Security
An Advanced Introduction

Overview

* Security concepts
  * Security policies
  * Security mechanisms
  * Trusted computing
  * Design principles

What is Security?

* Example 1: DOS
  * Single-user system with no access control
    * Is it secure?
      * ... if I have no data?
      * ... if I contain the payroll database?
      * ... if it is on a machine in the Dover
      * ... if it is in a locked room?
      * ... if it is behind a firewall?

What is Security?

* Example 2: Banking store's weekly earnings:
  * Is it secure to
    * ... ask random customer to do it?
    * ... ask many branch customers to do it?
    * ... ask several staff members to do it?
    * ... have a security firm?
    * ... have several security firms?
    * Depends? On what?

Secure System

* Requires a security policy
  * specifies allowed and disallowed states of a system
    * system needs to ensure that no disallowed state is ever entered
    * need OS mechanisms to prevent transitions from allowed to disallowed states
  * Security policy needs to identify the assets to be secured
    * for computer security, assets are typically data
    * Perfect security is generally unachievable
      * need to be aware of choices
      * need to understand what tradeoffs can be tolerated

Data Security

Three aspects:

* Confidentiality: prevent theft of data
  * concealing data from unauthorized agents
  * need-to-know principle
* Integrity: prevent damage of data
  * trustworthiness of data: data correctness
  * trustworthiness of means of data authentication
* Availability: prevent denial of service
  * ensuring data is usable when needed
**Threats**

- A **weakness** is a potential for a security violation
- An **attack** is an attempt by an attacker to violate security
  - generally implies exploiting a weakness
- A **threat** is a potential for an attack
- There is never a shortage of attackers, hence in practice:
  - threat = attack
  - weakness $\rightarrow$ violation

**Security Policy**

- Partitioned system into allowed and disallowed states
- Ideally mathematical model
  - In practice, natural-language description
  - often imprecise, ambiguous, inconsistent, unenforceable
  - Example: transactions over $50k require manager approval
  - but transferring $10k into own account is no violation

**Security Mechanisms**

- **Authentication**
  - computer access control (login authentication)
  - OS file access control system
  - controls implemented in tools
- Example: only accountant can access financial system
- Mechanism: on-networked computer in locked room with only one key
- A secure system provides mechanisms that ensure that violations are:
  - prevented
  - detected
  - recovered from

**Assumptions**

- Security is always based on assumptions
  - e.g., lock is secure, key holders are trustworthy
- Invalid assumptions void security!
- Problem: assumptions are often implicit and poorly understood
- Security assumptions must be:
  - clearly identified
  - evaluated for validity

**Potentially Invalid Assumptions**

- The security policy is unambiguous and consistent
- The mechanisms used to implement the policy are correctly designed
- The union of mechanisms implements the policy correctly
- The mechanisms are correctly implemented
- The mechanisms are correctly installed and administered
**Trust**

- Systems always have trusted entities
- hardware, operating system, system
- TB is trusted entity, the trusted computing base (TCB)
- part of the system that can determine security
- Assume to be trustworthy
- is it?

**Trusted Computing Base**

TCB: The highest protection mechanism within a computer system — including hardware, firmware, and software — the combination of which is responsible for enforcing a security policy.

(RFC 2636)

A TCB consists of one or more components that together enforce a unified security policy over a product or system.

The ability of the TCB to correctly enforce a security policy depends solely on the mechanisms within the TCB and on the correct inputs by system administrative personnel or parameters related to the security policy. (G009)

**Trusted Computing**

- TCB is by definition trusted. That doesn’t make it trustworthy!
- Aim of trusted computing is to establish and maintain trustworthiness
- and with respect to certain security requirements
- TC ensures that system is operating in defined configuration
- based on the assumption that certain components can be trusted
- Challenge: maintain system security during configuration changes
- Idea: based on notion of secure booting (AF/SDD):
  - Root of trust provided by hardware
  - Software components are certified as trusted
  - TCB securely expanded by loading trusted components only
  - Hardware- and software mechanisms to prevent tampering
  - Established plans of logs from root of trust

**Covert Channels**

- Information flow that is not controlled by a security mechanism
- security requires absence of covert channels
- Two types of covert channels:
  - covert timing channel uses an attribute of a shared resource
    - typically read data, like existence or accessibility of an object
    - globally visible covert storage channels
    - in principle subject to access control
  - a sound access control system should be free of covert channels
  - covert timing channel uses temporal order of accesses to shared resource
  - outside access-control system
  - difficult to reason about
  - difficult to prevent

**Covert Timing Channels**

- Created via shared resource whose behaviour can be monitored
  - network bandwidth
  - CPU load
  - response time
  - time
- Requires access to a time source
  - real-time clock
  - anything else that allows unrelated processes to synchronize
  - preventable by perfect virtualization?
- Critical issue is bandwidth
  - in practice the damage is limited if the bandwidth is low
  - e.g., CRIME doesn’t care about low-bandwidth channels
  - beware of amplification
  - e.g., leaking of passwords

**Assurance**

- Process for substantiating (substantiating or specifying) trust
  - Specifications
    - unambiguous description of system behaviour
    - can be formal or informal
  - Design
    - justification that it meets specification
    - mathematical translation of specification or compelling argument
    - implementation
    - justifications that it is consistent with the design
    - mathematical proof of rigorous testing
    - by implication must also satisfy specification
    - operations and maintenance
    - justification that system is used as per assumptions in specification
    - Assurance does not guarantee correctness or security
Assurance: Orange Book

US Department of Defence "Orange Book" [ODD66]

- Defined security classes:
  - D: minimal protection
  - C1-C7: discretionary access control
  - B1-B3: mandatory access control
- A1: certified design
- Designed for military use
- Systems can be certified to a certain class
  - any country, hence only available for big companies
  - most systems only certified C2 (assembly line-style security)

Assurance: Common Criteria

Common Criteria [NNSS99]

- ISO standard, developed out of Orange Book and other approaches
  - US, Canada, UK, Germany, France, Netherlands
  - for general use (not just military)
- Unlike Orange Book, doesn't prescribe specific security requirements
- Evaluates quality assurance used to ensure requirements are met
- Target of Evaluation (TOE) evaluated against security target (ST)
  - ST a statement of desired security properties
  - based on protection profiles (PPs) — generic sets of requirements
  - defined by "owners" (typically governments)
- Security assurance assessment levels (SALs)
  - Higher levels imply more thorough evaluation (and higher cost)
  - not necessarily better security
- Details later

Summary

- Computer security is complex
  - depends on many aspects of computer system
- Policy defines security, mechanisms enforce security
- Important to consider:
  - what are the assumptions about threats and trustworthiness?
  - incorrect assumptions
- Security is never absolute
  - given enough resource, mechanisms can be defeated
  - important to understand limitations
  - shared thresholds between security and usability
- Human factors are important
  - people make mistakes
  - people may not understand security impact of actions
  - people may be less trustworthy than thought

Overview

- Security concepts
- Security policies
- Security mechanisms
- Trusted computing
- Design principles

Security Policies: Categories

- Discretionary (user-controlled) policies
  - e.g. a user can read file objects only with file permission
  - user decides about access (at their discretion)
  - classical example: Unix permissions
- Mandatory (system-controlled) policies
  - e.g. certain users cannot ever access certain objects
  - no user can change these
  - focus on restricting information flow
  - Role-based policies
    - agents can take on specific pre-defined roles
    - well-defined set of roles for each agent
    - e.g. normal user, systemadmin, database admin
    - access rights depend on role

Security Policy Models

- Represent a whole class of security policies
- Most system-wide policies focus on confidentiality
  - e.g. military-style multi-level security models
    - classical example: Bell-LaPadula model [BL73]
    - most others developed from this
    - Orange Book-based on this model
- Other models
  - Chinese wall policy focuses on conflict of interest
  - Clark-Wilson model focuses on separation of duty
Bell-LaPadula Model

- Each object has a security classification $L_o$.
- Each agent has a security clearance $C_a$.
- Classifications and clearances form hierarchical security levels-
  - e.g. top secret > secret > confident > unclassified
- Rule 1 (no read up)
  - $a$ can read only if $L_a \leq L_o$.
  - standard confidentiality
- Rule 2 (no write down)
  - $a$ can write only if $L_a \leq L_o$.
  - prevents leaking (accidental or by conspiracy)
  - logging
  - command chain
  - need way to declassify data

Bell-LaPadula Extensions

- Can combine with discretionary access rights
  - need explicit permission on specific objects
  - e.g. SELinux
- Can add orthogonal security categories indicating types of data
  - restrict access to relevant categories
  - Dirmac’s similar reader [See Ref]

Chinese Wall Policy

- Employed by investment banks to manage conflicts of interest
- Idea: Consultant cannot talk to clients’ competitors
- Single consultant can have multiple concurrent clients
- Define conflict classes (groups of potentially competing clients)
  - e.g., banks, oil companies, insurance companies, OS vendors
  - Consultant dealing with client of class $A$ cannot talk to others in $A$ but can continue talking to members of other classes
  - some data belongs to several conflict classes
- Public information is not restricted
  - consultant can read and write public info at any time
  - but must observe policy (cannot publish confidential info)
- Example of a dynamic MAC policy
  - allowed information flow changes over time

Common Criteria Protection Profiles for OS

- Controlled Access Protection Profile (CAPP)
  - standard OS security, derived from Orange Book C2
  - certified up to level EAL3
- Single-level Operating System Protection Profile
  - super of CAPP
  - certified up to EAL4+
- Labelled Security Protection Profile (LSPP)
  - mandatory access control for C2/C3/C4/C5
  - specified to Orange Book B1
- Role-based Access Control Protection Profile
  - Multi-level Operating System Protection Profile
  - validated [CAPP, LSPP]
  - separation kernel protection profile (SKPP)
  - strict partitions
  - specifications as ISO 15408

Common Criteria Assurance Levels

- EAL1: functionally tested
  - simple to do, can be done without help from developer
- EAL2: structurally tested
  - functional and interface specs, basic and functional testing
- EAL3: functionally checked, tested
  - improved test coverage, procedures to avoid tempering during development
  - highest assurance level achieved for Mac OS 10
- EAL4: methodically designed, tested and reviewed
  - design docs used for testing, avoid tempering during delivery
  - independent vulnerability analysis
  - highest level feasible on existing product (not developed for C4 level)
  - achieved by major vendors:
    - Solaris: EAL4+ in 2005
    - Solaris 10 EAL4+ in 2005
    - comprehensive access protection profile (CAPP)
    - RedHat EAL4++ in 2007
  - still may get locked:
    - certification based on assumptions about environment, etc.
    - must use in outside those assumptions
    - certification means nothing in such a case
    - presumably there were no compromise assumptions held
Common Criteria Assurance Levels

- **EAL5**: formally designed and tested
  - formal model of TEO security policy
  - semi-formal model of functional specs & high-level design
  - architecture & security requirements
  - system-level analysis
  - defense-in-depth analysis

- **EAL6**: semi-formally verified design and tested
  - semi-formal low-level design
  - structure representation of implementation
  - modular and layered TOE design
  - independent vulnerability analysis
  - systematic covert channel identification
  - re-use of components, including security technology underlying EAL6+ certification

**Note:**
- **Even EAL7 relies on testing**
  - EAL7 requires proof of correspondence between formal descriptions
  - however, no requirement for formal proof of implementation completeness

Overview

- Security concepts
- Security policies
- **Security mechanisms**
- Trusted computing
- Design principles

Security Mechanisms

- Used to implement security policies
  - Based on access control
    - discretionary access control (DAC)
    - mandatory access control (MAC)
    - role-based access control (RBAC)

- Access rights
  - simple rights
    - read, write, execute/modify, send, receive
  - more rights (DAC only)
    - copy
    - propagate own rights to another agent
    - more
  - change rights of an object or agent

Access Control Matrix

<table>
<thead>
<tr>
<th>Agent</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>define</td>
<td>read</td>
<td>write</td>
<td>read</td>
</tr>
<tr>
<td>S1</td>
<td>read</td>
<td>execute</td>
<td>modify</td>
<td>execute</td>
</tr>
<tr>
<td>Others</td>
<td>execute</td>
<td>modify</td>
<td>read</td>
<td>write</td>
</tr>
</tbody>
</table>

- Defines each agent’s rights on any object
- Note: agents are objects too

Properties of Access Control Matrix

- Rows define agents’ protection domains (PDs)
- Columns define objects’ accessibility
- Dynamic data structures
  - frequent permanent changes (e.g. databases)
  - frequent temporary changes (e.g. logs)
- Very sparse with many repeated entries
- Impractical to store explicitly
Issues for Protection System Design (DAC)

- Propagation of rights:
  - Can agent grant right to other?
- Restriction of rights:
  - Can agent grant access to others?
- Revocation of rights:
  - Can access, once granted, be revoked?
- Amplification of rights:
  - Can unprivileged agent perform restricted operations?
- Determination of object accessibility:
  - Which agents have access to particular objects?
- Is object accessible in all (or some) environments?
- Determination of agent’s protection domain:
  - Which objects are accessible?

Access Matrix Implementation: ACLs

- Represent column-wise: access roles (ACLs):
  - ACL associated with object
    - Propagation: meta-right (e.g., owner can modify)
    - Restriction: meta-right
    - Revocation: meta-right
    - Amplification: protected-inheritance right (e.g., setuid)
  - Protection domain: hard (if not impossible) to determine
  - Usually commanded via domain classes (UNIX, NT groups)
  - Full ACLs used by Multics, Apollo Domain, Andrew FS, NTFS
  - Can have negative rights to:
    - reduce window of vulnerability
    - simplify management of groups of capabilities
  - Implemented in almost all commercial systems

Access Matrix Implementation: Capabilities

- Represent row-wise: capabilities:
  - Capability list associated with agent
    - Each capability confers a certain right to its holder
  - Propagation: capability between agents (how?)
  - Restriction: least-restriction of all (derived) caps
  - Revocation: require revocation of caps from all agents
  - Amplification: special invocation capability
    - Propagation: capability list from one agent to another (how?)
    - Protection domain: explicit in capability list
  - Can have negative rights to:
    - reduce window of vulnerability
    - simplify management of groups of capabilities
  - Only successful commercial systems: IBM System/38 etc.

Capabilities

- Main advantage of capabilities is the fine-grained access control
  - easy to provide access to specific agents
  - Capability preserves prime face evidence of the right to access
    - capability => object identifier (policy naming)
    - capability => set of access rights
    - Capability representation must contain object ID and access rights
  - Propagation must protect capability from forgery
  - How are caps implemented and protected?
    - tagged — protected by hardware
    - protected by software
    - implementation — protected by policy (terminologically secure, like encryption)

Tagged Capabilities

- Tag bits) with every (group of) memory word(s)
  - tag identifies capabilities
  - capability is stored with the data
  - hardware checks permissions when determining capability
  - for security reasons, certain instructions (e.g., write) can turn tag on
  - propagation easy
  - propagation requires kernel to make new capability
  - revocation requires kernel to turn off capability
  - protection domain difficult to establish
  - IBM System/38, AS/400, +Series, many historical systems

Protected Procedure Call (AS/400)

- AS/400 has segmented memory architecture:
  - that’s why the PowerPC has segments
- Capabilities confer rights over segments
- Capabilities can confer invocation rights
- Each user has a profile, which is essentially a capability list
- Capabilities can be of "profile propagation" type
  - on invocation, segment owner’s profile is added to caller’s PD
  - when user has set of profiles, and the profile contains caps
  - on return, profile adoption is cancelled
  - user can delete a subset of the profile to be used in adoption
  - called profile propagation

CAPABILITIES FROM: xref Security
Tagged Capabilities Outside RAM

- Disk has no tags
- 4K/IO realizes tags by restricting physical I/O to low-level OS
  - 4K/IO page size is 4KB
  - physical disk block size is 512B
  - extra 64B per page used to store tag box (among others)
  - on-page-in, pages must be scanned and all tags collected
  - on-page-out, all tags are reconstructed
  - significant processing overhead on all I/O

Tagged Capabilities Summary

- Secure through hardware protection
- Convenient for applications (appears as normal pointers)
- Checked by hardware => fast validation
- Hardware solution is not for everyone
- Capability hardware is complex (hence above)
- Separate mechanisms required for I/O and distribution

Partitioned Capabilities

- System maintains capability list (ClRL) with each process
  - user code uses indirect references to caps (dist index)
  - c.f. Unix file descriptors
  - System validates access via dist when mapping any page
  - validation is explicit at map time
  - propagation: system call to copy between clients
  - reservation: kernel to make new capability
  - revocation: kernel to remove cap from dist
    - one specific or all
  - accessibility: requires scanning all clients
  - protection domain: explicitly represented in dist
- Hydra (C/JT), Mach [RTY+91], KeyKOS [BF9+92], ERCS [GSP98], and many others

Partitioned Capabilities Summary

- Secure through protection by kernel
  - real caps live in kernel space
- Validation at mapping time => apps use "normal" pointers
- Fast validation (dist check is simple, validation cached by MMU)
- Propagation requires marshaling and kernel intervention
- Reference counting possible to detect unaccessible objects

Propagating Partitioned Capabilities (Mach)

- Capabilities can be included in IPC messages
  1) user inserts dist index into caps field in message
  2) kernel looks up dist and inserts representation of cap (marshaling)
  3) receiver's kernel inserts caps into receiver's dist
  4) kernel replaces global cap reference by local dist index
- Simplified if IPC is local
- Amplification can be performance by schemes similar to AS/400

Sparse Capabilities

- Basic idea similar to encryption
  - add/chang string to make valid capabilities a very small subset of cap space
  - either encrypted object info [GLT91] or password MT96, APW89
  - secure by infeasibility of exhaustive search of cap space

---

The diagram shows the architecture and security mechanisms of partitioned capabilities in Mach, including how capabilities are propagated and how sparse capabilities are implemented through encryption-like methods to secure access to a subset of the capabilities.
Sparse Capabilities

- Sparse caps are user-level objects
  - can be passed like other data
  - similar to tagged caps, but without hardware support
  - validated at registration time (explicit or implicit)
- good match to user-level servers
- no control authority – no kernel required on most caps
- cannot reference-count objects
- Issues:
  - Full mediation requires extra work
  - Not stable, see Microsoft [REV880]
  - Limited system-wide cross-domain isolation
  - High overhead of creating data
  - problem with covert channels

Confinement

- Problem 1: Executing untrusted code
  - you downloaded a game from the internet?
  - how can you be sure it doesn’t steal/alter your data?
- Problem 2: Digital rights management (DRM)
  - you own copyrighted material (e.g., entertainment media content)
  - you want to let others use it (for a fee)
  - how can you prevent them from making unauthorized copies?

- You need to confine the program (game, viewer) so it cannot leak
- Cannot be done with most protection schemes!
  - not with non-trust or non-trusted code-based schemes
  - not with most tagged or sparse-capability schemes
  - multi-level security has some inherent confinement (but can’t do DRM)
- Some protection models can confine in principle
  - a segregated cap system, can restrict system call to accept only
  - EJB5 has formal proof of confinement of a model of the system [GWV]
- In practice difficult to achieve due to covert channels

Overview

- Security concepts
- Security policies
- Security mechanisms
- Trusted computing
- Design principles

Trusted Computing

- Trusted Computing Group (TCG)
  - industry consortium with many members
  - defines industry standards to enable trusted computing
  - form trusted computing now virtually synonymous with TCG model
- TCG Trusted Computing Module (TCM)
  - hardware root of trust, aimed at PC/Linux platforms
  - minimal functionality to support TC
  - implemented either as separate chip or onboard processor chip
- Similarly Mobile Trusted Module (MTM) for mobile devices
- Also TCG Software Stack (TSS) for higher-level functionality

TPM-Enabled Functionality

- Authenticated booting
  - supply OS in a well-defined configuration
  - execute only certified binaries
- Remote attestation
  - allow remote party to confirm system configuration
- Sealed storage
  - ensures that data can only be read if system is in particular configuration

Enabled by a set of TPM-provided mechanisms:
- Random-number generation
  - Key generation
  - key storage
  - public-key encryption
  - configuration storage
  - certificate storage

TPM Components

- Hardware implementations of security-relevant low-level functions
  - random numbers, SHA-1 hash, public-key generation, RSA encryption
  - slow and meant for use before enough trusted software is booted
- Endorsement key (EK)
  - hardware-backed key, uniquely identifies physical device
  - public EK certified and supplied by manufacturer
- Non-volatile storage
  - small amount of EK, some symmetric keys, op-in flags
  - storage root key (SRK), protected by SKR pass phrase
  - to encrypt keys stored outside TPM
**Integrity Measurement**

- Idea: Measure all components and securely store measurements.
- Measurement: SHA-1 hash of component.
  - Computed at component load time, before execution.
  - Normally computed by software (outside TPM) as TPM SHA-1 is slow.
- Secure storage of measurements:
  - Store log of measurements outside TPM.
  - Inside TPM's PCR store combined by SHA engine (measurement): PCR = SHA(PCR | SHA-1 component).
- Suffix to verify configuration:
  - Compute condensed measurement from log and compare to PCR.
  - Does not guarantee that software hasn't been modified after loading.
- SHA-1 engine is point of trust for measurement (RTM).

**Remote Attestation (aka Integrity Reporting)**

- Idea: Provide certified representation of machine state to challenger.
  - E.g., service provider who wants to verify configuration.
- Two parts reported:
  - Measurement log key (PCR).
  - PCR value (accumulated measurement) signed by endorsement key.
- Alternatively can set up specific attestation identify key (AIK).
- Challenger can verify:
  - Re-compute PCR value.
  - Verify signature using:
    - Knowledge of endorsement key, or
    - Previously exchanged AIK.
- Endorsement key is root of trust for reporting (RTR).

**Secure Storage Channel: Sealing**

- Idea: Make certain data accessible only to correct machine state.
  - Pass data securely from "render" to "receiver" configuration.
  - Time-travel IPC.
- Uses secure encryption:
  - Generate secret key (random number).
  - Use this to encrypt data with trusted (authenticated) program.
  - Encrypt secret key using SKK, can then be stored anywhere.
- Sealing:
  - RSA engine can optionally include PCR configuration in encryption.
  - When encrypting key, include:
    - Present ("render") PCR state.
    - Desired ("receiver") PCR state.
  - Only decrypt if PCR state matches "receiver" state.
  - Return "render" PCR state with decrypted key for confirmation.
- Storage root key is root of trust for storage (RTS).

**Authenticated Boot**

- TPM ROM contains:
  - Boot block.
  - Public key of OS manufacturer.
- OS components signed by manufacturer key(s):
  - Only load components after verifying signature.
  - Measure components prior to executing.
- Boot block loads first OS component:
  - Using TPM cryptography hardware.
- First OS components contain:
  - SM implementation of crypto.
  - Potential further software vendor keys.

**Secure Boot**

- Seal (rather than just sign) OS components:
  - Makes it impossible to boot other than predetermined OS version.
- Rather painful:
  - Complete OS must be sealed separately for individual target machine.
  - Any software upgrade requires re-sealing.
  - Quite impractical for normal OS.
  - But could be feasible for hypervisor or microkernel.
- Based on secure bootwork by Arbaugh et al [AFSW].

**Overview**

- Security concepts.
- Security policies.
- Security mechanisms.
- Trusted computing.
- Design principles.
### Design Principles for Secure Systems

- Least privilege (PLOA)
- Fail-safe defaults
- Economy of mechanisms
- Complete mediation
- Open design
- Separation of privilege
- Least common mechanisms
- Psychological acceptability

### Least Privilege

- Agent should only be given the minimal rights needed for task
  - minimal protection domain
  - PID determined by familiarity, not identity
  - Unix root is evil
  - Use of role-based access control (RBAC)
  - Rights added as needed, removed when no longer needed

### Fail-Safe Defaults

- Default action is no-access
  - If action fails, system remains secure
  - If security administrator forgets to add rule, system remains secure
  - “Better safe than sorry”

### Economy of Mechanisms

- KISS principle of engineering
  - “Keep it simple, stupid!”
- Least code/feature/infrastructure to get wrong
  - Easier to secure if something goes wrong
  - Complexity is the natural enemy of security
- Also applies to interfaces, interactions, protocols, …
- Also implies minimal TCB!

### Complete Mediation

- Check every access
  - validate in box (file access)
    - access rights checked at open(), then cached
    - access remains enabled until close(), even if attributes change
    - also implies that any rights propagation must be controlled
    - not true with tagged or symmetric capability systems
  - In practice conflicts with performance
    - caching of buffers, file descriptors etc.
    - otherwise unacceptable performance in distributed systems
- Should at least limit window of opportunity
  - e.g. guarantee caches are flushed after some fixed period

### Open Design

- Security must not depend on secrecy of design or implementation
  - TCB must be open to scrutiny
  - Security by obscurity is point security
  - e.g. U.S. government’s Clipper initiative (SD)
  - FCC still doesn’t seem to understand this
- Note that this doesn’t rule out passwords or secret keys
  - but their creation requires careful oversight
Separation of Privilege

- Require a combination of conditions for granting access
  - e.g. user is in group wheel and knows the root password
  - closely related to least privilege

Least Common Mechanisms

- Avoid sharing mechanisms
  - shared mechanisms = shared channel
  - potential covert channel
- Inclined conflict with other design imperatives
  - simplicity = shared mechanisms

Psychological Acceptability

- Security mechanisms should not add to difficulty of use
  - hides complexity introduced by security mechanisms
    - ensure ease of installation, configurations, use
    - systems are used by humans!
- Inherently problematic:
  - security inherently inhibit ease of use
  - idea is to minimize impact
- Security-usability tradeoff is to a degree unavoidable