Contribution

- Glossary of Security Terms
- Categories of Protection Systems
- Design Principles for Specifying Secure Systems
- Examples of Applying Design Principles
Categories of Protection Systems

- Unprotected Systems
  - No security

- All-or-nothing Systems
  - Effectively a private machine for each user
  - May have equally shared libraries

- Controlled Sharing
  - Simple ACL determines user access to a resource

- User-programmed Sharing Controls
  - Detailed control over user access to resource
  - May use a protected subsystem
Design Principles

- Economy of Mechanism
  - Keep it SIMPLE!
- Fail Safe Defaults
  - Default condition is lack of access
- Complete Mediation
  - “Every access to every object must be checked…”
Design Principles (cont’d)

- **Open Design**
  - Strength of system not dependent on secrecy of the system’s design
  - Only keys, passwords, etc. should be secret

- **Separation of Privilege**
  - Single accident does not comprise the system

- **Least Privilege**
  - All users/programs operate with the set of least privileges
Design Principles (cont’d)

- Least Common Mechanism
  - Minimize resources common to multiple user and depended on by all users
  - Shared resources are potential information paths between users

- Psychological Acceptability
  - Human interface must be easy to use
Design Principles (cont’d)

- Work Factor
  - System should be immune from “guessing”
  - e.g. good passwords
- Compromise Recording
  - Record security related events (auditing)
Related Work

- Design Principals Can Guide Development
  - Basic specification for nearly any modern secure system
    - Authentication systems, operating systems, encryption systems, etc.

- Various OS implementations
  - DOS for IBM System 370, Dartmouth Time-Sharing System, DEC DECsystem/10, Multics, etc.
  - Virtual Machines, modern operating systems
Roadmap

- Need to develop a **model** for protection.
- First Model: Define TOTALLY ISOLATED partitions.
- Next: *Extend* this model to allow SHARING OF INFORMATION.
Some Analogies

- All info is within partitions – MUTUALLY EXCLUSIVE.
- Objects within partitions: fundamental units of info to be protected.
- High walls around partitions (isolation)
- Strong door with a guard.
- (Later) – How to lower these walls systematically (For info sharing).
- Why do you need a guard??? Controlled Use.
- => You need an id… and guard must be able to verify it (aka *authentication*).
Isolated VMs… A concrete example

- Computer: Multiplexed among several users… (aka multiprogramming)
- Several Virtual Processors, each unaware of the other.
- How to create this illusion? *Descriptor Register*
- Controls access to memory areas
Isolated VMs

- Who loads descriptor? **Supervisor**
- When? **Privileged bit set.**
- Who can set it? **Supervisor only.**
- How does user program load reg? **Trap to supervisor.**
  Supervisor changes privilege mode and loads descriptor.
- Supervisor maintains table of descriptor-virtual processor mappings, in its own memory area.
- The Supervisor is also an isolated virtual processor.
Isolated VMs

- Info being protected: distinct programs. Guard: extra piece of hardware; does enforcement.
- Strong door: Memory accesses are forced through descriptor checks.
- Supervisor: Protected subsystem
- ECONOMY OF MECHANISM: Supervisor is common to all mechanisms. Also, it uses the same mechanism to protect itself from other apps.
Authentication

- What if the Virtual machine has to be accessed remotely?
- How to trust the terminal making the request?
- Which Virtual machine to associate with a terminal?
- Some form of **authentication** is required.
Authentication Mechanisms

- Different from that of implementing VMs on a single machine.
- Objects being protected: VMs themselves.
- Two ways: **Secrecy** or **Unforgeability**
  - **Secrecy**(what you know): password based. Name-password matching.
    - Usually, system uses a one-way function to transform password, then uses it.
  - **Unforgeability**: Give user something unforgeable and use it as his identification to authenticate. E.g., biometrics.
  - **Weaknesses**: Vulnerable to man-in-the-middle / eavesdropping.
- A slightly better scheme is encryption using shared secret keys. But what about key distribution?
Shared Information Protection

- Apps may use same code libraries / modify same database. VMs not appropriate, because they provide Total isolation.

- Generally, the mechanisms that protect shared info are of the two categories:
  - List-oriented Mechanisms
  - Ticket-oriented Mechanisms
Shared Information Protection

- **List-oriented:**
  - Guard has a list of ids of authorised users.
  - User presents id. Guard does a match.
  - Authorisation (Granting access): Done by adding id of user to list.

- **Ticket-oriented:**
  - Guard holds just one identifier.
  - Anybody who presents the identifier can get access. So, the user will maintain his set of ids he needs to access various objects.
  - Authorisation: Done by giving user ticket for object.
Shared Information Protection

- **List-based**
  - Guard needs to search for every object access.
  - Bad for high-traffic app
  - Guard needs to know who is making access
  - User can be identified easily
  - Revocation: easy.
  - Fail-safe default: No access if id not in list.

- **Ticket-based**
  - User needs to search for every object access.
  - Better for high-traffic app
  - Guard has no clue who is making access.
  - Revocation hard. Why?
  - Confinement problem...
  - Fail-safe default: Have to present ticket for access. Otherwise, access denied.
Principals and Domains

- What else needs to be protected?
  - Guard’s authorisation info
  - Mapping between user and unforgeable label / ticket(s).
- Need to map user to VM for accountability. Hence an abstraction for acc. – \textit{principal}. Set of objects that principal can access: \textit{Domain}
- Also need to distinguish between users and principals they may assume. (aka roles)
- So, the sequence of steps would be:
  - User authenticates himself, to establish right to use a principal.
  - All further actions labelled with id of principal (accountability)
- Principal may be used to implement privilege separation. How?
  - Create a principal that \textit{has} to be assumed by several users.
Info-sharing VMs

- Say a math routine is to be shared.
- Add another descriptor reg for math routine.
- Why not store descriptor in memory instead of reg?
- As before, only supervisor can load reg.
- Should prog A be able to write to math routine? No. How do we enforce it?
- Which reg. to use for checks against addresses now?
Implications of sharing

- Can P1 overwrite math routine?
- Can math routine make modifications to itself?
- How about multiple shared routines?
- How does supervisor know which principals can access shared routines?
  - Access control and related problems pop up – again!
Conclusions

- The Problem: Protect Information.
- Basic design principles were laid down; some abstractions developed on the way..
- Two cases:
  - Totally isolated apps
  - Apps that share code / data.
- Hardware regs used to enforce isolation, along with supervisor and controlled entry points.
- Minimal mechanism used in common between all apps. Same mechanism used to protect subsystem also.
- Principals, domains etc used to implement the accountability, separation of privilege.
Take-away

- Seminal paper: **531 direct citations** (similar one: “the end-to-end argument” by saltzer, reed & clark)
- Work-factor: You can **never correctly predict it** and make arrangements. So you’d better be prepared!
- Security should be **integral part of system design**: it should be designed into the system, right from the hardware onward to the o.s. Retrofitting is a big headache.
- Total isolation is **unrealistic**; sharing is **necessary** – but it complicates the security problem – and also increases cost. But we have to live with it.
Take-away

- Strong authentication and authorisation mechanisms underlie all the other protection mechanisms. If they are bad, they bring the whole structure crashing down.
- The supervisor should be trustworthy. But who watches the watchers?
- Also Note: This paper discusses Mechanism, not Policy.
- Further: Not all security principles are implemented in the mechanisms outlined so far in this paper.
- So………. Security is hard!