Security
An Advanced Introduction

COMP9242
2007/S2 Week 6
UNSW

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 it has no data?
    - ... if it contains the payroll database?
    - ... if it is on a machine in the foyer
    - ... 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 a random customer to do it?
    - ... ask many ransom customers to do it?
    - ... ask a staff member to do it?
    - ... ask several staff members to do it?
    - ... hire a security firm?
    - ... hire 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 tranistions from allowed to disallowed states
- Security policy needs to indentify the *assets* to be secured
  - for computer security, assets are typically data
- Perfect security is generally unachievable
  - need to be waware of *threats*
  - need to understand what *risks* can be tolerated

Data Security

Three aspects:

- **Confidentiality**: prevent *theft* of data
  - concealing data from unauthorised agents
  - need-to-know principle
- **Integrity**: prevent *damage* of data
  - trustworthiness of data: data *correctness*
  - trustworthiness of origin 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 ═▶ violation

Threats

- Snooping
  - disclosure of data
  - attack on *confidentiality*
- Modification/alteration
  - unauthorised change of data
  - attack on *data integrity*
- Masquerading/spoofing
  - one entity impersonating another
  - attack on *authentication integrity*
  - delegation?
- Repudiation of origin
  - false denial of being source
  - attack on *integrity*
- Denial of receipt
  - false denial of receiving
  - attack on *availability* and *integrity*
- Delay
  - temporarily inhibiting service
  - attack on *availability*
- Denial of service
  - permanently inhibiting service
  - attack on *availability*

Security Policy

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

Security Mechanisms

- Used to enforce security policy
  - computer access control (login authentication)
  - OS file access control system
  - controls implemented in tools
- Example:
  - Policy: only accountant can access financial system
  - Mechanism: on un-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
  - eg. 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 corectly installed and administered

Trust

- Systems always have *trusted entities*
  - hardware, operating system, sysadmin
- Totally of trusted entities is the *trusted computing base* (TCB)
  - the part of the system that can circumvent security
- Assumed to be *trustworthy*
  - is it???

Trusted Computing Base

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

[RFC 2828]

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. [Gol99]

Trusted Computing

- TCB is by definition *trusted*. That doesn't make it *trustworthy*!
- Aim of *trusted computing* (TC): establish and maintain trustworthiness
  - ... 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* [AFS97]:
  - *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 *chain of trust* from root of trust

Covert Channels

- Information flow that is not controlled by a security mechanism
  - security requires *absence of covert channels*
- Two type of covert channels
  - covert *storage* channel uses an attribute of a shared resource
    - typically meta data, like exisstence or accessibility of an object
    - global names create 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
  - locks
- Requires access to a time source
  - real-time clock
  - anything else that allows unrelated processes to synchronise
  - preventable by perfect virtualisation?
- Critical issue is bandwidth
  - in practice the damage is limited if the bandwidth is low
    - e.g. DRM doesn't care about low-bandwidth channels
  - beware of amplification
    - e.g. leaking of passwords

Assurance

- Process for *bolstering* (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
    - justification that it is consistent with the design
    - mathematical proof or rigorous testing
    - by implication must also satisfy specification
  - Operation and maintenance
    - justification that system is used as per assumptionss in specification
- Assurance does not *guarantee* correctness or security!

Assurance: Orange Book

US Department of Defence "*Orange Book*" [DoD86]:

- Defines security classes
  - D: minimal protection
  - C1–2: discretionary access control
  - B1–B3: mandatory access control
  - A1: verified design
- Designed for military use
- Systems can be certified to a certain class
  - very costly, hence only available for big companies
  - most systems only certified C2 (essentially Unix-style security)

Assurance: Common Criteria

Common Criteria [NIS99]:

- 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 is statement of desired security properties
  - based on *protection profiles* (PPs) — generic sets of requirements
    - defined by “users” (typically governments)
- Seven *evaluation assurance levels* (EALs)
  - 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 ═► no security
- Security is never absolute
  - given enough resources, mechanisms can be defeated
  - important to understand limitations
  - inherent tradeoffs 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 can read B's objects only with A's 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
    - eg normal user, sysadmin, 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 is *Bell-LaPadula* model [BL76]
  - most others developed from this
  - Orange Book based on this model
- Other models
  - *Chinese-wall* policy focusses on conflict of interest
  - *Clark-Wilson* model focusses on separation of duty

Bell-LaPadula Model

- Each object a has a security *classification* $L(a)$
- Each agent o has a security *clearance* $L(o)$
- Classifications and clearances form hierachical *security levels*
  - e.g. top secret > secret > confidential > unclassified
- Rule 1 (*no read up*):
  - a can *read* o only if $L(a) \geq L(o)$
  - standard confidentiality
- Rule 2 (*✶ Property — no write down*)
  - a can *write* o only if $L(a) \leq L(o)$
  - prevents *leakage* (accidental of by conspiracy)
  - problems:
    - logging
    - command chain
  - need way to *de-classify* data

Bell-LaPadula Extensions

- Can combine with discretionary access rights
  - read/write permissions on specific objects
  - e.g. SELinux
- Can add orthogonal security categories indicating types of data
  - restrict access to relevant categories
  - Denning's *lattice model* [Den76]

Chinese Wall Policy

- Employed by investment banks to manage conflict of interest
- Idea: Consultant cannot talk to clients' competitors
  - single consultant can have multiple concurrent clients
- Define *conflict classes* (groups of potentially competing clients)
  - eg 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 $\star$ property (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
  - superset of CAPP
  - certified up to EAL4+
- Labeled Security Protection Profile (LSPP)
  - mandatory access control for COTS OSes
  - similar to Orange Book B1
- Role-based Access Control Protection Profile
- Multi-level Operating System Protection Profile
  - superset of CAPP, LSPP
  - certified up to EAL4+
- Separation Kernel Protection Profile (SKPP)
  - strict partitioning
  - certifications aiming for EAL6+

Common Criteria Assurance Levels

- EAL1: functionally tested
  - simple to do, can be done without help from developer
- EAL2: structurally tested
  - functional and interface spec
  - black- and white-box testing
  - vulnerability analysis
- EAL3: methodically tested and checked
  - improved test coverage
  - procedures to avoid tampering during development
  - highest assurance level achieved for Mac OS X

Common Criteria Assurance Levels

- EAL4: methodically designed, tested and reviewed
  - design docs used for testing, avoid tampering during delivery
  - independent vulnerability analysis
  - highest level feasible on existing product (not developed for CC cert)
  - achieved by main-stream Oses
    - Windows 2000: EAL4 in 2003
    - SuSe Enterprise Linux: EAL4 in 2005
    - Solaris-10 EAL4+ in 2006
      - controlled access protection profile (CAPP)
      - role-based access control PP
    - RedHat EAL4+ in 2007
  - still they get broken!
    - certification is based on assumptions about environment, etc...
    - most use is outside those assumptions
      - certification means nothing in such a case
      - presumably there were no compromises were assumptions held

Common Criteria Assurance Levels

- EAL5: semiformally designed and tested
  - formal model of TEO security policy
  - semi-formal model of functional spec & high-level design
  - semi-formal argument about correspondence
  - covert-channel analysis
  - IBM z-Series hypervisor EAL5 in 2003 (partitioning)
  - attempted by Mandrake for Linux with French Government support
- EAL6: semiformally verified design and tested
  - semiformal low-level design
  - structured representation of implementation
  - modular and layered TOW design
  - independent vulnerability analysis
  - systematic covert-channel identification
  - Green Hills Integrity microkernel presently undergoing EAL6+ certification
    - separation kernel protection profile

Common Criteria Assurance Levels

- EAL7: formally verified design and tested
  - formal functional spec and high-level design
  - formal and semiformal demonstration of correspondence
    - between specification and low-level design
  - simple TOE
  - complete independent confirmation of developer tests
  - LynuxWorks claims LynxSecure separation kernel EAL7 “certifiable”
    - but not *certified*
  - Green Hills also aiming for EAL7

Note:

- *Even EAL7 relies on testing!*
  - EAL7 requires proof of correspondence between formal descriptions
  - However, no requirement of formalising implementation
  - Hence no requirement for formal proof of implementation correctness

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/invoke, send, receive
  - meta rights (DAC only)
    - copy
      - propagate own rights to another agent
    - own
      - change rights of an object or agent

Access Control Matrix

	Objects			
Agents	S_1	S_2	O_3	O_4
S_1	terminate	wait, signal, send	read	
S_2	wait, signal, terminate			read, execute, write
S_3		wait, signal, receive		
S_4	control		execute	write

- 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 structure:
  - frequent permanent changes (e.g. `chmod`)
  - frequent temporary changes (e.g. `setuid`)
- Very *sparse* with many repeated entries
- Impractical to store explicitly

Issues for Protection System Design (DAC)

- Propagation of rights:
  - Can agent grant access to other?
- Restriction of rights:
  - Can agent propagate subset of own rights?
- 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 object?
  - Is object accessible at all (garbage collection)?
- Determination of agent's protection domain
  - Which objects are accessible?

Access Matrix Implementation: ACLs

Represent column-wise: *access control list* (ALC):

- *ACL* associated with *object*
  - Propagation: meta right (e.g. owner can `chmod`)
  - Restriction: meta right
  - Revocation: meta right
  - Amplification: protected-invocation right (e.g. `setuid`)
  - Accessibility: explicit in ACL
  - Protection domain: hard (if not impossible) to determine
- Usually condensed 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 exclusion from groups
- Sometimes implicit (Unix process hierarchy)
- 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: copy capabilities between agents (how?)
  - Restriction: lesser rights require creation of new (derived) caps
  - Revocation: requires invalidation of caps from all agents
  - Amplification: special invocation capability
  - Accessibility: requires inspection of all capability lists (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 *et fils*

Capabilities

- Main advantage of capabilities is the fine-grained access control:
  - easy to provide access to specific agents
- Capability presets *prima facie* evidence of the *right to access*
  - capability =► *object identifier* (imples naming)
  - capability =► (set of) *access rights*

  ➔ any representation must contain object ID and access rights

  ➔ any representation must protect capability from forgery
- How are caps implemented and protected?
  - tagged — protected by hardware
  - partitioned/segregated — protected by software
  - sparse — protected by sparsity (probabilistically secure, like encryption)

Tagged Capabilities

- Tag bit(s) with every (group of) memory word(s)
  - tag identifies capabilities
  - capabilities are used and copied like “normal” pointers
  - hardware checks permissions when deferencing capability
  - modifications turn tags off (convert to plain data)
  - Only privileged instructions(kernel) can turn tags on
    - ➔ propagation easy
    - ➔ restriction requires kernel to make new capability
    - ➔ revocation virtually impossible (requires memory scan)
    - ➔ amplification possible (below)
    - ➔ accessibility virtually impossible to determine
    - ➔ protection domain difficult to establish
- IBM System/38, AS/400, i-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 invokation rights
- Each user has a *profile*, which is essentially a capability list
  - relevant for propagation only, not for access
- Capabilities can be of *profile adoption* type
  - on invokation, segment *owner's profile* is added to caller's PD
  - normal pointers can be de-references if the profile contains caps
  - on return, profile adoption is cancelled
  - user can denote a submset of thir profile to be used in adoption
    - called *profile propagation*

Tagged Capabilities Outside RAM

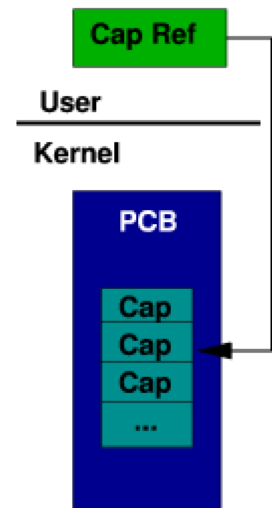
- Disk has no tags
- AS/400 simulates them by restricting physical I/O to low-level OS
  - AS/400 page isze is 4KiB
  - physical disk blocks are 520B, logical block size is 512B
  - extra 64B per page used to store tag bits (among others)
    - on page-out, page must be scanned and all tags collecte
    - on page-in, all tags are reconstructed
    - significant processing overhead on all I/O

Tagged Capabilities Summary

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

Partitioned Capabilities

- System maintains capability list (Clist) with each process
  - user code uses indirect references to caps (clist index)
    - c.f. Unix file descriptors
  - System validates access via clist when mapping any page
    - validation is explicit at map time
    - propagation: system call to copy between clients
    - restriction: kernel to make new capability
    - revocation: kernel to remove cap from clist
      - one specific or all
    - accessibility: requires scanning all clists
    - protection domain: explicitly represented in clist
- Hydra [CJ75], Mach [RTY+88], KeyKOS [BFF+92], EROS [SSF99], and many others



Propagating Partitioned Capabilities (Mach)

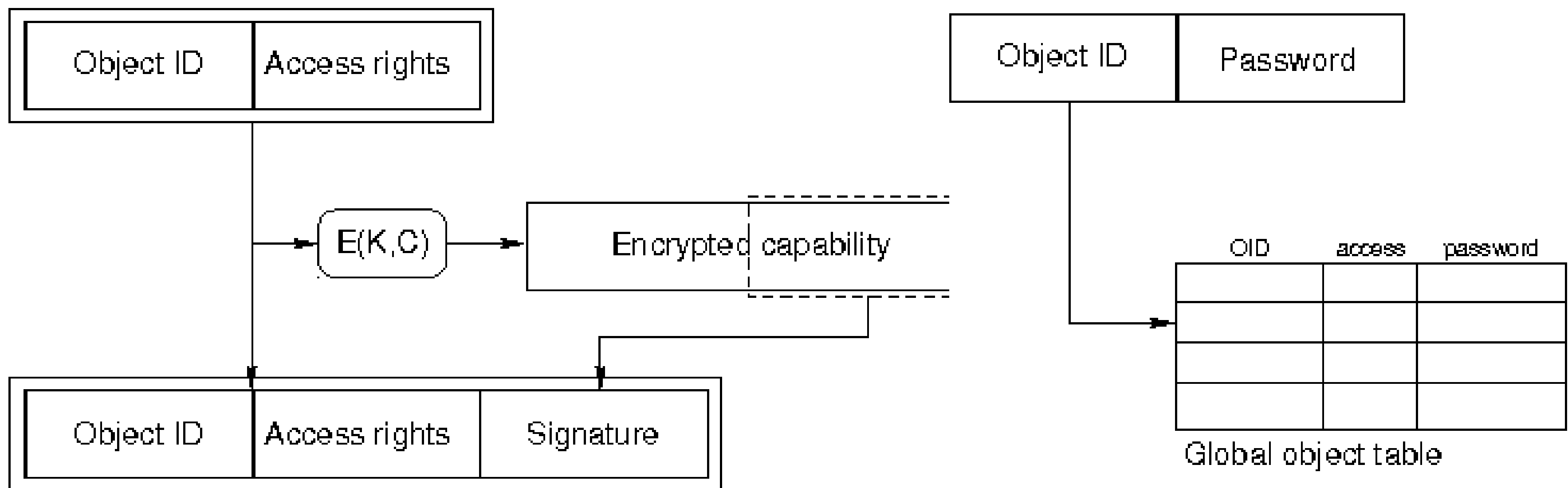
- Capabilities can be included in IPC messages
  1) User inserts clist indices into caps field in message
  2) kernel looks up clists and inserts representation of cap (marshalling)
  3) receiver's kernel inserts caps into receiver's clist
  4) kernel replaces global cap reference by local clist index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

Partitioned Capabilities Summary

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

Sparse Capabilities

- Basic idea similar to encryption
  - add bit string to make valid capabilities a very small subset of cap space
  - either encrypted object info [GL79] or password MT86, APW86]
  - secure by infeasibility of exhaustive search of cap space



Sparse Capabilities

- Sparse caps are user-level objects
  - can be passed like other data
    - similar to tagged caps, but without hardware support
    - validated at mapping time (explicit or implicit)
  - good match to user-level servers
    - no central authority, no kernel required on most ops
    - cannot reference-count objects
- Issues:
  - Full mediation requires extra work
    - but doable, see Mungi [HEV+98]
    - essentially provided user-level cap segregation
  - High amplification of leaked data
    - problem with convert channels

Confinement

- Problem 1: Executing untrusted code
  - you downloaded a game from the internet
  - how can you be sure it doesn't steal/corrupts 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 unauthorised copies?
- You need to *confine* the program (game, viewer) so it cannot leak
- Cannot be done with most protection schemes!
  - not with Unix or most other ACL-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
  - e.g segregated caps system, can instruct system not to accept any
  - EROS has formal proof of confinement of a model of the system [SW00]
- 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
  - term trusted computing now virtually synonymous with TCG model
- Defines Trusted Computing Module (TCM)
  - hardware root of trust, aimed at PC/server 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

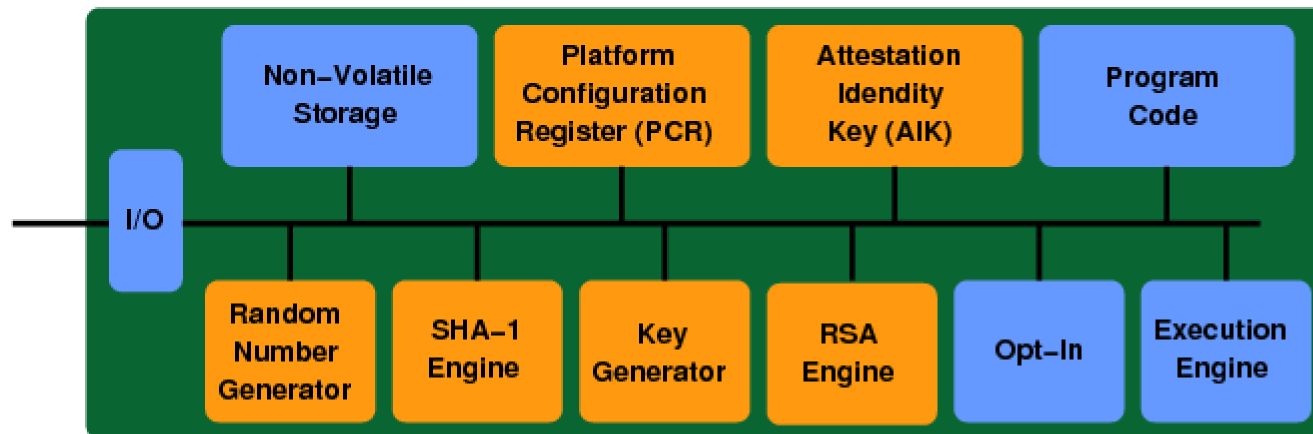
- Authenticated booting
  - bring up system in well-defined configuration
  - executing only certified binaries
- Remote attestation
  - allow remote party to confirm system configuration
- Sealed storage
  - ensure that data can only be read if system is in particular configuration

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)
  - hard-wired private key, uniquely identifies physical device
  - public EK certified and supplied by manufacturer
- Non-volatile storage
  - small amount for EK, some symmetric keys, opt-in flags
  - storage root key (SRK), protected by SRK 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 condensed (by SHA engine) measurement:
    PCR ← SHA-1(PCR || SHA-1 (component))
- Suffices 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 *root of trust for measurement* (RTM)

Remote Attestation (aka Integrity Reporting)

- Idea: Provide certified representation of machine state to challenger
  - e.g. service provider who insists on particular configuration
- Two parts reported
  - measurement log kept by software
  - PCR value (accumulated measurements) signed by endorsement key
    - alternatively can set up specific attestation identity key (AIK)
- Challenger can verify
  - recompute 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 “sender” 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 SRK, can then be stored anywhere
- Sealing:
  - RSA engine can optionally include PCR configuration in encryption
  - when encrypting key, include
    - present (“sender”) PCR state
    - desired (“receiver”) PCR state
  - only decrypt key if present PCR state matches “receiver” state
  - return “sender” 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 manufacturers key(s)
  - only load components after verifying signatures
  - *measure* components prior to executing
- Boot block loads first OS component
  - using TPM cryptography hardware
- First OS components contain
  - SW 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 bootstrap work by Arbaugh et al [AFS97]

Overview

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

Design Principles for Secure Systems

- Least privilege (POLA)
- 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
  - also called the principle of *least authority* (POLA)
  - minimal protection domain
  - PD determined by *function*, not *identity*
    - Unix root is evil
    - Aim 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!”
- Less code/features/stuff ═► less to get wrong
  - makes it easier to fix if something does go wrong
  - complexity is the natural enemy of security
- Also applies to interfaces, interactions, protocols, ...
- Also implies minimal TCB!

Complete Mediation

- Check every access
  - violated in Unix 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 done with tagged or sparse capability systems
- In practice conflicts with performance!
  - caching of buffers, file descriptors etc
  - otherwise unacceptable performance in distributed systems
- Should at least limite 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 poor security
    - e.g. US government's Clipper initiative ('92)
    - FCC still doesn't seem to understand this
- Note that this doesn't rule out passwords or secret keys
  - but their creation requires careful *cryptoanalysis*

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 mechanism ═▶ shared channel
  - potential covert channel
- Inherent conflict with other design imperatives
  - simplicity ═▶ shared mechanisms

Psychological Acceptability

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