the right Grant for *o*.
    - *s* has the right *r* on *p*.
* *create rule* allows a subject to create new objects (add a vertex and an edge from the subject to the new vertex)
* *remove rule* allows a subject to remove rights it has over on another object (remove an edge originating at the subject)

## 

*Island:* A maximal *tg*-*connected* subject-only subgraph.

*Bridge:* Connects two islands, special type of path that has *tg* between islands.

*Initially Span*: x` = x where x` is a subject. Or x` is a subject and there is a *tg* path between x` and x.

* In other words, **x** initially spans to **y** if **x** can grant a right it possesses to **y**.

*Terminally Span*: There is s` = subject and s` = s. Or there is a *tg* path between s` and s.

* In other words, **x** terminally spans to **y** if **x** can take any right that **y** possesses.

*Can.Share(r,* ***x, y,*** *G\_0):* If and only if there is a sequence of protection graphs G\_0, … G\_n such that G\_0 |–\* G\_n using only *dejure* rules in G\_n there is an edge from **x** to **y** labeled *r.*

can•share(α, **x, y,** G\_0) if, and only if, there is an edge from **x** to **y** labeled α in G\_0.

Or the following hold simultaneously:

•There is an **s** in G\_0 with an **s**-to-**y** edge labeled α.

•There is a subject **x`** = **x** or **initially spans** to **x.**

•There is a subject **s`** = **s** or **terminally spans** to **s.**

•There are islands **I\_1, ..., I\_k** connected by bridges, and **x`** in **I1** and **s`** in **I\_k**.

*Stealing:* A notion which no owner of any right over an object specifically grants that right to another.

can•steal holds iff

* There is no edge from **x** to **y** labeled in G0.
* There exists a subject vertex **x’** such that **x’** **initially spans** to x (*tg*-path between **x’**, **x** with word in { *t*\**g* } ∪ { ν }).
* There exists a vertex **s** with and edge labeled α to **y**in G0 and for which CanShare (t,**x,s**,G0) holds.

This can be done using the following sequence of events:

1. **x`** creates (*g* to new subject) **x``.**
2. **x`** grants (t to **s**) to **x``.**
3. **x`** grants (g to **x**) to **x``.**
4. **x``** takes ( to **y**) from **s.**
5. **x``** grants ( to **y**) to **x.**

Conspiracy Graphs and Access Sets:

*Access Set*: The *access set A(****y****)**with focus* ***y*** (subject)is the set of all vertices **x** to which **y** initially spans and all vertices **x`** to which **y** terminally spans**.**

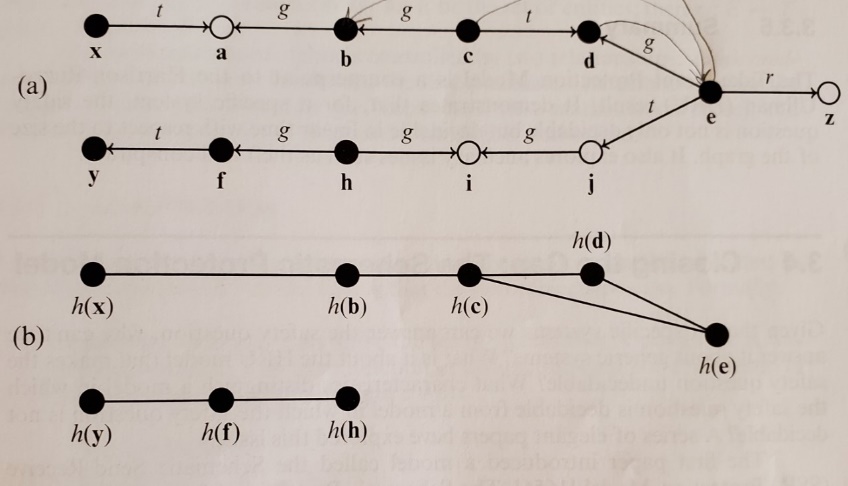
*Deletion Set:* ‏The set of all verticies that can be removed from the graph without affecting transfers.

δ(**y, y`**) contains all vertices **z** in the set A(**y**) ⋂ A(**y`**) for which:

1. **y** initially spans to **z** and **y`** terminally spans to z**.**
2. **y** terminally spans to **z** and **y`** initially spans to **z**.
3. **z** = **y.**
4. **z = y`.**

For example the shortest path between h(e) and h(x) has four vertices (h(x), h(b), h(c), and h(e)) so four conspirators are necessary to witness can.share(r, x, z, G\_0).

1. G grants (r to z) to d.
2. C takes (r to z) from d.
3. C grants (r to z) to b.
4. B grants (r to z) to a.
5. X takes (r to z) from a.



1. The tg protection graph.
2. The corresponding conspiracy graph.

## Chapter 4 (Security Policies):

*Security Policy*: A statement that partitions the states of the system into a set of *authorized, or secure,* states and a set of *unauthorized or nonsecure,* states.

*Secure System*: A system that starts in an authorized state and cannot enter an unauthorized state.

*Breach of Security*: When a system enters an *unauthorized* state.

*Confidentiality*: Let I be some information. Then I have *confidentiality* with respect to X if no member of X can obtain information about I.

*Integrity*: Let X be a set of entities and let I be some information. Then I has the property of *integrity* with respect to X if all members of X trust I.

*Availability*: Let X be a set of entities and let I be a resource. Then I has the property of *availability* with respect to X if all members of X can access I.

*Information Flow*: The leakage of rights and the illicit transmission of information without leakage of rights.

*Confidentiality Policy*: The policy that outlines the authorized access of information.

*Integrity Policy:* The parts of the security policy that describe the conditions and manner in which data can be altered.

*Military Security Policy:* Primarily provides confidentiality.

*Commercial Security Policy:* Primarily provides integrity.

## Chapter 5 (Confidentiality Policies):

Bell-LaPadula Model:

No read up, no write down. Restricts flow from High to low.

**Discretionary Access:** Allow or deny a user access to an object.

**Mandatory Access:** System controls access to an object.

L(s) = clearance level of subject s.

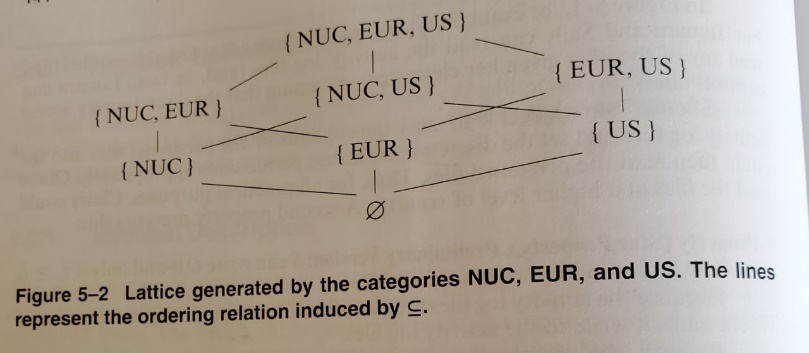
L(o) = clearance level of object o.

**Simple Security Condition, Preliminary Version:** S can read O, iff l\_o <= l\_s and S has discretionary read access to O.

**\*-Property (Star Property), Preliminary Version:** S can write to O, iff l\_o >= l\_s and S has discretionary write access to O.

***Basic Security Theorem, Preliminary Version:*** Let S be a system with a secure initial state s\_0, and let T be a set of state transformations. If every element of T preservers the simple security condition, and the \*-property, then every state s+i, I >= 0 is secure.

The model starts off with simple security clearances but can be expanded to include categories.



**Dominates:** The security level (L, C) dominates the security level (L`, C`),

written (L,C)*dom*(L`, C`), iff L` <= L and C` in C.

**Simple Security Condition:** S can read O iff S *dom* O and S has discretionary read access to O.

**\*-Property (Star Property):** S can write to O iff O *dom* S and S has discretionary write access to O.

Formal model:

* S = subjects
* O = objects
* P = rights
* M = ACMs
* C, K = Clearences, Categories
* (f\_s, f\_0, f\_c) = max security level, object level, current subject security level.

Basic Security Theorem:

Combines the simple security condition, \*-property, and discretionary security property. To satisfy the simple security condition, either s cannot read o or the security level of s must dominate that of o.

*Principal of Tranquility*: Subjects and objects may not change their security levels once that have been instantiated.

*Strong*: Security levels do not change during the lifetime of the system.

*Weak*: Security levels do not change in a way that violates the security policy.

Controversy Over the BLP Model: Given assumptions known to be unsecure the basic security theorem could prove a nonsecure system to be secure. The real lasting concept behind the controversy is that you can interpret McClean’s result as an integrity property and that BLP and Biba are duals of each other.

McLeans Star Property: The \*-property holds for a subject s and an object o if, whenever s has w rights over o, the clearance of s dominates the classification of o. This is the reverse of the normal \*-property which holds that the classification of o would dominate s.

## Chapter 6 (Integrity Policies):

Restricts loss of integrity.

Aggregation: Does declassifying information prevent information flow through deducing? Cannot restrict in BLP.

Biba Model:

The dual of the BLP model, a system consists of S subjects, O objects, and I integrity levels. The levels are ordered, the higher the level the more confidence one has that the program will execute correctly. Data at a higher level is more accurate, incorporates the notion of “trust”.

*Information Transfer Path:* A sequence of objects o\_1,…o\_n+1 and a sequence of subjects s\_1,…s\_n+1 such that s\_i r o\_i and s\_i w o\_i+` for all I, 1 <= I <= n.

Information paths can disrupt integrity.

Three different policies depending on your situation.

**Low-Water-Mark Policy:** Whenever a subject access an object the integrity level of the subject changes to be the lowest level between the subject or object.

1. s in S can write to o in O, iff i(o) <= i(s).
2. If s in S reads o in O then i`(s) = min(i(s), i(o)) where i` is the subjects integrity level after a read.
3. s\_1 in S can execute s\_2 in S, iff i(s\_2) <= i(s\_1).

If there is an information transfer path from object o\_1 in O to object o\_n+1 then the enforcement of the low-water-mark policy requires that i(o\_n+1) <= i(o\_1) for all n > 1.

**Ring Policy:** Ignores indirect modification but focuses on direct modifications:

1. Any subject may read any object, regardless of integrity level.
2. s in S can write to o in O iff i(o) <= i(s).
3. s\_1 in S can execute s\_2 in S iff i(s\_2) <= i(s\_1).

**Biba’s Model (Strict Integrity Policy):**

1. s in S can read o in O, iff i(s) <= i(o).
2. s in S can write to o in O, iff i(o) <= i(s).
3. s\_1 in S can execute s\_2 in S, iff i(s\_2) <= i(s\_1).