Introduction to
Computer Security

Lecture 4
SPM, Security Policies,
Confidentiality and Integrity Policies

September 23, 2004

Schematic Protection Model

- Key idea is to use the notion of a protection type
  - Label that determines how control rights affect an entity
  - Take-Grant:
    - *subject* and *object* are different protection types
  - TS and TO represent subject type set and object set
  - τ(X) is the type of entity X
- A *ticket* describes a right
  - Consists of an *entity name* and a *right symbol*: **X**/*z*
    - Possessor of the ticket **X**/*z* has right *r* over entity **X**
    - Y has tickets **X**/*r*, **X**/*w* -> Y has tickets **X**/*rw*
  - Each entity **X** has a set *dom*(**X**) of tickets **Y**/*z*
  - τ(X/*r:c*) = τ(X)/*r:c* is the type of a ticket

Schematic Protection Model

- Inert right vs. Control right
  - Inert right doesn't affect protection state, e.g. *read* right
  - *take* right in Take-Grant model is a control right
- Copy flag c
  - Every right *r* has an associated copyable right *rc*
  - *r:c* means *r* or *rc*
- Manipulation of rights
  - A link predicate
    - Determines if a source and target of a transfer are "connected"
  - A filter function
    - Determines if a transfer is authorized

Transferring Rights

- $dom(\mathbf{X})$: set of tickets that X has
- Link predicate: $link_i(\mathbf{X},\mathbf{Y})$
  - conjunction or disjunction of the following terms
    - $\mathbf{X}/z \in dom(\mathbf{X})$; $\mathbf{X}/z \in dom(\mathbf{Y})$;
    - $\mathbf{Y}/z \in dom(\mathbf{X})$; $\mathbf{Y}/z \in dom(\mathbf{Y})$
    - **true**
  - Determines if **X** and **Y** "connected" to transfer right
  - Examples:
    - Take-Grant: $link(\mathbf{X}, \mathbf{Y}) = \mathbf{Y}/g \in dom(\mathbf{X}) \vee \mathbf{X}/t \in dom(\mathbf{Y})$
    - Broadcast: $link(\mathbf{X}, \mathbf{Y}) = \mathbf{X}/b \in dom(\mathbf{X})$
    - Pull: $link(\mathbf{X}, \mathbf{Y}) = \mathbf{Y}/p \in dom(\mathbf{Y})$
    - Universal: $link(\mathbf{X}, \mathbf{Y}) = \mathbf{true}$
- *Scheme*: a finite set of link predicates is called a scheme

Filter Function

- Filter function:
  - Imposes conditions on when tickets can be transferred
  - f_i: $TS \times TS \rightarrow 2^{T \times R}$ (range is copyable rights)
- **X**/*r:c* can be copied from *dom*(**Y**) to *dom*(**Z**) iff $\exists i$ s. t. the following are true:
  - **X**/*rc* $\in$ *dom*(**Y**)
  - $link_i$(**Y**, **Z**)
  - τ(X)/*r:c* $\in f_i(\tau(\mathbf{Y}), \tau(\mathbf{Z}))$
- Examples:
  - If $f_i(\tau(\mathbf{Y}), \tau(\mathbf{Z}))$ = T x R then any rights are transferable
  - If $f_i(\tau(\mathbf{Y}), \tau(\mathbf{Z}))$ = T x RI then only inert rights are transferable
  - If $f_i(\tau(\mathbf{Y}), \tau(\mathbf{Z})) = \Theta$ then no tickets are transferable
- One filter function is defined for each link predicate

SCM Example 1

- Owner-based policy
  - Subject U can authorize subject V to access an object F iff U owns F
  - Types: TS= {*user*}, TO = {*file*}
  - Ownership is viewed as copy attributes
    - If U owns F, all its tickets for F are copyable
  - RI: { *r:c*, *w:c*, *a:c*, *x:c* }; RC is empty
    - read, write, append, execute; copy on each
  - $\forall$ **U**, **V** $\in$ user, *link*(**U**, **V**) = **true**
    - Anyone can grant a right to anyone else if they posses the right to do so (copy)
  - *f*(*user*, *user*) = { *file*/*r*, *file*/*w*, *file*/a, *file*/*x* }
    - Can copy read, write, append, execute

SPM Example 1

- *Peter* owns file *Doom*; can he give *Paul* execute permission over *Doom*?
  1. τ(*Peter*) is *user and* τ(*Paul*) is *user*
  2. τ(*Doom*) is *file*
  3. *Doom*/*xc* $\in$ *dom*(*Peter*)
  4. *Link*(*Peter*, *Paul*) = TRUE
  5. τ(*Doom*)/*x* $\in$ *f*(τ(*Peter*), τ(*Paul*)) - because of 1 and 2

  Therefore, Peter can give ticket *Doom*/*xc* to Paul

SPM Example2

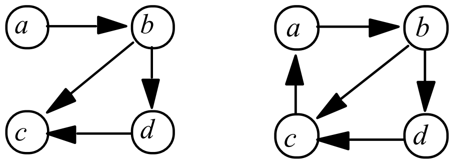
- Take-Grant Protection Model
  - *TS* = { subjects }, *TO* = { objects }
  - *RC* = {*tc*, *gc*}, *RI* = {*rc*, *wc*}
    - Note that all rights can be copied in T-G model
  - *link*(**p**, **q**) = **p**/*t* $\in$ *dom*(**q**) v **q**/*t* $\in$ *dom*(**p**)
  - *f*(*subject*, *subject*) = { *subject*, *object* } $\times$ { *tc*, *gc*, *rc*, *wc* }
    - Note that any rights can be transferred in T-G model

Demand

- A subject can demand a right from another entity
  - Demand function $d:TS \rightarrow 2^{T \times R}$
  - Let *a* and *b* be types
    - $a/r{:}c \in d(b)$: every subject of type *b* can demand a ticket $X/r{:}c$ for all *X* such that $\tau(X) = a$
  - A sophisticated construction eliminates the need for the demand operation – hence omitted

Create Operation

- Need to handle
  - type of the created entity, &
  - tickets added by the creation
- Relation $can\bullet create(a, b) \subseteq TS \times T$
  - A subject of type *a* can create an entity of type *b*
- Rule of *acyclic creates*
  - Limits the membership in $can\bullet create(a, b)$
  - If a subject of type *a* can create a subject of type *b*, then none of the descendants can create a subject of type *a*



Create operation Distinct Types

- create rule $cr(a, b)$ specifies the
  - tickets introduced when a subject of type *a* creates an entity of type *b*
- **B** object: $cr(a, b) \subseteq \{ b/r{:}c \in RI \}$
  - Only inert rights can be created
  - A gets B/r:c iff $b/r{:}c \in cr(a, b)$
- **B** subject: $cr(a, b)$ has two parts
  - $cr_P(a, b)$ added to **A**, $cr_C(a, b)$ added to **B**
  - **A** gets **B**/$r{:}c$ if $b/r{:}c$ in $cr_P(a, b)$
  - **B** gets **A**/$r{:}c$ if $a/r{:}c$ in $cr_C(a, b)$

Non-Distinct Types

- $cr(a, a)$: who gets what?
  - $self/r{:}c$ are tickets for creator
  - $a/r{:}c$ tickets for the created
- $cr(a, a) = \{ a/r{:}c, self/r{:}c \mid r{:}c \in R\}$
- $cr(a, a) = cr_C(a, b) | cr_P(a, b)$ is attenuating if:
  1. $cr_C(a, b) \subseteq cr_P(a, b)$ and
  2. $a/r{:}c \in cr_P(a, b) \Rightarrow self/r{:}c \in cr_P(a, b)$
- A scheme is attenuating if,
  - *For all types a,* $cc(a, a) \rightarrow cr(a, a)$ is attenuating

Examples

- Owner-based policy
  - Users can create files: *cc*(*user, file*) holds
  - Creator can give itself any inert rights: *cr*(*user*, *file*) = {*file*/*r*:*c*| $r \in RI$}
- Take-Grant model
  - A subject can create a subject or an object
    - *cc*(*subject*, *subject*) *and cc*(*subject*, *object*) hold
  - Subject can give itself any rights over the vertices it creates but the subject does not give the created subject any rights (although grant can be used later)
    - $cr_C(a, b) = \Theta$; $cr_P(a, b)$ = {*sub/tc, sub/gc, sub/rc, sub/wc*}

    Hence,
    - *cr*(*sub*, *sub*) = {*sub/tc, sub/gc, sub/rc, sub/wc*} | Θ
    - *cr*(*sub*, *obj*) = {*obj/tc, obj/gc, obj/rc, obj/wc*} | Θ

Safety Analysis in SPM

- Idea: derive *maximal state* where changes don't affect analysis
  - Indicates all the tickets that can be transferred from one subject to another
  - Indicates what the maximum rights of a subject is in a system
- Theorems:
  - A maximal state exists for every system
  - If parent gives child only rights parent has (conditions somewhat more complex), can easily derive maximal state
  - Safety: If the scheme is acyclic and attenuating, the safety question is decidable

Typed Access Matrix Model

- Finite set T of types ($TS \subseteq T$ for subjects)
- Protection State: (S, O, τ, A)
  - $\tau : O \rightarrow T$ is a type function
  - Operations same as in HRU model except create adds type
- τ is child type iff command create creates subject/object of type τ
- If parent/child graph from all commands acyclic, then:
  - Safety is decidable
  - Safety is NP-Hard
  - Safety is polynomial if all commands limited to three parameters

HRU *vs*. SPM

- SPM more abstract
  - Analyses focus on limits of model, not details of representation
- HRU allows revocation
  - SPM has no equivalent to delete, destroy
- HRU allows multiparent creates, SPM does not
  - SPM cannot express multiparent creates easily, and not at all if the parents are of different types because *can•create* allows for only one type of creator
  - Suggests SPM is less expressive than HRU

Comparing Models

- Expressive Power
  - HRU/Access Control Matrix subsumes Take-Grant
  - HRU subsumes Typed Access Control Matrix
  - SPM subsumes
    - Take-Grant
    - Multilevel security
    - Integrity models
- What about SPM and HRU?
  - SPM has no revocation (delete/destroy)
- HRU without delete/destroy (monotonic HRU)
  - MTAM subsumes monotonic mono-operational HRU

Extended Schematic Protection Model

- Adds “joint create”: new node has multiple parents
  - Allows more natural representation of sharing between mutually suspicious parties
    - Create joint node for sharing
- Monotonic ESPM and Monotonic HRU are equivalent

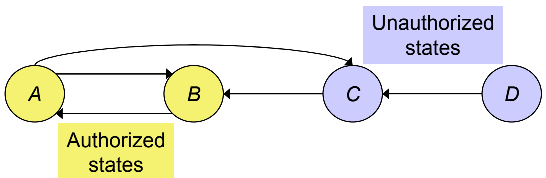
Security Policies

Overview

Security Policy

- Defines what it means for a system to be secure
- Formally: Partitions a system into
  - Set of secure (authorized) states
  - Set of non-secure (unauthorized) states
- Secure system is one that
  - Starts in authorized state
  - Cannot enter unauthorized state

Secure System - Example



- Is this Finite State Machine Secure?
  - *A* is start state ?
  - *B* is start state ?
  - *C* is start state ?
  - How can this be made secure if not?
  - *Suppose A*, *B*, and *C* are authorized states ?

Additional Definitions:

- Security breach: system enters an unauthorized state
- Let *X* be a set of entities, *I* be information.
  - *I* has **confidentiality** with respect to *X* if no member of *X* can obtain information on *I*
  - *I* has **integrity** with respect to *X* if all members of *X* trust *I*
    - Trust *I*, its conveyance and protection (data integrity)
    - *I* maybe origin information or an identity (authentication)
    - *I* is a resource – its integrity implies it functions as it should (assurance)
  - *I* has **availability** with respect to *X* if all members of *X* can access *I*
    - Time limits (quality of service

Confidentiality Policy

- Also known as *information flow*
  - Transfer of rights
  - Transfer of information without transfer of rights
  - Temporal context
- Model often depends on trust
  - Parts of system where information *could* flow
  - Trusted entity must participate to enable flow
- Highly developed in Military/Government

Integrity Policy

- Defines how information can be altered
  - Entities allowed to alter data
  - Conditions under which data can be altered
  - Limits to change of data
- Examples:
  - Purchase over $1000 requires signature
  - Check over $10,000 must be approved by one person and cashed by another
    - *Separation of duties :* for preventing fraud
- Highly developed in commercial world

Transaction-oriented Integrity

- Begin in consistent state
  - "Consistent" defined by specification
- Perform series of actions (*transaction*)
  - Actions cannot be interrupted
  - If actions complete, system in consistent state
  - If actions do not complete, system reverts to beginning (consistent) state

Trust

- Theories and mechanisms rest on some trust assumptions
- Administrator installs patch
  1. Trusts patch came from vendor, not tampered with in transit
  2. Trusts vendor tested patch thoroughly
  3. Trusts vendor's test environment corresponds to local environment
  4. Trusts patch is installed correctly

Trust in Formal Verification

- Formal verification provides a formal mathematical proof that given input *i*, program *P* produces output *o* as specified
- Suppose a security-related program *S* formally verified to work with operating system *O*
- What are the assumptions?

Trust in Formal Methods

1. Proof has no errors
   - Bugs in automated theorem provers
2. Preconditions hold in environment in which *S* is to be used
3. *S* transformed into executable *S*' whose actions follow source code
   - Compiler bugs, linker/loader/library problems
4. Hardware executes S' as intended
   - Hardware bugs

Security Mechanism

- Policy describes what is allowed
- Mechanism
  - Is an entity/procedure that enforces (part of) policy
- Example Policy: Students should not copy homework
  - Mechanism: Disallow access to files owned by other users
- Does mechanism enforce policy?

Security Model

- Security Policy: What is/isn't authorized
- Problem: Policy specification often informal
  - Implicit vs. Explicit
  - Ambiguity
- Security Model: Model that represents a particular policy (policies)
  - Model must be explicit, unambiguous
  - Abstract details for analysis
  - HRU result suggests that no single nontrivial analysis can cover all policies, but restricting the class of security policies sufficiently allows meaningful analysis

Common Mechanisms: Access Control

- Discretionary Access Control (DAC)
  - Owner determines access rights
  - Typically *identity-based access control*: Owner specifies other users who have access
- Mandatory Access Control (MAC)
  - Rules specify granting of access
  - Also called *rule-based access control*
- Originator Controlled Access Control (ORCON)
  - Originator controls access
  - *Originator need not be owner!*
- Role Based Access Control (RBAC)
  - Identity governed by role user assumes

Policy Languages

- High-level: Independent of mechanisms
  - Constraints expressed independent of enforcement mechanism
  - Constraints restrict entities, actions
  - Constraints expressed unambiguously
    - Requires a precise language, usually a mathematical, logical, or programming-like language
  - Example: Domain-Type Enforcement Language
    - Subjects partitioned into domains
    - Objects partitioned into types
    - Each domain has set of rights over each type

Example: Web Browser

- Goal: restrict actions of Java programs that are downloaded and executed under control of web browser
- Language specific to Java programs
- Expresses constraints as conditions restricting invocation of entities

Expressing Constraints

- Entities are classes, methods
  - *Class*: set of objects that an access constraint constrains
  - *Method*: set of ways an operation can be invoked
- Operations
  - *Instantiation*: *s* creates instance of class *c*: *s* ├ *c*
  - *Invocation*: *s*1 executes object *s*2: s1 |→☟ s2
- Access constraints
  - **deny**(*s op x*) **when** *b*
  - when *b* is true, subject *s* cannot perform *op* on (subject or class) *x*; empty *s* means all subjects

Sample Constraints

- Downloaded program cannot access password database file on UNIX system
- Program's class and methods for files:

```
class File {
    public file(String name);
    public String getfilename();
    public char read();
    ….
```

- Constraint:

deny(|→ file.read) **when**
    (file.getfilename() == "/etc/passwd")

Policy Languages

- Low-level: close to mechanisms
  - A set of inputs or arguments to commands that set, or check, constraints on a system
  - Example: Tripwire: Flags what has changed
    - Configuration file specifies settings to be checked
    - History file keeps old (good) example

Secure, Precise Mechanisms

- Can one devise a procedure for developing a mechanism that is both secure *and* precise?
  - Consider confidentiality policies only here
  - Integrity policies produce same result
- Program with multiple inputs and one output as an abstract function
  - Let p be a function p: $I_1 \times ... \times I_n \rightarrow R$. Then p is a program with n inputs $i_k \in I_k$, $1 \leq k \leq n$, and one output $r \rightarrow R$
  - *Goal:* determine if P can violate a security requirement (confidentiality, integrity, etc.)

Programs and Postulates

- Observability Postulate:
  - the output of a function encodes all available information about its inputs
    - Covert channels considered part of the output
  - Output may contain things not normally thought of as part of function result
- Example: authentication function
  - Inputs name, password; output Good or Bad
  - If name invalid, print Bad; else access database
  - Problem: time output of Bad, can determine if name valid
  - This means timing is part of output

Protection Mechanism

- Let p be a function $p: I_1 \times \ldots \times I_n \to R$. A protection mechanism m is a function $m: I_1 \times \ldots \times I_n \to R \cup E$ for which, when $i_k \in I_k$, $1 \le k \le n$, either
  - $m(i_1, \ldots, i_n) = p(i_1, \ldots, i_n)$ or
  - $m(i_1, \ldots, i_n) \in E$.
- E is set of error outputs
  - In above example, E = { "Password Database Missing", "Password Database Locked" }

Confidentiality Policy

- Confidentiality policy for program p says which inputs can be revealed
  - Formally, for $p: I_1 \times \ldots \times I_n \to R$, it is a function $c: I_1 \times \ldots \times I_n \to A$, where $A \subseteq I_1 \times \ldots \times I_n$
  - A is set of inputs available to observer
- Security mechanism is function $m: I_1 \times \ldots \times I_n \to R \cup E$
  - m *secure* iff $\exists\, m': A \to R \cup E$ such that, for all $i_k \in I_k$, $1 \le k \le n$, $m(i_1, \ldots, i_n) = m'(c(i_1, \ldots, i_n))$
  - m returns values consistent with c

Examples

- $c(i_1, \ldots, i_n) = C$, a constant
  - Deny observer any information (output does not vary with inputs)
- $c(i_1, \ldots, i_n) = (i_1, \ldots, i_n)$, and $m' = m$
  - Allow observer full access to information
- $c(i_1, \ldots, i_n) = i_1$
  - Allow observer information about first input but no information about other inputs.

Precision

- Security policy may be over-restrictive
  - Precision measures how over-restrictive
- m_1, m_2 distinct protection mechanisms for program p under policy c
  - m_1 as precise as m_2 ($m_1 \approx m_2$) if, for all inputs $i_1, \ldots, i_n$
    - $m_2(i_1, \ldots, i_n) = p(i_1, \ldots, i_n) \Rightarrow$
    - $m_1(i_1, \ldots, i_n) = p(i_1, \ldots, i_n$
  - m_1 more precise than m_2 ($m_1 \sim m_2$) if there is an input $(i_1', \ldots, i_n')$ such that
    - $m_1(i_1', \ldots, i_n') = p(i_1', \ldots, i_n')$ and
    - $m_2(i_1', \ldots, i_n') \neq p(i_1', \ldots, i_n')$.

Combining Mechanisms

- m_1, m_2 protection mechanisms
- $m_3 = m_1 \cup m_2$ defined as
  - $p(i_1, \ldots, i_n)$ when $m_1(i_1, \ldots, i_n) = p(i_1, \ldots, i_n)$ or $m_2(i_1, \ldots, i_n) = p(i_1, \ldots, i_n)$
  - else $m_1(i_1, \ldots, i_n)$
- Theorem: if m_1, m_2 secure, then m_3 secure
  - $m_1 \cup m_2$ secure
  - $m_1 \cup m_2 \approx m_1$ and $m_1 \cup m_2 \approx m_2$
  - Proof follows from the definitions

Modeling Secure/Precise: Confidentiality – existence theorem

- Theorem: Given p and c, $\exists$ a precise, secure mechanism m^* such that $\forall$ secure m for p and c, $m^* \approx m$
  - Proof: Induction from previous theorem
  - Maximally precise mechanism
  - Ensures security
  - Minimizes number of denials of legitimate actions
- There is no effective procedure that determines a maximally precise, secure mechanism for any policy and program.

Confidentiality Policies

Confidentiality Policy

- Also known as information flow policy
  - Integrity is secondary objective
  - Eg. Military mission "date"
- Bell-LaPadula Model
  - Formally models military requirements
    - Information has sensitivity levels or classification
    - Subjects have clearance
    - Subjects with clearance are allowed access
  - Multi-level access control or mandatory access control

Bell-LaPadula: Basics

- Mandatory access control
  - Entities are assigned security levels
  - Subject has security clearance $L(s) = l_s$
  - Object has security classification $L(o) = l_o$
  - Simplest case: Security levels are arranged in a linear order $l_i < l_{i+1}$
- Example

  Top secret > Secret > Confidential >Unclassified

"No Read Up"

- Information is allowed to flow *up*, not *down*
- *Simple security property:*
  - *s* can read *o* if and only if
    - $l_o \leq l_s$ and
    - *s* has read access to *o*
  - Combines mandatory *(security levels)* and discretionary *(permission required)*
  - Prevents subjects from reading objects at higher levels (*No Read Up rule*)

“No Write Down”

- Information is allowed to flow *up*, not *down*
- **property*
  - *s* can write *o* if and only if
    - $l_s \le l_o$ and
    - *s* has write access to *o*
  - Combines mandatory *(security levels)* and discretionary *(permission required)*
  - Prevents subjects from writing to objects at lower levels (*No Write Down rule*)

Example

security level	*subject*	*object*
Top Secret	Tamara	Personnel Files
Secret	Samuel	E-Mail Files
Confidential	Claire	Activity Logs
Unclassified	Ulaley	Telephone Lists

- Tamara can *read* which objects? And *write*?
- Claire cannot read which objects? And *write*?
- Ulaley can *read* which objects? And *write*?

Access Rules

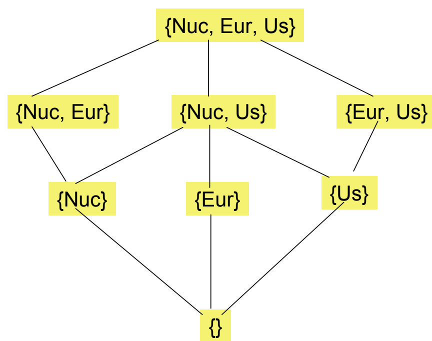
- Secure system:
  - One in which both the properties hold
- Theorem: Let Σ be a system with secure initial state σ_0, T be a set of state transformations
  - If every element of T follows rules, every state σ_i secure
  - Proof - induction

Categories

- Total order of classifications not flexible enough
  - Alice cleared for missiles; Bob cleared for warheads; Both cleared for targets
- Solution: Categories
  - Use set of compartments (from power set of compartments)
  - Enforce "*need to know*" principle
  - Security levels (security level, category set)
    - (Top Secret, {Nuc, Eur, Asi})
    - (Top Secret, {Nuc, Asi})
- Combining with clearance:
  - (L,C) *dominates* $(L',C') \Leftrightarrow L' \leq L$ and $C' \subseteq C$
  - Induces lattice of security levels

Lattice of categories

- Examples of levels
  - (Top Secret, {Nuc,Asi}) *dom* (Secret, {Nuc})
  - (Secret, {Nuc, Eur}) *dom* (Confidential, {Nuc,Eur})
  - (Top Secret, {Nuc}) ¬*dom* (Confidential, {Eur})
- Bounds
  - Greatest lower,
  - Lowest upper
  - *glb* of {X, Nuc, Us} & {X, Eur, Us}?
  - *lub* of {X, Nuc, Us} & {X, Eur, Us}?



Access Rules

- *Simple Security Condition*: *S* can read *O* if and only if
  - *S dominate O* and
  - *S* has read access to *O*
- **-Property*: *S* can write *O* if and only if
  - *O dom S* and
  - *S* has write access to *O*
- Secure system: One with above properties
- Theorem: Let Σ be a system with secure initial state σ_0, *T* be a set of state transformations
  - If every element of *T* follows rules, every state σ_i secure

Problem: No write-down

Cleared subject can't communicate to non-cleared subject

- Any write from l_i to l_k, $i > k$, would violate *-property
  - Subject at l_i can only write to l_i and above
- Any read from l_k to l_i, $i > k$, would violate simple security property
  - Subject at l_k can only read from l_k and below
- Subject at level *i* can't write something readable by subject at *k*
  - Not very practical

Principle of Tranquility

- Should we change classification levels?
- Raising object's security level
  - Information once available to some subjects is no longer available
  - Usually assumes information has already been accessed
  - Simple security property violated? Problem?
- Lowering object's security level
  - Simple security property violated?
  - The *declassification problem*
  - Essentially, a "write down" violating *-property
  - Solution: define set of trusted subjects that *sanitize* or remove sensitive information before security level is lowered

Types of Tranquility

- Strong Tranquility
  - The clearances of subjects, and the classifications of objects, do not change during the lifetime of the system
- Weak Tranquility
  - The clearances of subjects, and the classifications of objects, do not change in a way that violates the simple security condition or the *-property during the lifetime of the system

Example

- DG/UX System
  - Only a trusted user (security administrator) can lower object's security level
  - In general, process MAC labels cannot change
    - If a user wants a new MAC label, needs to initiate new process
    - Cumbersome, so user can be designated as able to change process MAC label within a specified range