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 behind a firewall?

- Example 2: Banking store's weekly earnings:
  - Is it secure to
    - ... ask a random customer to do it?
    - ... ask many random 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?
Security Policy

- The security policy specifies allowed and disallowed states of a system.
- An OS needs to ensure that no disallowed state is entered.
- OS mechanisms prevent transitions from allowed to disallowed states.
- Security policy needs to identify the assets to be secure.
- For computer security, assets are typically data.
- Perfect security is generally unachievable.
- Security policy needs to identify the assets to be secure.
- For computer security, assets are typically data.
- Perfect security is generally unachievable.

Data Security

Three aspects:
- Confidentiality: prevent theft of data
  - Concealing data from unauthorised agents
- Integrity: prevent damage to 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 P-attack
  - Weakness P-violation

Security Mechanisms

- Used to enforce security policy:
  - Computer access control (login authentication)
  - Operating system 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
  - 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

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
  - should really be called trustworthy computing
- TCB securely expanded by loading trusted components only
- Hardware- and software mechanisms to prevent tampering

Covert Channels (Side 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 temporal order of accesses to shared resources
  - Global names can 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 resources
  - Outside access-control system
  - Difficult to reason about
  - Difficult to prevent

Trusted Computing Base

- TCB: The totality of protection mechanisms within a computer system — the combination of which is responsible for enforcing a security policy
  - 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

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
- A trusted system can be used to process security-critical assets
  - gone through some process (“assurance”) to establish its trustworthiness
- Trusted computing:
  - provides mechanisms and procedures for trusted systems
  - in practice usually refers to TCG mechanisms for secure boot, encryption etc.
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
- 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

Establishing Trustworthiness

- Process to show TCB is trustworthy
  - Two approaches
    - assurance (systematic evaluation and testing)
    - formal verification (mathematical proof)
  - Certification confirms process was successfully concluded

Assurance

- Process for bolstering (substantiating or specifying) trustworthiness
  - Specifications
    - unambiguous description of system behaviour
    - can be formal (mathematical model) 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 code inspection and rigorous testing
    - by implication must also satisfy specification
  - Operation and maintenance
    - justification that system is used as per assumption in specification
- Assurance does not guarantee correctness or security!
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
  - 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

Summary

Operating systems security overview
- Types of secure systems
- Security policies
- Security mechanisms
- Trusted Computing
- Design principles
- OS security verification
- OS design for security

Secure Systems Classification
- Based on Orange Book terminology
  - assumes military-style security problem
  - data of different security classifications
  - system must ensure that classification is enforced
  - focused on confidentiality
- Classifies systems based on the kind of data they can deal with
  - single-level secure (SLS) system
  - multiple single-level secure (MSL) system
  - multi-level secure (MLS) system
- Basis of multiple-independent levels of security (MLS) architecture

Single-Level Secure (SLS) System
- Suitable only for processing data of one particular security level
  - generally the lowest, i.e. unclassified

Multi-Level Secure (MLS) System
- Suitable for processing data of several security levels
  - concurrently, up to some limit
  - needs to ensure that classifications are honoured
  - does this by labeling all data
- Requires mandatory access control in OS
MLS + MSL System

- MLS component handles multiple levels of data
- Only a single level of data goes to each of the MSL secure systems

Overview

- Operating systems security overview
- Types of secure systems
- Security policies
- Security mechanisms
- Trusted Computing
- Design principles
- OS security verification
- OS design for security

Security Policies: Categories

- Discretionary (user-controlled) policies (DAC)
  - e.g., A can read B's objects only with A's permission
  - user decides about access (at their discretion)
- Mandatory (system-controlled) policies (MAC)
  - e.g., certain users cannot ever access certain objects
  - no user can change these
  - focus on restricting information flow
  - inherent requirement for MLS systems, MILS
- Role-based policies (RBAC)
  - agents can take on specific pre-defined roles
  - well-defined set of roles for each agent
  - e.g., normal user, sysadmin, database admin
  - access rights depend on role

Models for Security Policies

- 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 the Bell-LaPadula model [Bell & LaPadula 76]
  - example of a labeled security model
  - most others developed from this
  - Orange Book based on this model
  - Chinese-wall policy focuses on conflict of interest
  - Some newer models focus on integrity
  - Biba model derived from Bell-LaPadula
  - Clark-Wilson model based on separation of duty
  - maps to role-based access control

Bell-LaPadula Model

- Each object $a$ has a security classification $L(a)$
- Each agent $o$ has a security clearance $L(o)$
- Classifications
  - e.g., top secret > secret > confidential > unclassified
- Rule 1 (no read up): a can read $o$'s objects only if $L(a) \leq L(o)$
- Rule 2 (no write down)
  - a can write only if $L(a) \geq L(o)$
- standard confidentiality
- Rule 2 (C2 Property — no write down)
  - a can write only if $L(a) = L(o)$
  - prevents leakage (accidental or by conspiracy)
Bell-LaPadula Model

- Mother of all military-style security models
- Inherently requires implementation as MAC
- All subjects must be bound to policy
- If implemented inside a single system, requires MLS system
- Major limitation: cannot deal with declassification
  - Need to pass any information from high- to low-security domain
  - Logging
  - Command chain
  - Documents where sensitive portions have been censored
  - Encrypted data
- Typically dealt with by special privileged functions
  - Outside security policy
  - Outside systematic reasoning
  - Part of TCB
  - Likely source of security holes

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 V property (cannot publish confidential info)
- Example of a dynamic MAC policy
  - Allowed information flow changes over time

Chinese Wall Policy

- In practice need a way to remove conflicts
  - Transaction completed...

Bibra Model

- Dual to Bell-LaPadula for integrity
- Each subject a, object o has an integrity level L
  - Rule 1 (no read down):
    - A can read o only if L(a) ≥ L(o)
  - Rule 2 (property — no write up):
    - A can write o only if L(a) ≥ L(o)
- Obviously incompatible with Bell-LaPadula
  - If higher security requires higher integrity
  - Must choose between confidentiality and integrity
- Bibra doesn’t model any practical system

Clark-Wilson Model

- Security framework for ensuring integrity based on separation of duties
  - Doesn’t provide specific state transformations, only constraints on them
  - Helps in formulating security policies
- Distinguishes constrained (integrity guaranteed) and unconstrained data
  - Operations on unconstrained data must be defined for all values and produce constrained data
- Specifies requirements on the system and its operations
  - Protect integrity-critical data, authentication, integrity of transformations, logging
  - Operations certified to operate on certain data
- Doesn’t actually specify what “separation of duties” means
  - “Allowed relations must meet the requirements of separation of duties”

Overview

- Operating systems security overview
- Types of secure systems
- Security policies
- Security mechanisms
- Trusted Computing
- Design principles
- OS security verification
- OS design for security
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
  - Own
  - Change rights of an object or agent

Access Control Matrix

<table>
<thead>
<tr>
<th>Agents</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$O_1$</td>
</tr>
<tr>
<td>write</td>
<td>read</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$O_2$</td>
</tr>
<tr>
<td>control</td>
<td>execute</td>
</tr>
<tr>
<td>$s_3$</td>
<td>write</td>
</tr>
</tbody>
</table>

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

Properties of the Access Control Matrix

- Rows define agents' protection domains (PDs)
- Columns define objects' accessibility
- Dynamic data structure:
  - Frequent permanent changes (e.g. object creation, chmod)
  - Frequent temporary changes (e.g. setuid)
- Very sparse with many repeated entries
- Impractical to store explicitly

Protection-Matrix Implementation: ACLs

Represent column-wise: access control list (ACL):
- ACL associated with object
- 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

Protection-Matrix Implementation: Capabilities

Represent row-wise: capabilities [Dennis & Van Horn 66]:
- Capability list associated with agent
  - each capability conveys a certain right to its holder
- Can have negative rights to:
  - reduce window of vulnerability
  - simplify management of groups of capabilities
- Caps have been popular in research for a long time
- Few successful commercial systems until recently:
  - main one is IBM System38 / A5400 / i-Series
  - increasingly appearing in commercial systems (usually add-on)

Capabilities

- Main advantage of capabilities is the fine-grained access control:
  - easy to provide specific agents access to individual objects
- Capability presets prima facie evidence of the right to access
  - capability bit on object signifies (implies) owning
  - capability bit (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
  - popular in the past, nearly today (exception: IBM i-Series)
  - sparse (or user-modifiable) — protected by sparsity
  - probabilistically secure, like encryption
  - propagation outside system control — hard to enforce security policies
  - partitioned/regressed — protected by software (kernel)
  - main version of caps used in modern systems
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 dereferencing capability
  - modifications turn tags off (convert to plain data)
  - issues:
    - capability hardware tends to be slow (too complex)
    - hard (if not impossible) to control propagation of authority
    - revocation virtually impossible (requires memory scan)
    - amplification possible (below)

- IBM System/38, AS/400, i-Series, many historical systems

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 or password
  - secure by infeasibility of exhaustive search of cap space

- IBM System/38, AS/400, i-Series, many historical systems

Issues:
- "capability hardware tends to be slow (too complex)
- hard (if not impossible) to control propagation of authority
- revocation virtually impossible (requires memory scan)
- amplification possible (below)

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
- issues:
  - Full mediation requires extra work
  - but doable, see Mungi [Heiser et al. 98]

Segregated (Partitioned) Capabilities

- System maintains capability list (Clist) with each agent (process)
  - User code uses indirect references to caps (clist index)
  - System validates permissions on access
  - Many research systems
  - Hydra, Mach, EROS, and many others
  - Increasingly commercial systems
  - KeyKOS (92), OKL4 (08), seL4 (09)

Confinement

- Problem 1: Executing untrusted code
  - you downloaded a game from the internet
  - how can you be sure it doesn’t steal/Corrupt 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
  - 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 for system model [Shapiro & Weber 00]
  - similar for seL4 (machine-checked proof)

In practice difficult to achieve due to covert channels
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
- 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)

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 — meant for use before enough trusted software is booted
- Endorsement key (EK)
  - small amount for EK, some symmetric keys, opt-in flags
  - storage root key (SRK), protected by SRK pass phrase
- Non-volatile storage
  - required minimum for EK, some symmetric keys, opt-in flags
- Firmware configuration registers (PCR)
  - SHA-1 Magic Values

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 (“extended”) measurement:
    - PCR = SHA-1(PCR || SHA-1 (component))

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 if 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 to authenticate
- First OS components contains
  - 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 [Arbaugh et al. 97]

Trusted Computing vs Secure OS

- TPM-based trusted-computing approach is based on
  - Hardware root of trust
  - Mechanisms to provide a chain of trust
  - Not designed to protect against hardware attacks
- Objective is to guarantee that system boots into a well-defined configuration
  - Guarantees that a particular OS binary is running
  - What does this mean about security/trustworthiness?

Overview

- Operating systems security overview
- Types of secure systems
- Security policies
- Security mechanisms
- Trusted Computing
- Design principles
- OS security verification
- OS design for security

Design Principles for Secure OS

- Least privilege (POLA)
- Economy of mechanisms
- Fail-safe defaults
- Complete mediation
- Open design
- Separation of privilege
- Least common mechanisms
- Psychological acceptability
  [Saltzer & Schroeder 1975]
Least Privilege

- Also called the principle of least authority (POLA)
- Military "need to know" is an example
- Agent should only be given the minimal rights needed for task
  - 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
  - violated by all mainstream OSes
- Example: executing web applet
  - should not have all of user's privileges, only minimal access
  - hard to do with AGL-based systems
  - main motivation for using caps

Least Privilege: Implications for OS

- OS kernel executes in privileged mode of hardware
  - kernel has unlimited privilege
- POLA implies keeping kernel code to an absolute minimum
  - this means a secure OS must be based on a microkernel
- Trusted computing base can bypass security
- POLA requires that TCB is minimal
  - microkernel plus minimal security manager

Economy of Mechanisms

- KISS principle of engineering
  - "keep it simple, stupid!"
- Less code/features/stuff b 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, ...
- Specifically applies to TCB
- Reflected in CC EAL7 "design minimality" requirement
  - justified by need for manual inspection
  - trumped by formal methods

Fail-Safe Defaults

- Default action is no-access
  - if action fails, system remains secure
  - if security administrator forgets to add rule, system remains secure
  - eg, default firewall setting of OS should be "secure", not "open!"
  - "better safe than sorry"

Complete Mediation

- Reference monitor checks 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
    - without caching unacceptable performance
  - Should at least limit window of opportunity
    - e.g guarantee caches are flushed after some fixed period
    - guarantee no cached access after revoking access

Open Design

- Security must not depend on secrecy of design or implementation
  - TCB must be open to scrutiny
  - Security by obscurity is poor security
  - Not all security/certification agencies seem to understand this
  - Note that this doesn’t rule out passwords or secret keys
  - ... but their creation requires careful cryptanalysis
Separation of Privilege

- Require a combination of conditions for granting access
  - e.g., user is in group wheel and knows the root password
  - Take-grant model for capability-based protection:
    - sender needs grant right on capability
    - receiver needs take right to accept capability
  - In reality, the security benefit of a separate take right is minimal
  - practical cap implementations only provide grant as a privilege

Least Common Mechanisms

- Avoid sharing mechanisms
  - shared mechanism if shared channel
  - potential covert channel
- Also, shared mechanisms lead to complex interdependencies
  - makes it harder to analyse system for correctness
  - more likely to hide security problems
  - many classical security breaches result from unforeseen interactions
- Inherent conflict with other design imperatives
  - simplicity if shared mechanisms
  - classical tradeoff...

Psychological Acceptability

- Security mechanisms should provide a simple mental model
  - users need to understand the consequences of their actions
- 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

Overview

- Operating systems security overview
- Types of secure systems
- Security policies
- Security mechanisms
- Trusted Computing
- Design principles
- OS security verification
- OS design for security

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 EAL6–7

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 and tested
  - design docs used for testing, avoid tampering during delivery
  - independent vulnerability analysis
  - highest level feasible on existing product (not developed for CC certification)
  - achieved by a number of mainstream OSes
    - SUSE Enterprise Linux: EAL4 in 2005
    - Solaris-10: EAL4+ in 2006
    - controlled access protection profile (CAPP) — Note: EAL3 profile
    - role-based access control PP — example of non-NSA PP?
  - they still 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

- They still 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

- EAL5: semi-formally designed and tested
  - formal model of TEO security policy
  - semi-formal model of functional spec & high-level design
  - semi-formal arguments 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 TOE design
    - systematic covert-channel identification
    - Green Hills Integrity microkernel presently undergoing EAL6+ certification
    - separation kernel protection profile

- EAL6: semiformally verified design and tested
  - semiformal low-level design
  - structured representation of implementation
  - modular and layered TOE design
  - systematic covert-channel identification
  - Green Hills Integrity microkernel presently undergoing EAL6+ certification
  - separation kernel protection profile

- 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
  - Lynxsoft/Iforks 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 LLD, implementation
  - Hence no requirement for formal proof of implementation correctness

- EAL6: semiformally verified design and tested
  - semiformal low-level design
  - structured representation of implementation
  - modular and layered TOE design
  - systematic covert-channel identification
  - Green Hills Integrity microkernel presently undergoing EAL6+ certification
  - separation kernel protection profile

- 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
  - Lynxsoft/Iforks 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 LLD, implementation
  - Hence no requirement for formal proof of implementation correctness

- Common Criteria Limitations
  - Little (if any) use in commercial space outside national security
    - This was one of the intentions — by all indications, CC failed here
  - Very expensive
    - industry rule-of-thumb: EAL6+ costs $10k per LOC
    - dominated by documentation requirements
    - no “credit” for doing things better
      - eg formal methods instead of excessive documentation
  - Lower EALs of limited practical use
    - Windows is EAL4+ certified!
    - marketing seems to be main driver behind EAL3–4 certification
  - Over-evaluation abuses system
    - eg. CAPP (EAL3 profile) certification to EAL4
    - in reality a pointless exercise

- Formal Verification
  - Based on mathematical model of the system
  - Complete verification requires two parts:
    - proof that model satisfies requirements of security policies
      - typically prove generic properties that actual policies map to
        - required by CC: EAL5–7
      - proof of correspondence between model and implementation
        - not required by CC even at EAL7
        - done by some kernels with very limited functionality
        - never done for any general-purpose OS!
    - Model-checking (static analysis) is incomplete formal verification
      - shows presence or absence of certain properties
      - eg uninitialed variables, array bounds overflows
      - nevertheless useful for assurance
Overview

- Operating systems security overview
- Types of secure systems
- Security policies
- Security mechanisms
- Trusted Computing
- Design principles
- OS security verification
- OS design for security

OS Design for Security

- Minimize kernel code
  - kernel = code that executes in privileged mode
  - kernel can bypass any security
  - kernel is inherently part of TCB
  - kernel can only be verified as a whole (not in components)
  - it’s hard enough to verify a minimal kernel
- How?
  - generic mechanisms (economy of mechanisms)
  - no policies, only mechanisms
  - mechanisms as simple as possible
  - only code that must be privileged in order to support secure systems
  - free of covert channels: no global names, absolute time
  - Formally specify API

Minimize mandatory TCB

- unless formally verified, TCB must be assumed imperfect
- the smaller, the fewer defects
- POLA requires, economy of mechanisms leads to minimal TCB
- Ensure TCB is well defined and understood
  - make security policy explicit
  - make granting of authority explicit
- Flexibility to support various uses
  - make authority delegatable
  - ensure mechanisms allow high-performance implementation
- Design for verifiability
  - minimize implementation complexity

Example: NICTA’s seL4

- High-security version of L4 microkernel API
  - all authority granted by capabilities
    - full mediation, least privilege, separation of privilege, fail-safe defaults
  - only four system calls: read, write, create, derive
  - economy of mechanisms
  - semi-formal and formal models and design specs
    - open design (once published)
  - kernel memory explicitly managed by user-level resource manager
    - least privilege, separation of privilege
  - 7,000–10,000 lines of kernel code
    - least privilege
- Details later...