Trusted?
- An operating system is *trusted* if we have confidence it provides
  - Memory protection,
  - File protection.
  - General object access control,
  - User authentication
    in a *consistent and effective* way.
- How to design such an OS?

What Makes System Trusted
- Policy
  - Security requirements, well defined, consistent, unambiguous, implementable.
- Model
  - Representation of the policy, formal. Should not degrade functionality.
- Design
  - Includes functionality, implementation options.
- Trust
  - Review of features, assurance makes an OS worthy of trust.
Terminology

<table>
<thead>
<tr>
<th>Secure</th>
<th>Trusted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure: Either-or</td>
<td>Graded: degrees of trustworthiness</td>
</tr>
<tr>
<td>Property of presenter</td>
<td>Property of receiver</td>
</tr>
<tr>
<td>Asserted based on product characteristics</td>
<td>Judged based on evidence and analysis</td>
</tr>
<tr>
<td>Absolute: not qualified by usage</td>
<td>Relative: viewed in the context of use</td>
</tr>
<tr>
<td>A goal</td>
<td>A characteristic</td>
</tr>
</tbody>
</table>

Trust in What?
- Process
  - Absent of security flaws and malicious segments
- Product
  - Evaluated, approved product.
- Software
  - Part trusted to enforce security policy
- Computing base
  - Hw/SW/firmware enforcing unified policy.
- System
  - Trusted to process sensitive information

Security Policy
- The statement we expect the system to enforce.
- Military security policy: Need-to-know.
Military Security Policy

- Each object has its *classification*.
  - `<rank; compartment>`.
- *Clearance*: Trust to access information up to a certain level of sensitivity.
- For subject `s` and object `o`:
  
  \[
  s \leq o \iff \text{rank}_s \leq \text{rank}_o \text{ and } \text{compartments}_s \leq \text{compartments}_o
  \]

- `O` dominates `S`, `S` is dominated by `O`

Military Security Policy (2)

- `S` can access `O` (`S` dominates `O`) only if:
  - Clearance of `S` at least as high as `O` (hierarchy).
  - `S` needs to know about ALL compartments where `O` is classified (nonhierarchical).
- `S:<top_secret, Sweden>, O:<secret, Sweden>`
  - `S` dominates `O`.
- `S:<top_secret, Sweden>, O:<secret, {Sweden, crypto}>`
  - `S` does not dominate `O`.
- `S:<top_secret, ,{Sweden, crypto}>, O:<secret, Sweden>`
  - `S` dominates `O`

Commercial Security Policies

- Data sensitivity: *public, proprietary, internal...*
  - Names vary, but not the meaning.
- Projects similar to compartments.
  - Corporate lever responsibilities range over project/department boundaries.
- But, outside the government, no notion of *clearance*!
  - Rules regulating access are not formalized.
- Integrity and availability as important as confidentiality, but not as well formulated.
Clark-Wilson Security Policy

- Defines well-formed transactions.
  - Do not sign a receipt unless the appropriate goods received.
  - Do not pay without having a signed receipt.
- Such constrained data items processed by transformation procedures.
  - The only way to access these items, validates the order.
- Transaction: <userID, TP_i, {CDI_j, CDI_k, ...}>
- Separation of duties.
  - Different person orders, receives, pays.
  - Introduce state information into the triplet notation.

Chinese Wall Security Policy

- Information access protection.
- Objects divided in conflict classes.
  - Access possible to only one object in each class.

Security Models

- Models formalize policies.
- Useful for:
  - Conceptualization and design.
  - Completeness and consistency checking.
  - Documentation of policy.
  - Verification and validation.
Multilevel Security

- Need for multiple layers of sensitivity.
- Mathematical formalizations of military-type security policies
  - Lattice model.
  - Relation forms partial ordering if
    - Transitive: If \( a \leq b \) and \( b \leq c \) then \( a \leq c \)
    - Antisymmetric: If \( a \leq b \) and \( b \leq a \) then \( a = b \)
  - Lattice elements may not be comparable, but they have an upper bound and a lower bound.

Lattice Model of Access Security

- Example: “Factor of” lattice.
- Helpful in modeling hierarchical relationships.
- Dominance relation in the military security is a lattice.
- Upper bound: \(<\text{top secret, all compartments}>\)
- Lower: \(<\text{unclassified, no compartments}>\)
- Commercial security may form a lattice
  - Public, proprietary, internal

Bell-LaPadua Confidentiality Model

- Describes allowable access paths.
  - Central to US DoD evaluation criteria.
- Goal: concurrent computations at different sensitivity levels.
  - No information leaks allowed.
- \( C(s) \) and \( C(o) \): sensitivity levels.
- Simple Security Property:
  - \( S \) may have a read access to \( O \) iff \( C(o) \leq C(s) \)
Bell-LaPadua (2)
- * - property
  S that has a read access to O, may have write access to P iff \( C(o) \leq C(p) \)
  - Contents of sensitive documents cannot be written below their level of sensitivity.
  - Prevent write-down.
  - How stringent is this?
    - Should a classified husband talk with his wife?
    - Consideration of noninterference needed.

Biba Integrity Model
- Counterpart of Bell-LaPadua
- Subject S can modify (write) object o only if \( I(s) \geq I(o) \)
- If subject s has read access to object o with integrity level \( I(o) \), s can have write access to object p only if \( I(o) \geq I(p) \).

Trusted OS Design
- Security cannot be “added” successfully.
  - It needs to be designed into an OS.
  - Modularity, information hiding principles.
  - Software engineering principles essential:
    - Requirements, design and test traceability, formal and informal reviews.
    - Integrated and centralized design treatment of security featurees.
Good design principles
- Least privilege.
- Economy of mechanisms
  - Small, simple.
- Open design.
- Complete mediation
  - Check every access attempt.
- Permission based
  - Denial of access is the default.
- Separation of privilege
  - Multiple barriers (authentication & encryption).
- Least common mechanism (share less).
- Ease of use.

Features of Trusted OS
- User identification and authentication
- Mandatory access controls
- Discretionary access controls
- Object reuse protection.
- Complete mediation.
- Trusted paths.
- Audits.
- Audit log reduction.
- Intrusion detection.

Identification and authentication
- Key to computer security
- Two steps
  - Who is the service requester.
  - Verify the claimed identity.
- I&A in trusted systems goes beyond that of traditional OS.
Access Control (AC)

- Mandatory AC implies that a central authority makes access decisions.
  - Owner not involved.
  - Military policy.
- Discretionary AC makes owner or someone else in charge of the decision.
  - Typical for commercial environments.
  - Access rights may change dynamically.
- If applied simultaneously, MAC has a precedence over DAC.

Object reuse policy

- If an object is “freed” by the user, its allocated space remains “dirty”.
  - Previous user’s data may become available to someone.
  - New user may scavenge for sensitive data.
  - This approach is called **object reuse**.
- OS needs to prevent leakage, by overwriting all reassigned space.

Complete mediation & Trusted paths

- All accesses to all resources need control.
- But, modern OS have numerous paths towards an object.
  - Network ports, processes, DMA, etc.
- Trusted paths needed to establish unmistakable communication trust.
  - Unique keys (CTRL-ALT-DEL), for example.
  - Cannot be intercepted by software.
Audits

- Security information relevant log files are necessary.
  - These should accommodate audit requests.
- How to avoid enormously large files?
  - Limit to open/close object actions. Even these can become too large over time.
- Audit: Find a needle in a haystack!
- Continuous audits performed in the background.

Intrusion detection

- ID software establishes patterns of "normal usage".
  - Consequently, it can sound alarm if these patterns change.
- Detailed study later.

Kernelized design principles

- Kernel: part of OS that performs lowest level functions.
  - Synchronization, interprocess communication, message passing, interrupt handling.
- Security kernel responsible for enforcing security mechanisms.
  - Security kernel contained within OS kernel.
Why security kernel?

- **Coverage**
  - Every access MUST pass through.
- **Separation**
  - From the rest of OS and from the users.
- **Unity**
  - Concentration of security relevant code.
- **Modifiability**
- **Compactness**
  - Likely to be relatively small
- **Verifiability**

Reference Monitors

- Control access to all objects.
  - Collection of access control code for devices, files, memory, communication...
- RM must be
  - Tamperproof
  - Invoked by any access request
  - Small enough to be verifiable
- Part of a wider suite of tools and techniques

Trusted Computing Base (TCB)

- Code responsible for implementing security policy
- Trust in an OS depends on trust in TCB!
- Security enforcement depends on:
  - Hardware (processors, memory, registers, I/O)
  - Processes (protect security critical ones)
  - Files (special files as I&A data, access control DB)
  - Protected memory (protects reference monitor against tampering)
  - Interprocess communication (so parts of TCP can pass data and activate other parts)
TCB (2)

- TCB monitors four basic interactions:
  - Process activation
    - Includes memory allocation, context switching, file access lists, status information
  - Execution domain switching
    - Request for sensitive data may cause a process to switch execution domains.
  - Memory protection
    - Monitor all memory accesses, maintain confidentiality, integrity.
  - I/O operation

Virtualization

- Virtual machine is a collection of resources (HW, SW), real or simulated.
- Each user given the perception of unique ownership.
  - Multiple virtual memory spaces, divided by users or user groups or types.