Trusted Computing Systems

To create a TCS, it is necessary to consider how to:

1. *Achieve system, physical and communications security*
2. *Evaluate the degree of assurance achieved*

Achieving System Security

- Development of secure systems require:
  - Models: *Capture security requirements of the system, identify entities (subjects and objects) and define relationship between them*
  - Policies: *Statement of the specific security that the system should enforce (instantiate the model)*
  - Mechanisms: *Set of functions that interprets and enforces the security policy*
- Much of the early work on access control was carried out in the context of military computing systems

Structure of Military Security

Classification of Information

- Information is classified according to national security interests
  - Top Secret
  - Secret
  - Confidential
  - Restricted
  - Unclassified

- Additional compartments within each class is used to enforce the *need-to-know* principle:
  <rank,compartment> describes the classification and the set of people who may need to access the information

Structure of Military Security

Information Storage

- Information is stored in documents (on paper)
- Documents are stored in an opaque binder
  - Content cannot be read from the outside
- The *<rank; compartment>* is clearly marked (labeled) on the binder
- All classified information is stored in a safe, which is guarded by armed personnel
  - Further physical security measures may be in place to prevent unauthorized people from entering the area where information is stored or processed

Structure of Military Security

Accessing Classified Information

- All users are "cleared" to see information up to a certain level
  - labels and clearances have the same domain
- Users have to prove their clearance to the guard, before they can withdraw a binder from the safe
- Labels are clearly visible, so it is difficult to sneak around with classified information

Structure of Military Security

Creating New Information

- New files are labeled with the classification of the most secret component
  - unclassified + secret = secret
  - confidential + secret + top secret = top secret
- Aggregation of unclassified information may generate a "top secret" file
  - Normally not modelled by the security model
- Sanitization downgrades the label of existing information
  - Battle plans for Waterloo are now only of historical interest

Military security policy

Formal Definition

- The original example of a *confidentiality policy*!
- Based on a ranked set of *sensitivity levels*, $\mathcal{R}$, and a set of projects, $\mathcal{C}$ ("*compartments*").
  - Classification for *object* o = $<\mathcal{R}_o; \mathcal{C}_o>$, where $\mathcal{C}_o \subseteq \mathcal{C}$
  - Clearance for *subject* s = $<\mathcal{R}_s; \mathcal{C}_s>$, where $\mathcal{C}_s \subseteq \mathcal{C}$
  - Dominance ("s dominates o"):
    $$o \preceq s \Leftrightarrow (\mathcal{R}_o \leq \mathcal{R}_s) \wedge (\mathcal{C}_o \subseteq \mathcal{C}_s)$$
  - A *security label*, $\mathcal{L}_i$, is often used to identify a $<\mathcal{R}_i; \mathcal{C}_i>$ pair, and $o \preceq s$ is often written $\mathcal{L}_o \preceq \mathcal{L}_s$
- A *subject* s may read an *object* o iff $o \preceq s$, i.e.:
  - s's clearance level is *at least as high* as o's classification level.
  - s needs to know about *all* compartments for which information in o is classified.

Commercial confidentiality policy

- Intended to prevent leaks of commercially sensitive information while allowing ordinary business practices to continue
- Ranked list of *sensitivity levels*, e.g. public, proprietary, internal
- Set of *areas of responsibility*: projects, departments, corporate functions (accounts, personnel,…).
- No concept of *classification* or *dominance*: A manager may allow a subject access to, say, internal information from any project or department

Commercial integrity policies

- Intended to ensure *correct treatment* of information from beginning to end of business process.
- *Correct treatment* involves:
  - Performing exactly the required steps
  - Using only reliable data (from reliable sources)
  - Performing the steps in the required order
  - Authenticating the *subjects* who perform the steps
- *Well-formed transactions* (Clark-Wilson):
  In each step, a given *user* performs a given *transformation procedure* on a set of *constrained data items*, described by *access triple*:
  $$<\text{uid}, \text{TP}_i, \{\text{CDI}_j, \text{CDI}_k, \ldots\}>$$
  Policy is expressed in terms of possible sequences of access triples
- *Separation of duty* (Nash-Poland):
  Same *subject* must not be involved (alone) in several steps - this restricts the possible sequences of access triples

Chinese Wall security policy

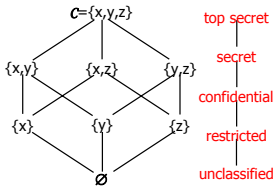
- Attacks the problem of *conflict of interest*.
- Three abstraction levels:
  1. *Objects*: Files etc. Each object belongs to one company
  2. *Company groups*: Each is a set of *objects* belonging to a single company
  3. *Conflict classes*: Each is a set of *companies* which are competitors
- A *subject* s can access any *object* o iff s has never accessed an *object* from another *company* in same *conflict class*

Basic read access models

- *Single-level systems*: Handle *subjects* and *objects* with a single security classification.
  An *object* is either sensitive or not, and a *subject* either has authorised access or not.
- *Compartmented systems*: There is a set $\mathcal{C}$ of compartments for information. An *object* belongs to a subset of compartments, $\mathcal{C}_o \subseteq \mathcal{C}$.
  A *subject* must be cleared for (at least) all the compartments of each *object* which it wants to access: $\mathcal{C}_o \subseteq \mathcal{C}_s$
- *Multi-level systems*: There is an ordered set of ranks, $\mathcal{R}$, and each *subject* and *object* have a rank.
  - Simple security property (*no read up*):
    s has read access to o iff $\mathcal{R}_o \leq \mathcal{R}_s$
  - *-property (*no write down*):
    If s has read access to o, it has write access to p iff $\mathcal{R}_o \leq \mathcal{R}_p$

Lattices as access models

- A *lattice* is a structure $\mathcal{L}$=<$\mathcal{S}$, $\preceq$> with a set of values $\mathcal{S}$ and partial ordering $\preceq$, such that any pair of elements $a,b \in \mathcal{S}$ has an:
  - *Upper bound*, u, such that: $a \preceq u \wedge b \preceq u$
  - *Lower bound*, l, such that: $l \preceq a \wedge l \preceq b$
- This covers:
  - *Compartmented systems*, where $\mathcal{L}$=<$\mathcal{C}$, $\subseteq$>
  - *Multi-level systems*, where $\mathcal{L}$=<$\mathcal{R}$, $\leq$>
  - *Military systems*, where $\mathcal{L}$=<$\mathcal{C}\times\mathcal{R}$, $\preceq$>
  - … and many others.



Access and Integrity models

- *Access models*: rules for R/W access to maintain confidentiality
- *Bell-La Padula access model* for security classes $\mathcal{A}$ ordered by $\preceq$:
  - *Simple security property* (*no read up*):
    s has read access to *o* iff $\mathcal{A}_o \preceq \mathcal{A}_s$
  - **-property* (*no write down*):
    If *s* has read access to *o*, it has write access to *p* iff $\mathcal{A}_o \preceq \mathcal{A}_p$

  Matches general lattice model of access security

- *Integrity models*: rules for R/W access to avoid incorrect changes to data by untrustworthy persons:
- *Biba integrity model* for integrity levels $\mathcal{I}$ ordered by $\preceq$:
  - *Simple integrity property* (*no untrusted write*):
    s has write access to *o* iff $\mathcal{I}_o \preceq \mathcal{I}_s$
  - *Integrity *-property* (*don't propagate untrusted info*):
    If *s* has read access to *o*, it has write access to *p* iff $\mathcal{I}_p \preceq \mathcal{I}_o$

N.B.!

Formal system models

- Intended to enable us to answer questions like:
  "Can *subject s* ever obtain write access to *object o*?"
- Can be used to decide limits to what a protection system can be proved to offer
- Typically based on description of system in terms of:
  - Set of *subjects*, $\mathcal{S}$
  - Set of *objects*, $\mathcal{O}$
  - Set of *rights*, $\mathcal{R}$
  - *Access matrix*, $\mathcal{A}$
  - Set of *primitive operations*, $\mathcal{P}$
- Examples:
  - Graham-Denning, 1972
  - Harrison-Ruzzo-Ullman, 1976 ("HRU")
  - Lipton 1977, Snyder 1981 ("Take-Grant")

HRU model

Basic Elements

- Set of *subjects* $\mathcal{S}$; set of *objects* $\mathcal{O}$ (which *includes* $\mathcal{S}$)
- Set of abstract *primitive operations*, $\mathcal{P}$:
  - Create *subject s*
  - Create *object o*
  - Destroy *subject s*
  - Destroy *object o*
  - Add *right r* to $\mathcal{A}[s,o]$
  - Delete *right r* from $\mathcal{A}[s,o]$

HRU model

Operational Model

- Each system command consists of a set of conditions $\mathcal{C}$ and a sequence of operations $[p_1;p_2;\ldots;p_n]$, where $p_i \in \mathcal{P}$:

  command cnam($o_1, o_2, \ldots, o_k$):
  if $(r_1 \in \mathcal{A}[s_1, o_1]) \wedge (r_2 \in \mathcal{A}[s_2, o_2]) \wedge \ldots \wedge (r_m \in \mathcal{A}[s_m, o_m])$
  then $[p_1; p_2; \ldots; p_n]$

- Shown adequate to model and analyse several examples of real-life protection systems
- But results for whether a given system can confer given rights are only *decidable* for single-operation commands!

HRU Commands

Examples

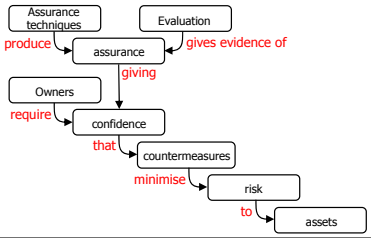
- **Creation of a file:**
  command CREATE_FILE(*s*,*o*)
  create object *o*
  add *own* to $\mathcal{A}[s,o]$
  end CREATE_FILE
- **Transfer a privilege, e.g. read access:**
  command CONFER_READ(s_1,*o*,s_2)
  if *own* $\in \mathcal{A}[s_1, o]$
  then add *read* to $\mathcal{A}[s_2, o]$
  end CONFER_READ
- **Revoke a privilege, e.g. write access:**
  command REVOKE_WRITE(s_1,*o*,s_2)
  if (*own* $\in \mathcal{A}[s_1, o]$) $\wedge$ (write $\in \mathcal{A}[s_2, o]$)
  then delete *write* from $\mathcal{A}[s_2, o]$
  end REVOKE_WRITE

Pause

Back in 15 minutes

TCS Evaluation

- Two key aspects:
  - *Functionality*: Functions in system which enforce security
  - *Assurance*: Effectiveness and correctness of design and impl.



TCSEC "Orange Book"

A by now somewhat old-fashioned (1983/85) approach to system security

- Systems are divided into evaluation classes, each characterised by a set of *functionality requirements* and an *assurance level*:

Class	Description
D	Minimal protection
C1	Discretionary Security Protection
C2	Controlled Access Protection
B1	Labelled Security Protection
B2	Structured Protection
B3	Security Domains
A1	Verified Design

- Security *functionality* and required level of *assurance* increase from D (no requirements) to A1 (high requirements, formal verification)

TCSEC Functionality Requirements

		D	C1	C2	B1	B2	B3	A1
Security policy	Discretionary Access Control		⊕	⊕	=	=	⊕	=
	Object reuse			⊕	=	=	=	=
	Labelled output				⊕	=	=	=
	Mandatory Access Control				⊕	⊕	=	=
	Device labels					⊕	=	=
Accountability	Identification/authentication		⊕	⊕	⊕	=	=	=
	Audit (logging)			⊕	⊕	⊕	⊕	=
	Trusted path					⊕	⊕	=
Assurance	Assured system architecture		⊕	⊕	⊕	⊕	⊕	=
	Assured system integrity		⊕	=	=	=	=	=
	Covert channel analysis					⊕	⊕	⊕
	Trusted recovery						⊕	=

⊕ New/enhanced in this class = Same as in previous (lower) class

Problems with TCSEC

- TCSEC represents best practice *anno 1980*
- TCSEC focuses on *confidentiality* issues, as in the military security model
- TCSEC is based on *US DoD* security concepts:
  - Security-cleared personnel
  - Authority and management control
  - Secure physical environments
- TCSEC is focused on *monolithic, stand-alone, multi-user* OSs:
  - Very little attention paid to networks
  - No attention to modern system design (Client/server, Agents, P2P, Grid, …)

Common Criteria

- A more modern (1999) approach to system security
- The aim of IT Security is defined as:
  "Reduction of risks associated with threats to the information arising directly or indirectly from human error or deliberate subversion."
- Key concepts:
  - *Protection Profile (PP)* : Set of generic security requirements for some particular class of (sub-)system
  - *Security Target (ST)* : A particular instance of a PP
  - *Target of Evaluation (TOE)* : The actual system evaluated
- Basic steps in security analysis:
  - *Threat Analysis* : to discover conceivable threats
  - *Risk Analysis* : to determine suitable countermeasures

CC Functionality Classes

- In CC, *functionality* requirements are *separate* from *evaluation* classes and *assurance* levels.
- Functionality classes are:
  - Identification and authentication (FIA): Verification of user identity.
  - Trusted path (FTP): Establishment of trusted user-TSB path.
  - Security audit (FAU): Recognition, recording, analysis of info. related to security activities.
  - TOE access (FTA): Establishment of user's session.
  - User data protection (FDP): Access control, information flow control.
  - Resource utilisation and availability (FRU): Assurance of availability.
  - Protection of TOE security functions (FPT): Protection of internal, security-related functions.
  - Privacy (FPR): Protection against discovery of user identity.
  - Communication (FCO): Proof of origin and non-repudiation.
  - Cryptographic support (FCS): Key management, cryptographic operation.

CC Functional Requirements

- For each class, specific requirement families are defined
- E.g. class *FIA, Identification and authentication*:
  - *Authentication failures (FIA_AFL)* : Action in event of (multiple) authentication failures
  - *User attribute definition (FIA_ATD)* : Security attributes apart from user's identity
  - *Specification of secrets (FIA_SOS)* : Generation and verification of secrets according to some quality metric
  - User authentication (FIA_UAU) : Types of user authentication mechanisms and the timing of their use
  - *User identification (FIA_UID)* : When do users have to identify themselves
  - *User-subject binding (FIA_USB)* : How to maintain binding between user and a subject ("proxy") working for the user
- Requirements in a family may be specified as amenable to:
  - *Management* (see class FMT)
  - *Auditing* (see class FAU)

CC Assurance Levels

- *Assurance levels* state level of checking which has been documented:
  - AL0: Unassured
  - AL1: Tested
  - AL2: Structurally tested
  - AL3: Methodically tested and checked
  - AL4: Methodically tested and reviewed
  - AL5: Semiformal design
  - AL6: Semiformal verified design
  - AL7: Formally verified design.

Trusted Systems in Practice

- Historically, many systems analysed according to *TCSEC Orange Book* criteria
- More recently, design and evaluation according to *ITSEC* or *CC* has become the norm
- For *CC*, a number of *Protection Profiles* for various areas of operation are available and should be used if they apply.
- *Evaluation* can be expensive – especially to high assurance levels:
  - *TCSEC*: B1 and above
  - *CC*: AL4 and above