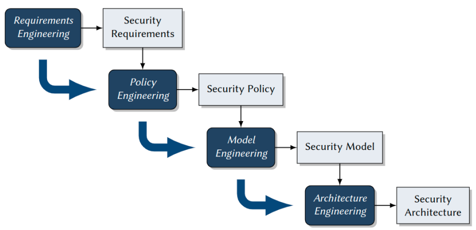
Goal of IT Security **Reduction of Operational Risks of IT Systems**

- **Confidentiality** the property of information to be available only to an authorized user group
- **Integrity** the property of information to be protected against unauthorized modification
- **Availability** the property of information to be available in an reasonable time frame
- **Authenticity** the property to be able to identify the author of an information
- **Conditio sine qua non** Provability of information properties
- **Non-repudiability** the combination of integrity and authenticity
- **Safety** To protect environment against hazards caused by system failures
  - Technical failures: power failure, ageing, dirt
  - Human errors: stupidity, lacking education, carelessness
  - Force majeure: fire, lightning, earth quakes
- **Security** To protect IT systems against hazards caused by malicious attacks
  - Industrial espionage, fraud, blackmailing
  - Terrorism, vandalism

Security Engineering

- Is a methodology that tries to tackle this complexity.
- Goal: Engineering IT systems that are secure by design.
- Approach: Stepwise increase of guarantees



Security Requirements

Methodology for identifying and specifying the desired security properties of an IT system.

- Security requirements, which define what security properties a system should have.
- These again are the basis of a security policy: Defines how these properties are achieved

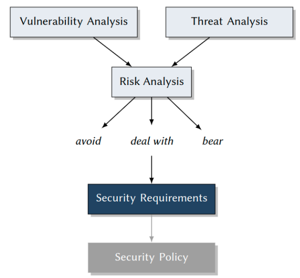
Influencing Factors

- Codes and acts (depending on applicable law)
  - EU General Data Protection Regulation (GDPR)
  - US Sarbanes-Oxley Act (SarbOx)
- Contracts with customers
- Certification
  - For information security management systems (ISO 27001)
  - Subject to German Digital Signature Act (Signaturgesetz)
- Company-specific guidelines and regulations
  - Access to critical data
  - Permission assignment
- Company-specific infrastructure and technical requirements
  - System architecture
  - Application systems (OSs, Database Information Systems)

Specialized steps in regular software requirements engineering

1. Identify and classify vulnerabilities
2. Identify and classify threats
3. Match both, where relevant, to yield risks
4. Analyze and decide which risks should be dealt with

→ Fine-grained Security Requirements



Vulnerability Analysis

Identification of technical, organizational, human vulnerabilities of IT systems.

Vulnerability Feature of hardware and software constituting, an organization running, or a human operating an IT system, which is a necessary precondition for any attack in that system, with the goal to compromise one of its security properties. Set of all vulnerabilities = a system's attack surface.

Human Vulnerabilities

- Laziness
  - Passwords on Post-It
  - Fast-clicking exercise: Windows UAC pop-up boxes
- Social Engineering
  - Pressure from your boss
  - A favor for your friend
  - Blackmailing: The poisoned daughter, ...
- Lack of knowledge
  - Importing and executing malware
  - Indirect, hidden information flow in access control systems
- Limited knowledge/skills of users

Social Engineering Influencing people into acting against their own interest or the interest of an organisation is often a simpler solution than resorting to malware or hacking.

Indirect Information Flow in Access Control Systems

Security Requirement No internal information about a project, which is not approved, should ever go public

Forbidden Information Flow Internal information goes into unwanted publicity

Problem Analysis

- Problem complexity → effects of individual permission assignments by users to system-wide security properties
- Limited configuration options and granularity: archaic and inapt security mechanisms in system and application software
  - no isolation of non-trusted software
  - no enforcement of global security policies

→ Effectiveness of discretionary access control (DAC)

Organizational Vulnerabilities

- Access to rooms (servers)
- Assignment of permission on organizational level, e. g.
  - 4-eyes principle
  - need-to-know principle
  - definition of roles and hierarchies
- Management of cryptographic keys

Technical Vulnerabilities

The Problem: Complexity of IT Systems

- ... will in foreseeable time not be
- Completely, consistently, unambiguously, correctly specified → contain specification errors
- Correctly implemented → contain programming errors
- Re-designed on a daily basis → contain conceptual weaknesses and vulnerabilities
- Weak security paradigms

Threat Analysis

- Identification of Attack objectives and attackers
- Identification of Attack methods and practices (Techniques)

→ know your enemy

Approach: Compilation of a threat catalog, content:

- identified attack objectives
- identified potential attackers
- identified attack methods & techniques
- damage potential of attacks

Attack Objectives and Attackers

- Economic Espionage and political power
  - Victims: high tech industry
  - Attackers:
    - Competitors, governments, professional organizations
    - Insiders
    - regular, often privileged users of IT systems
  - often indirect → social engineering
  - statistical profile: age 30-40, executive function
  - weapons: technical and organisational insider knowledge
  - damage potential: Loss of control over critical knowledge → loss of economical or political power
- Personal Profit
  - Objective: becoming rich(er)
  - Attackers: Competitors, Insiders
  - damage potential: Economical damage (loss of profit)
- Wreak Havoc
  - Objective: damaging or destroying things or lives, blackmailing,...
  - Attackers:
    - Terrorists: motivated by faith and philosophy, paid by organisations and governments
    - Avengers: see insiders
    - Psychos: all ages, all types, personality disorder
    - → No regular access to IT systems, no insider knowledge, but skills and tools.
  - damage potential: Loss of critical infrastructures
- Meet a challenge (Hackers both good or evil)

Attack Methods

Scenario 1: Insider Attack

- Social Engineering
- Exploitation of conceptual vulnerabilities (DAC)
- Professionally tailored malware

Scenario 2: Malware (a family heirloom ...)

- Trojan horse: Executable code with hidden functionality
- Virus: Code for self-modification and self-duplication
- Logical bomb: Code that is activated by some event recognizable from the host (e. g. time, date, temperature, ...).
- Backdoor: Code that is activated through undocumented interfaces (mostly remote).
- Ransomware: Code for encrypting possibly all user data found on the host, used for blackmailing the victims
- Worm: Autonomous, self-duplicating programs

Scenario 3: Outsider Attack

- Attack Method: Buffer Overflow
- Exploitation of implementation errors

Buffer Overflow Attacks

Privileged software can be tricked into executing attacker's code.
Approach: Cleverly forged parameters overwrite procedure activation frames in memory → exploitation of missing length checks on input buffers → buffer overflow
What an Attacker Needs to Know

- Source code of the target program, obtained by disassembling
- Better symbol table, as with an executable
- Better most precise knowledge about the compiler used (Stack)

Sketch of the Attack Approach (Observations during program execution)

- Stack grows towards the small addresses
- in each procedure frame: address of the next instruction to call after the current procedure returns (ReturnIP)
- after storing the ReturnIP, compilers reserve stack space for local variables → these occupy lower addresses

Result

- Attacker makes victim program overwrite runtime-critical parts of its stack
  - by counting up to the length of msg
  - at the same time writing back over previously save runtime information → ReturnIP
- After finish: victim program executes code at address of ReturnIP (=address of a forged call to execute arbitrary programs)
- Additional parameter: file system location of a shell

Security Breach The attacker can remotely communicate, upload, download, and execute anything- with cooperation of the OS, since all of this runs with the original privileges of the victim program!

Scenario 4: High-end Malware (Root Kits)

- Invisible, total, sustainable takeover of a complete IT system
- Method: Comprehensive tool kit for fully automated attacks
  1. automatic analysis of technical vulnerabilities
  2. automated attack execution
  3. automated installation of backdoors
  4. installation and activation of stealth mechanisms
- Target: Attacks on all levels of the software stack:
  - firmware & bootloader
  - operating system (e. g. file system, network interface)
  - system applications (e. g. file and process managers)
  - user applications (e. g. web servers, email, office)
- tailored to specific software and software versions found there

Root Kits

Step 1: Vulnerability Analysis

- Tools look for vulnerabilities in
  - Active privileged services and demons
  - Configuration files → Discover weak passwords, open ports
  - Operating systems → Discover kernel and system tool versions with known implementation errors
- built-in knowledge base: automatable vulnerability database
- Result: System-specific collection of vulnerabilities → choice of attack method and tools to execute

Step 2: Attack Execution

- Fabrication of tailored software to exploit vulnerabilities in
  - Server processes or system tool processes (demons)
  - OS kernel to execute code of attacker with root privileges
- This code
  - First installs smoke-bombs for obscuring attack
  - replaces original system software by pre-fabricated modules
  - containing backdoors or smoke bombs for future attacks
- Backdoors allow for high-privilege access in short time
- System modified with attacker's servers, demons, utilities...
- Obfuscation of modifications and future access

Step 3: Attack Sustainability

- Backdoors for any further control & command in Servers, ...
- Modifications of utilities and OS to prevent
  - Killing root kit processes and connections (kill,signal)
  - Removal of root kit files (rm,unlink)
- Results: Unnoticed access for attacker anytime, highly privileged, extremely fast, virtually unpreventable

Step 4: Stealth Mechanisms (Smoke Bombs)

- Clean logfiles (entries for root kit processes, network connections)
- Modify system admin utilities
  - Process management (hide running root kit processes)
  - File system (hide root kit files)
  - Network (hide active root kit connections)
- Substitute OS kernel modules and drivers (hide root kit processes, files, network connections), e.g. /proc/..., stat, fstat, pstat
- Processes, files and communication of root kit become invisible

Risk and Damage Potential:

- Likeliness of success: extremely highin today's commodity OSs (High number of vulnerabilities, Speed, Fully automated)
- Fighting the dark arts: extremely difficult (Number and cause of vulnerabilities, weak security mechanisms, Speed, Smoke bombs)
- Prospects for recovering the system after successful attack ~ 0

Countermeasure options

- Reactive: even your OS might have become your enemy
- Preventive: Counter with same tools for vulnerability analysis
- Preventive: Write correct software

Security Engineering

- New paradigms: policy-controlled systems → powerful software platforms
- New provable guarantees: formal security models → reducing specification errors and faults by design
- New security architectures → limiting bad effects of implementation errors and faults

Risk Analysis

Identification and Classification of scenario-specific risks

- Risks $\subseteq$ Vulnerabilities $\times$ Threats
- Correlation of vulnerabilities and threats → Risk catalogue
- n Vulnerabilities, m Threats → x Risks
- $max(n,m) << x \leq nm$ → quite large risk catalogue
- Classification of risks → Complexity reduction → Risk matrix

Damage Potential Assessment

- Cloud computing → loss of confidence/reputation
- Industrial plant control → damage or destruction of facility
- Critical public infrastructure → impact on public safety
- Traffic management → maximum credible accident

Occurrence Probability Assessment

- Cloud computing → depending on client data sensitivity
- Industrial plant control → depending on plant sensitivity
- Critical public infrastructure → depending on terroristic threat
- Traffic management → depending on terroristic threat level

Damage potential & Occurrence probability is scenario-specific

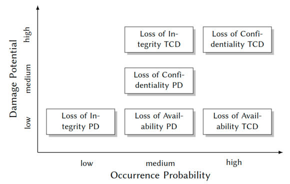
Depends on diverse, mostly non-technical side conditions → advisory board needed for assessment

Advisory Board Output Example

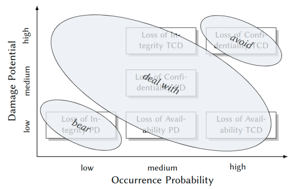
Object	Risk (Loss of...)	Dmg. Pot.	Rationale
PD	Integrity	low	Errors fast and easily detectable and correctable
PD	Integrity	low	Certified software, small incentive
PD	Availability	low	Failures up to one week can be tolerated by manual procedures
PD	Availability	med	Certified software
PD	Confidentiality	med	Data protection acts
PD	Confidentiality	med	Certified software
TCD	Availability	low	Minimal production delay, since backups are available
TCD	Availability	low	Small gain by competitors or terroristic attackers
TCD	Integrity	med	Medium gain by competitors or terroristic attackers
TCD	Integrity	high	Production downtime
TCD	Confidentiality	high	Huge financial gain by competitors
TCD	Confidentiality	high	Loss of market leadership

PD = Personal Data; TCD = Technical Control Data

Resulting Risk Matrix



Identify 3 Regions



- **avoid** Intolerable risk, no reasonable proportionality of costs and benefits → Don't implement such functionality
- **bear** Acceptable risk → Reduce economical damage (insurance)
- **deal with** Risks that yield security requirements → Prevent or control by system-enforced security policies

Additional Criteria:

- Again, non-technical side conditions may apply:
  - Expenses for human resources and IT
  - Feasibility from organizational and technological viewpoints

→ Cost-benefit ratio: management and business experts involved

Security Policies and Models

- protect against collisions → Security Mechanisms
→ Competent & coordinated operation of mechanisms → Security Policies
→ Effectiveness of mechanisms and enforcement of security policies

Security Policies: a preliminary Definition

- Malware attack → violation of confidentiality and integrity
- infer security requirements: Valid information flows
- design a security policy: Rules for controlling information flows

Security Policy a set of rules designed to meet a set of security objectives

Security Objective a statement of intent to counter a given threat or to enforce a given security policy

Policy representations:

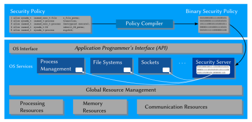
- informal (natural language) text
- formal model
- functional software specification
- executable code

How to Implement Security Policies

- (A) Integrated in systems software (Operating, Database)
- (B) Integrated in application systems

Implementation Alternative A

The security policy is handled an OS abstractionon its own → implemented inside the kernel

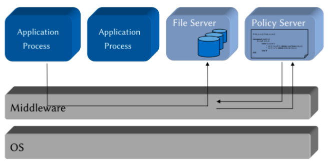


Policy Enforcement in SELinux

- **Security Server** Policy runtime environment
- **Interceptors** Total control of critical interactions
- **Policy Compiler** Translates human-readable policy modules in kernel-readable binary modules
- **Security Server** Manages and evaluates these modules

Implementation Alternative B

- **Application-embedded Policy** policy is only known and enforced by a user program → implemented in a user-space application
- **Application-level Security Architecture** policy is known and enforced by several collaborating user programs in an application systems → implemented in a local, user-space security architecture
- **Policy Server Embedded in Middleware** policy is communicated and enforced by several collaborating user programs in a distributed application systems → implemented in a distributed, user-space security architecture



Security Models

Complete, unambiguous representation of security policies for

- analyzing and explaining its behavior
- enabling its correct implementation

How We Use Formal Models: Model-based Methodology

- Abstraction from (too complex) reality → get rid of details
- Precision in describing what is significant → Model analysis and implementation

Security Model A security model is a precise, generally formal representation of a security policy.

Model Spectrum

- Models for access control policies:
  - identity-based access control (IBAC)
  - role-based access control (RBAC)
  - attribute-based access control (ABAC)
- Models for information flow policies → multilevel security (MLS)
- Models for non-interference/domain isolation policies → non-interference (NI)
- In Practice: Most often hybrid models

Access Control Models

Formal representations of permissions to execute operations on objects
Security policies describe access rules → security models formalize them

Identity-based access control models (IBAC) Rules based on the identity of individual subjects (users, processes, ...) or objects (files, ...)

Role-based access control models (RBAC) Rules based on roles of subjects in an organization

Attribute-based access control models (ABAC) Rules based on attributes of subjects and objects

Discretionary Access Control (DAC) Individual users specify access rules to objects within their area of responsibility (at their discretion).

Consequence: Individual users

- granting access permissions as individually needed
- need to collectively enforce their organization's security policy
  - competency problem
  - responsibility problem
  - malware problem

Mandatory Access Control (MAC) System designers and administrators specify system-wide rules, that apply for all users and cannot be sidestepped.

Consequence:

- Limited individual freedom
- Enforced by central instance:
  - clearly identified
  - competent (security experts)
  - responsible (organizationally & legally)

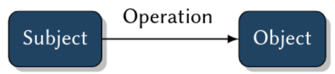
DAC vs. MAC

In Real-world Scenarios: Mostly hybrid models enforced by both discretionary and mandatory components

- **DAC** locally within a project, team members individually define permissions w. r. t. documents inside this closed scope
- **MAC** globally for the organization, such that e. g. only documents approved for release by organizational policy rules may be accessed from outside a project's scope

Identity-based Access Control Models (IBAC)

To precisely specify the rights of individual, acting entities.



- **Subjects**, i.e. active and identifiable entities, that execute
- **Operations** on
- passive and identifiable **Objects**, requiring
- **Rights** (also: permissions, privileges) which
  - control (restrict) execution of operations,
  - are checked against identity of subjects and objects.

Access Control Functions [Lampson, 1974]

- basic model to define access rights: Who (subject) is allowed to do what (operation) on which object
- Access Control Function (ACF)
  - $f: S \times O \times OP \rightarrow \{true, false\}$ where
  - S is a set of subjects (e.g. users, processes),
  - O is a set of objects (e.g. files, sockets),
  - OP is a finite set of operations (e.g. read, write, delete)
- Interpretation: Rights to execute operations are modeled by ACF
  - any $s \in S$ represents an authenticated active entity which potentially executes operations on objects
  - any $o \in O$ represents an authenticated passive entity on which operations are executed
  - for any $s \in S$,$o \in O$,$op \in OP$: s is allowed to execute op on o iff $f(s, o, op) = true$.
  - Model making: finding a $tuple\langle S, O, OP, f\rangle$

Access Control Matrix

Lampson addresses how to ...

- store in a well-structured way,
- efficiently evaluate and
- completely analyze an ACF

Access Control Matrix (ACM) An ACM is a matrix $m: S \times O \rightarrow 2^{OP}$, such that $\forall s \in S, \forall o \in O: op \in m(s,o) \Leftrightarrow f(s, o, op)$.

An ACM is a rewriting of the definition of an ACF: nothing is added, nothing is left out („⇔").

- $S = \{s_1, \dots, s_n\}$
- $O = \{o_1, \dots, o_k\}$
- $OP = \{read, write\}$
- $2^{OP} = \{\varnothing, \{read\}, \{write\}, \{read, write\}\}^2$

- ACMs are implemented in most OS, DB, Middlewear
- whose security mechanisms use one of two implementations

Access Control Lists (ACLs)

- Columns of the ACM: $char * o3[N] = \{'-', '-', 'rw', \dots\};$
- Found in I-Nodes of Unix, Windows, Mac OS

Capability Lists

- Rows of the ACM: $char * s1[K] = \{'-', 'r', '-', \dots\}$;
- Found in distributed OSs, middleware, Kerberos

Protection State A fixed-time snapshot of all active entities, passive entities, and any meta-information used for making access decisions is called the protection state of an access control system.

ACF/ACM are to precisely specify a protection state of an AC system

The Harrison-Ruzzo-Ullman Model (HRU)

Privilege escalation question: „Can it ever happen that in a given state, some specific subject obtains a specific permission?" $\varnothing \Rightarrow \{r, w\}$

- ACM models a single state $\langle S, O, OP, m\rangle$
- ACM does not tell anything about what might happen in future
- Behavior prediction → proliferation of rights → HRU safety

We need a model which allows statements about

- Dynamic behavior of right assignments
- Complexity of such an analysis

Idea [Harrison et al., 1976]: A (more complex) security model combining

- Lampson's ACM → for modeling single protection state of an AC
- Deterministic automata → for modeling runtime changes of a protection state

Deterministic Mealy Automata $(Q, \sum, \Omega, \delta, \lambda, q_0)$

- Q is a finite set of states, e. g. $Q = \{q_0, q_1, q_2\}$
- $\sum$ is a finite set of input words, e. g. $\sum = \{a, b\}$
- Ω is a finite set of output words, e. g. $\Omega = \{yes, no\}$
- $\delta : Q \times \sum \rightarrow Q$ is the state transition function
- $\lambda : Q \times \sum \rightarrow \Omega$ is the output function
- $q_0 \in Q$ is the initial state
- $\delta(q, \sigma) = q'$ and $\lambda(q, \sigma) = \omega$ can be expressed through the state diagram

HRU Security Model How we use Deterministic Automata

- Snapshot of an ACM is the automaton's state
- Changes of the ACM during system usage are modeled by state transitions of the automaton
- Effects of operations that cause such transitions are described by the state transition function
- Analyses of right proliferation (→ privilege escalation) are enabled by state reachability analysis methods

An HRU model is a deterministic automaton $\langle Q, \sum, \delta, q_0, R\rangle$ where

- $Q = 2^S \times 2^O \times M$ is the state space where
  - S is a (not necessarily finite) set of subjects,
  - O is a (not necessarily finite) set of objects,
  - $M = \{m | m : S \times O \rightarrow 2^R\}$ is a set of possible ACMs,
- $\sum = OP \times X$ is the (finite) input alphabet where
  - OP is a set of operations,
  - $X = (S \cup O)^k$ is a set of k-dimensional vectors of arguments (subjects or objects) of these operations,
- $\sigma : Q \times \sum \rightarrow Q$ is the state transition function,
- $q_0 \in Q$ is the initial state,
- R is a (finite) set of access rights.
- Each $q = S_q, O_q, m_q \in Q$ models a system's protection state:
  - current subjects set $S_q \subseteq S$
  - current objects set $O_q \subseteq O$
  - current ACM $m_q \in M$ where $m_q : S_q \times O_q \rightarrow 2^R$
- State transitions modeled by δ based on
  - the current automaton state
  - an input word $\langle op, (x_1, \dots, x_k)\rangle \in \sum$ where op
  - may modify S_q (create a user x_i),
  - may modify O_q (create/delete a file x_i),
  - may modify the contents of a matrix cell $m_q(x_i, x_j)$ (enter or remove rights) where $1 \leq i, j \leq k$.
  - → We also call δ the state transition scheme (STS) of a model

State Transition Scheme (STS) Using the STS,

$\sigma : Q \times \sum \rightarrow Q$ is defined by a set of specifications in the normalized form $\sigma(q, \langle op, (x_1, \dots, x_k)\rangle)$=if $r_1 \in m_q(x_{s1}, x_{o1}) \wedge \dots \wedge r_m \in m_q(x_{sm}, x_{om})$ then $p_1 \circ \dots \circ p_n$ where

- $q = \{S_q, O_q, m_q\} \in Q, op \in OP$
- $r_1 \dots r_m \in R$
- $x_{s1}, \dots, x_{sm} \in S_q$ and $x_{o1}, \dots, x_{om} \in O_q$ where s_i and o_i, $1 \leq i \leq m$, are vector indices of the input arguments: $1 \leq s_i, o_i \leq k$
- $p_1, \dots, p_n$ are HRU primitives
- $\circ$ is the function composition operator: $(f \circ g)(x) = g(f(x))$

Conditions: Expressions that need to evaluate „true" for state q as a necessary precondition for command op to be executable (= can be successfully called).
Primitives: Short, formal macros that describe differences between q and a successor state $q' = \sigma(q, \langle op, (x_1, \dots, x_k)\rangle)$ that result from a complete execution of op:

- enter r into $m(x_s, x_o)$
- delete r from $m(x_s, x_o)$
- create subject x_s
- create object x_o
- destroy subject x_s
- destroy object x_o
- Each above with semantics for manipulating S_q, O_q or m_q.

Note the atomic semantics: the HRU model assumes that each command successfully called is always completely executed!
How to Design an HRU Security Model:

1. Model Sets: Subjects, objects, operations, rights → define the basic sets S, O, OP, R
2. STS: Semantics of operations (e. g. the future API of the system to model) that modify the protection state → define σ using the normalized form/programming syntax of the STS
3. Initialization: Define a well-known initial state $q_0 = \langle S_0, O_0, m_0\rangle$ of the system to model

Summary: Model Behavior

- The model's input is a sequence of actions from OP together with their respective arguments.
- The automaton changes its state according to the STS and the semantics of HRU primitives.
- In the initial state, each subject may (repeatedly) use a right on an object

HRU Model Analysis Analysis of Right Proliferation

HRU Safety (also simple-safety) A state q of an HRU model is called HRU safe with respect to a right $r \in R$ iff, beginning with q, there is no sequence of commands that enters r in an ACM cell where it did not exist in q.

Transitive State Transition Function δ^*: Let $\sigma\sigma \in \sum^*$ be a sequence of inputs consisting of a single input $\sigma \in \sum \cup \{\epsilon\}$ followed by a sequence $\sigma \in \sum^*$, where ϵ denotes an empty input sequence. Then, $\delta^* : Q \times \sum^* \rightarrow Q$ is defined by

- $\delta^*(q, \sigma\sigma^*) = \delta^*(\delta(q, \sigma), \sigma^*)$
- $\delta^*(q, \epsilon) = q$.

According to Tripunitara and Li, simple-safety is defined as:

HRU Safety For a state $q = \{S_q, O_q, m_q\} \in Q$ and a right $r \in R$ of an HRU model $\langle Q, \sum, \delta, q_0, R\rangle$, the predicate $safe(q, r)$ holds iff $\forall q' = S_{q'}, O_{q'}, m_{q'} \in \{\delta^*(q, \sigma^*) | \sigma^* \in \sum^*\}, \forall s \in S_{q'}, \forall o \in O_{q'} : r \in m_{q'}(s, o) \Rightarrow s \in S_q \wedge o \in O_q \wedge r \in m_q(s, o)$. We say that an HRU model is safe w.r.t. r iff $safe(q_0, r)$.

showing that an HRU model is safe w.r.t. r means to

1. Search for any possible (reachable) successor state q' of q_0
2. Visit all cells in $m_{q'}$ ($\forall s \in S_{q'}, \forall o \in O_{q'} : \dots$)
3. If r is found in one of these cells ($r \in m_{q'}(s, o)$), check if
   - m_q is defined for this very cell ($s \in S_q \wedge o \in O_q$),
   - r was already contained in this very cell in m_q ($r \in m_q \dots$).
4. Recursiv. proceed with 2. for any possible successor state q'' of q'

Theorem 1 [Harrison] Ingeneral, HRU safety is not decidable.

Theorem 2 [Harrison] For mono-operational models, HRU safety is decidable.

- Insights into the operational principles modeled by HRU models
- Demonstrates a method to prove safety property for a particular, given model
- → „Proofs teach us how to build things so nothing more needs to be proven." (W. E. Kühnhauser)

a mono-operational HRU model → exactly one primitive for each operation in the STS

Proof of Theorem - Proof Sketch

1. Find an upper bound for the length of all input sequences with different effects on the protection state w.r.t. safety If such can be found: $\exists$ a finite number of input sequences with different effects
2. All these inputs can be tested whether they violate safety. This test terminates because:
   - each input sequence is finite
   - there is only a finite number of relevant sequences

→ safety is decidable

Proof: Transform $\sigma_1 \dots \sigma_n$ into shorter sequences

1. Remove all input operations that contain delete or destroy primitives (no absence, only presence of rights is checked).
2. Prepend the sequence with an initial create subject s_{init} operation.
3. Prune the last create subject s operation and substitute each following reference to s with s_{init}. Repeat until all create subject operations are removed, except from the initial create subject s_{init}.
4. Same as steps 2 and 3 for objects.
5. Remove all redundant enter operations.

init	...
...	create subject s_{init};
...	create object o_{init}
create subject $x2$;	-
create object $x5$;	-
enter r1 into $m(x2, x5)$;	enter r1 into $m(s_{init}, o_{init})$;
enter r2 into $m(x2, x5)$;	enter r2 into $m(s_{init}, o_{init})$;
create subject $x7$;	-
delete r1 from $m(x2, x5)$;	-
destroy subject $x2$;	-
enter r1 into $m(x7, x5)$;	-

Conclusions from these Theorems (Dilemma)

- General (unrestricted) HRU models
  - have strong expressiveness → can model a broad range of AC policies
  - are hard to analyze: algorithms and tools for safety analysis
- Mono-operational HRU models
  - have weak expressiveness → goes as far as uselessness (only create files)
  - efficient to analyze: algorithms and tools for safety analysis
  - → are always guaranteed to terminate
  - → are straight-forward to design

(A) Restricted Model Variants

Static HRU Models

- Static: no create primitives allowed
- safe(q,r) decidable, but NP-complete problem
- Applications: (static) real-time systems, closed embedded systems

Monotonous Mono-conditional HRU Models

- Monotonous (MHRU): no delete or destroy primitives
- Mono-conditional: at most one clause in conditions part
- safe(q,r) efficiently decidable
- Applications: Archiving/logging systems (nothing is ever deleted)

Finite Subject Set

- $\forall q \in Q, \exists n \in N : |S_q| \leq n$
- $safe(q, r)$ decidable, but high computational complexity

Fixed STS

- All STS commands are fixed, match particular application domain (e.g. OS access control) → no model reusability
- For Lipton and Snyder [1977]: $safe(q, r)$ decidable in linear time

Strong Type System

- Special model to generalize HRU: Typed Access Matrix (TAM)
- $safe(q, r)$ decidable in polynomial time for ternary, acyclic, monotonous variants
- high, though not unrestricted expressiveness in practice

(B) Heuristic Analysis Methods

- Restricted model variants often too weak for real-world apps
- General HRU models: safety property cannot be guaranteed
- → get a piece from both: Heuristically guided safety estimation

Idea:

- State-space exploration by model simulation
- Task of heuristic: generating input sequences (educated guessing)

Outline: Two-phase-algorithm to analyze $safe(q_0, r)$:

1. Static phase: knowledge from model to make ,,good" decisions
   → Runtime: polynomial in model size ($q_0 + STS$)
2. Simulation phase: The automaton is implemented and, starting with q_0, fed with inputs $\sigma =< op, x >$
   → For each σ, the heuristic has to decide:
   - which operation op to use
   - which vector of arguments x to pass
   - which q_i to use from the states in Q known so far
   - Termination: As soon as $\sigma(q_i, \sigma)$ violates $safe(q_0, r)$.

Goal: Iteratively build up the Q for a model to falsify safety by example (finding a violating but possible protection state).
Termination: only a semi-decidable problem here. It can be guaranteed that a model is unsafe if we terminate. We cannot ever prove the opposite → safety undecidability

- Find typical errors in security policies: Guide designers, who might know there's something wrong but not what and why
- Increase understanding of unsafety origins: By building clever heuristics, we started to understand how we might design specialized HRU models that are safety-decidable yet practically (re-)usable

The Typed-Access-Matrix Model (TAM)

- Adopted from HRU: subjects, objects, ACM, automaton
- New: leverage the principle of strong typing (like programming)
- → safety decidability properties relate to type-based restrictions
- Foundation of a TAM model is an HRU model $\langle Q, \sum, \delta, q_0, R\rangle$, where $Q = 2^S \times 2^O \times M$
- However: $S \subseteq O$, i. e.:
  - all subjects can also act as objects (=targets of an access)
  - useful for modeling e.g. delegation
  - objects in $O\backslash S$: pure objects
- Each $o \in O$ has a type from a type set T assigned through a mapping $type : O \rightarrow T$
- An HRU model is a special case of a TAM model:
  - $T = \{tSubject, tObject\}$
  - $\forall s \in S : type(s) = tSubject; \forall o \in O\backslash S : type(o) = tObject$

TAM Security Model A TAM model is a deterministic automaton $\langle Q, \sum, \delta, q_0, T, R\rangle$ where
- $Q = 2^S \times 2^O \times TYPE \times M$ is the state space where S and O are subjects set and objects set as in HRU, where $S \subseteq O$, $TYPE = \{type|type : O \rightarrow T\}$ is a set of possible type functions, M is the set of possible $ACMs$ as in HRU,
- $\sum = OP \times X$ is the (finite) input alphabet where OP is a set of operations as in HRU, $X = O^k$ is a set of k-dimensional vectors of arguments (objects) of these operations,
- $\delta : Q \times \sum \rightarrow Q$ is the state transition function,
- $q_0 \in Q$ is the initial state,
- T is a static (finite) set of types,
- R is a (finite) set of access rights.

Convenience Notation where

- $q \in Q$ is implicit
- $op, r_1, \dots, r_m, s_1, \dots, s_m, o_1, \dots, o_m$ as before
- $t_1, \dots, t_k$ are argument types
- $p_1, \dots, p_n$ are TAM-specific primitives

TAM-specific

- Implicit Add-on: Type Checking where t_i are the types of the arguments $x_i, 1 \leq i \leq k$.
- Primitives:
  - enter r into m(x_s,x_o)
  - delete r from m(x_s,x_o)
  - create subject x_s of type t_s
  - create object x_o of type t_o
  - destroy subject x_s
  - destroy object x_o
- Observation: S and O are dynamic (as in HRU), thus $type : O \rightarrow T$ must be dynamic too (cf. definition of Q in TAM).

TAM Example: The ORCON Policy

- Creator/owner of a document should permanently retain controlover its accesses
- Neither direct nor indirect (by copying) right proliferation
- Application scenarios: Digital rights management, confidential sharing
- Solution with TAM: A confined subject type that can never execute any operation other than reading

Model Behavior (STS): The State Transition Scheme

- $createOrconObject(s_1 : s, o_1 : co)$
- $grantCRead(s1 : s, s2 : s, o1 : co)$
- $useCRead(s_1 : s, o_1 : co, s_2 : cs)$
- $revokeCRead(s_1 : s, s_2 : s, o_1 : co)$
- $destroyOrconObject(s_1 : s, o_1 : co)$ (destroy conf. object)
- $revokeRead(s_1 : s, s_2 : cs, o_1 : co)$ (destroy conf. subject)
- $finishOrconRead(s_1 : s, s_2 : cs)$ (destroy conf. subject)
- Owner retains full control over
- Use of her confined objects by third parties → transitive right revocation
- Subjects using these objects → destruction of these subjects
- Subjects using such objects are confined: cannot forward read information

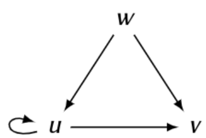
TAM Safety Decidability

- General TAM models → safety not decidable
- **MTAM** monotonous TAM models; STS without delete or destroy primitives → safety decidable if mono-conditional only
- **AMTAM** acyclic MTAM models → safety decidable but not efficiently (NP-hard problem)
- **TAMTAM** ternary AMTAM models; each STS command requires max. 3 arguments → provably same computational power and thus expressive power as AMTAM; safety decidable in polynomial time

Acyclic TAM Models

Parent- and Child-Types For any operation op with arguments $\langle x_1, t_1\rangle, \dots, \langle x_k, t_k\rangle$ in an STS of a TAM model, it holds that $t_i, 1 \leq i \leq k$
- is a child type in op if one of its primitives creates a subject or object x_i of type t_i,
- is a parent type in op if none of its primitives creates a subject or object x_i of type t_i.

Type Creation Graph The type creation graph $TCG = \langle T, E = T \times T\rangle$ for the STS of a TAM model is a directed graph with vertex set T and an $edge\langle u, v\rangle \in E$ iff $\exists op \in OP : u$ is a parent type in $op \wedge v$ is a child type in op.



Note: In bar u is both a parent type (because of s_1) and a child type (because of s_2) → hence the loop edge.

Safety Decidability: We call a TAM model acyclic, iff its TCG is acyclic.

Theorem 5 Safety of a ternary, acyclic, monotonous TAM model (TAMTAM) is decidable in polynomial time in the size of m_0.

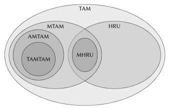
Crucial property acyclic, intuitively:

- Evolution of the system (protection state transitions) checks both rights in the ACM as well as argument types
- TCG is acyclic ⇒ ∃ a finite sequence of possible state transitions after which no input tuple with argument types, that were not already considered before, can be found
- One may prove that an algorithm, which tries to expandall possible different follow-up states from q_0, may terminate after this finite sequence

Expressive Power of TAMTAM

- MTAM: obviously same expressive power as monotonic HRU
  - no transfer of rights: ,,take r ... in turn grant r to ..."
  - no countdown rights: ,,r can only be used n times"
- ORCON: allow to ignore non-monotonic command s from STS since they only remove rights and are reversible
- AMTAM: most MTAM STS may be re-written as acyclic
- TAMTAM: expressive power equivalent to AMTAM

IBAC Model Comparison: family of IBAC models to describe different ranges of security policies they are able to express



Roles-based Access Control Models (RBAC)

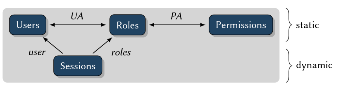
Solving Scalability and Abstraction results in smaller modeling effort results in smaller chance of human errors made in the process

- Improved scalability and manageability
- application-oriented semantic: $roles \approx functions$ in organizations
- Models include smart abstraction: roles
- AC rules are specified based on roles instead of identities
- Users, roles, and rights for executing operations
- Access rules are based on roles of users → on assignments
- improved Scalability
- improved Application-oriented model abstractions
- Standardization (RBAC96) → tool-support
- Limited dynamic analyses w.r.t. automaton-based models

Basic RBAC model An $RBAC_0$ model is a tuple $\langle U, R, P, S, UA, PA, user, roles\rangle$ where
- U is a set of user identifiers,
- R is a set of role identifiers,
- P is a set of permission identifiers,
- S is a set of session identifiers,
- $UA \subseteq U \times R$ is a many-to-many user-role-relation,
- $PA \subseteq P \times R$ is a many-to-many permission-role-relation,
- $user : S \to U$ is a total function mapping sessions to users,
- $roles : S \to 2^R$ is a total function mapping sessions to sets of roles such that $\forall s \in S : r \in roles(s) \Rightarrow \langle user(s), r\rangle \in UA$.

Interpretation

- Users U model people: actual humans that operate the AC system
- Roles R model functions, originate from workflows/responsibility
- Permissions P model rights for any particular access
- user-role-relation $UA \subseteq U \times R$ defines which roles are available to users at any given time → assumed during runtime before usable
- permission-role-relation $PA \subseteq P \times R$
- UA and PA describe static policy rules
- Sessions S describe dynamic assignments of roles → a session $s \in S$ models when a user is logged in
  - $S \to U$ associates a session with its (,,owning") user
  - $S \to 2^R$ associates a session with the set of roles currently assumed by that user (active roles)



RBAC Access Control Function

- access rules have to be defined for operations on objects
- implicitly defined through P → made explicit: $P \subseteq O \times OP$ is a set of permission tuples $\langle o, op\rangle$ where
  - $o \in O$ is an object from a set of object identifiers,
  - $op \in OP$ is an operation from a set of operation identifiers.
- We may now define the ACF for $RBAC_0$

$RBAC_0$ **ACF** $f_{RBAC_0} : U \times O \times OP \to \{true, false\}$ where $\begin{cases} true, & \exists r \in R, s \in S : u = user(s) \wedge r \in roles(s) \wedge \langle\langle o, op\rangle, r\rangle \in PA \\ false, & \text{otherwise}\end{cases}$

RBAC96 Model Family

In practice, organizations have more requirements that need to be expressed in their security policy

- $RBAC_1 = RBAC_0 + hierarchies$
- $RBAC_2 = RBAC_0 + constraints$
- $RBAC_3 = RBAC_0 + RBAC_1 + RBAC_2$

RBAC 1: Role Hierarchies

Roles often overlap

1. disjoint permissions for roles → any user X must always have Y assigned and activated for any of her workflows → role assignment redundancy
2. overlapping permissions: $\forall p \in P : \langle p, proDev\rangle \in PA \Rightarrow \langle p, proManager\rangle \in PA$ → any permission must be assigned to two different roles → role definition redundancy
3. Two types of redundancy → undermines scalability goal of RBAC

Solution: Role hierarchy → Eliminates role definition redundancy through permissions inheritance
Modeling Role Hierarchies: Lattice here: $\langle R, \leq\rangle$

- Hierarchy expressed through dominance relation: $r_1 \leq r_2 \Leftrightarrow r_2$ inherits any permissions from r_1
- **Reflexivity** any role consists of its own permissions
- **Antisymmetry** no two different roles may mutually inherit their respective permissions
- **Transitivity** permissions may be inherited indirectly

$RBAC_1$ **Security Model** An $RBAC_1$ model is a tuple $\langle U, R, P, S, UA, PA, user, roles, RH\rangle$ where
- U, R, P, S, UA, PA and $user$ are defined as for $RBAC_0$,
- $RH \subseteq R \times R$ is a partial order that represents a role hierarchy where $\langle r, r'\rangle \in RH \Leftrightarrow r \leq r'$ such that $\langle R, \leq\rangle$ is a lattice,
- roles is defined as for $RBAC_0$, while additionally holds: $\forall r, r' \in R, \exists s \in S : r \leq r' \wedge r' \in roles(s) \Rightarrow r \in roles(s)$.

RBAC 2: Constraints

roles in org. often more restricted

- Certain roles may not be active at the same time (session) for any user
- Certain roles may not be together assigned to any user

→ separation of duty (SoD)

- While SoD constraints are a more fine-grained type of security requirements to avoid mission-critical risks, there are other types represented by RBAC constraints

Constraint Types

- **Separation of duty** mutually exclusive roles
- **Quantitative constraints** maximum number of roles per user
- **Temporal constraints** time/date/week/... of role activation
- **Factual constraints** assigning or activating roles for specific permissions causally depends on any roles for a certain

Modeling Constraints Idea

- $RBAC_2 : \langle U, R, P, S, UA, PA, user, roles, RE\rangle$
- $RBAC_3 : \langle U, R, P, S, UA, PA, user, roles, RH, RE\rangle$
- where RE is a set of logical expressions over the other model components (such as $UA, PA, user, roles$)

Attribute-based Access Control Models (ABAC)

- Scalability and manageability
- Application-oriented model abstractions
- Model semantics meet functional requirements of open systems:
  - user IDs, INode IDs, ...only available locally
  - roles limited to specific organizational structure

→ application-specific context of access: attributes of subjects and objects (e. g. age, location, trust level, ...)

Idea: Generalizing the principle of indirection already known from RBAC

- IBAC: no indirection between subjects and objects
- RBAC: indirection via roles assigned to subjects
- ABAC: indirection via arbitrary attributes assigned to sub-/objects
- Attributes model application-specific properties of the system entities involved in any access
  - Age, location, trustworthiness of a application/user/...
  - Size, creation time, access classification of resource/...
  - Risk quantification involved with these subjects and objects

ABAC Access Control Function

- $f_{IBAC} : S \times O \times OP \to \{true, false\}$
- $f_{RBAC} : U \times O \times OP \to \{true, false\}$
- $f_{ABAC} : S \times O \times OP \to \{true, false\}$

→ Evaluates attribute values for $\langle s, o, op\rangle$

ABAC Security Model

- Note: There is no such thing (yet) like a standard ABAC model
- Instead: Many highly specialized, application-specific models.
- Here: minimal common formalism, based on Servos and Osborn

ABAC Security Model An ABAC security model is a tuple $\langle S, O, AS, AO, attS, attO, OP, AAR\rangle$ where
- S is a set of subject identifiers and O is a set of object identifiers,
- $A_S = V_S^1 \times \dots \times V_S^n$ is a set of subject attributes, where each attribute is an n-tuple of values from arbitrary domains V_S^i, $1 \leq i \leq n$,
- $A_O = V_O^1 \times \dots \times V_O^m$ is a corresponding set of object attributes, based on values from arbitrary domains V_O^j, $1 \leq j \leq m$,
- $att_S : S \to A_S$ is the subject attribute assignment function,
- $att_O : O \to A_O$ is the object attribute assignment function,
- OP is a set of operation identifiers,
- $AAR \subseteq \Phi \times OP$ is the authorization relation.

Interpretation

- Active and passive entities are modeled by S and O, respectively
- Attributes in AS, AO are index-referenced tuples of values, which are specific to some property of subjects V_S^i (e.g. age) or of objects V_O^j (e. g. PEGI rating)
- Attributes are assigned to subjects and objects via att_S, att_O
- Access control rules w.r.t. the execution of operations in OP are modeled by the AAR relation → determines ACF!
- AAR is based on a set of first-order logic predicates Φ: $\Phi = \{\phi_1(x_{s1}, x_{o1}), \phi_2(x_{s2}, x_{o2}), \dots\}$. Each $\phi_i \in \Phi$ is a binary predicate, where x_{si} is a subject variable and x_{oi} is an object variable.

ABAC Access Control Function (ACF)
- $f_{ABAC} : S \times O \times OP \to \{true, false\}$ where
- $f_{ABAC}(s, o, op) = \begin{cases} true, & \exists\langle\phi, op\rangle \in AAR : \phi(s, o) = true \\ false, & \text{otherwise}\end{cases}$.
- We call ϕ an authorization predicate for op.

Information Flow Models (IF)

Abstraction level of AC Models: rules about subjects accessing objects. Goal: Problem-oriented definition of policy rules for scenarios based on information flows(rather than access rights)

- Information flows and read/write operations are isomorphic
  - s has read permission o ⇔ information flow from o to s
  - s has write permission o ⇔ information flow from s to o

→ Implementation by standard AC mechanisms!

Analysis of Information Flow Models

- IF Transitivity → goal: covert information flows
- IF Antisymmetry → goal: redundancy

Denning Security Model A Denning information flow model is a tuple $\langle S, O, L, cl, \bigoplus\rangle$ where
- S is a set of subjects,
- O is a set of objects,
- $L = \langle C, \leq\rangle$ is a lattice where
  - C is a set of classes,
  - $\leq$ is a dominance relation where c $\leq d \Leftrightarrow$ information may flow from c to d,
- $cl : S \cup O \rightarrow C$ is a classification function, and
- $\bigoplus : C \times C \rightarrow C$ is a reclassification function.

Interpretation

- Subject set S models active entities, which information flows originate from
- Object set O models passive entities, which may receive information flows
- Classes set C used to label entities with identical information flow properties
- Classification function cl assigns a class to each entity
- Reclassification function $\bigoplus$ determines which class an entity is assigned after receiving certain a information flow

This enables

- precisely define all information flows valid for a given policy
- define analysis goals for an IF model w.r.t.
  - **Correctness** ∃ covert information flows?
  - **Redundancy** ∃ sets of subjects and objects with equivalent information contents?
- implement a model through an automatically generated, isomorphic ACM

Multilevel Security (MLS)

- Introducing a hierarchy of information flow classes: levels of trust
- Subjects and objects are classified:
  - Subjects w.r.t. their trust worthiness
  - Objects w.r.t. their criticality
- Within this hierarchy, information may flow only in one direction → „secure" according to these levels!
- → ∃ MLS models for different security goals!

Modeling Confidentiality Levels

- Class set: levels of confidentiality e.g. $C = \{public, conf, secret\}$
- Dominance relation: hierarchy between confidentiality levels e.g. $\{public \leq confidential, confidential \leq secret\}$
- Classification of subjects and objects: $cl : S \cup O \rightarrow C$ e.g. $cl(BulletinBoard) = public, cl(Timetable) = confidential$
- In contrast due Denning $\leq$ in MLS models is a total order

The Bell-LaPadula Model

MLS-Model for Preserving Information Confidentiality. Incorporates impacts on model design ...

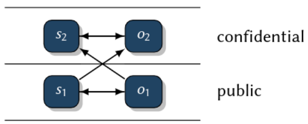
- from the application domain: hierarchy of trust
- from the Denning model: information flow and lattices
- from the MLS models: information flow hierarchy
- from the HRU model:
  - Modeling dynamic behavior: state machine and STS
  - Model implementation: ACM

→ application-oriented model engineering by composition of known abstractions

BLP Security Model A BLP model is a deterministic automaton $\langle S, O, L, Q, \sum, \sigma, q_0, R\rangle$ where
- S and O are (static) subject and object sets,
- $L = \langle C, \leq\rangle$ is a (static) lattice consisting of
  - the classes set C,
  - the dominance relation $\leq$,
- $Q = M \times CL$ is the state space where
  - $M = \{m|m : S \times O \rightarrow 2^R\}$ is the set of possible ACMs,
  - $CL = \{cl|cl : S \cup O \rightarrow C\}$ is a set of functions that classify entities in $S \cup O$,
- $\sum$ is the input alphabet,
- $\sigma : Q \times \sum \rightarrow Q$ is the state transition function,
- $q_0 \in Q$ is the initial state,
- $R = \{read, write\}$ is the set of access rights.

Interpretation

- $S, O, M, \sum, \sigma, q_0, R$: same as HRU
- L: models confidentiality hierarchy
- cl: models classification meta-information about sub-/objects
- $Q = M \times CL$ models dynamic protection states; includes
  - rights in the ACM,
  - classification of subjects/objects,
  - not: S and O (different to HRU)
- Commands in the STS may therefore
  - change rights in the ACM,
  - reclassify subjects and objects.



⟺

m	o_1	o_2
s_1	{r, w}	{w}
s_2	{r}	{r, w}

- L is an application-oriented abstraction
  - Supports convenient for model specification
  - Supports easy model correctness analysis
  - → easy to specify and to analyze
- m can be directly implemented by standard OS/DBIS access control mechanisms (ACLs, Capabilities) → easy to implement
- m is determined (= restricted) by L and cl, not vice-versa
- L and cl control m

BLP Security

Read-Security Rule A BLP model state $\langle m, cl\rangle$ is called read-secure iff $\forall s \in S, o \in O : read \in m(s, o) \Rightarrow cl(o) \leq cl(s)$.

Write-Security Rule A BLP model state $\langle m, cl\rangle$ is called write-secure iff $\forall s \in S, o \in O : write \in m(s, o) \Rightarrow cl(s) \leq cl(o)$.

State Security A BLP model state is called secure iff it is both read- and write-secure.

Model Security A BLP model with initial state q_0 is called secure iff
1. q_0 is secure and
2. each state reachable from q_0 by a finite input sequence is secure.

BLP Basic Security Theorem A BLP model $\langle S, O, L, Q, \sum, \sigma, q_0, R\rangle$ is secure iff both of the following holds:
1. q_0 is secure
2. σ is build such that for each state q reachable from q_0 by a finite input sequence, where $q = \langle m, cl\rangle$ and $q' = \sigma(q, \delta) = m', cl', \forall s \in S, o \in O, \delta \in \sum$ the following holds:

- Read-security conformity:
  - read $\notin m(s, o) \wedge read \in m'(s, o) \Rightarrow cl'(o) \leq cl'(s)$
  - read $\in m(s, o) \wedge \neg(cl'(o) \leq cl'(s)) \Rightarrow read \notin m'(s, o)$
- Write-security conformity:
  - write $\notin m(s, o) \wedge write \in m'(s, o) \Rightarrow cl'(s) \leq cl'(o)$
  - write $\in m(s, o) \wedge \neg(cl'(s) \leq cl'(o)) \Rightarrow write \notin m'(s, o)$

Idea: Encode an additional, more fine-grained type of access restriction in the ACM → compartments

- Comp: set of compartments
- $co : S \cup O \rightarrow 2^{Comp}$: assigns a set of compartments to an entity as an (additional) attribute
- Refined state security rules:
  - $\langle m, cl, co\rangle$ is read-secure $\Leftrightarrow \forall s \in S, o \in O : read \in m(s, o) \Rightarrow cl(o) \leq cl(s) \wedge co(o) \subseteq co(s)$
  - $\langle m, cl, co\rangle$ is write-secure $\Leftrightarrow \forall s \in S, o \in O : write \in m(s, o) \Rightarrow cl(s) \leq cl(o) \wedge co(o) \subseteq co(s)$
- BLP with compartments: $\langle S, O, L, Comp, Q_{co}, \sigma, \delta, q_0\rangle$ where $Q_{co} = M \times CL \times CO$ and $CO = \{co|co : S \cup O \rightarrow 2^{Comp}\}$

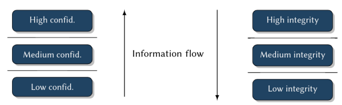
BLP Model Summary

- Application-oriented modeling → hierarchical information flow
- Scalability → attributes: trust levels
- Modeling dynamic behavior → automaton with STS
- Correctness guarantees (analysis of)
  - consistency: BLP security, BST
  - completeness of IF: IFG path finding
  - presence of unintended IF: IFG path finding
  - unwanted redundancy: IF cycles
  - safety properties: decidable
- Implementation
  - ACM is a standard AC mechanism in contemporary implementation platforms (cf. prev. slide)
  - Contemporary standard OSs need this: do not support mechanisms for entity classification, arbitrary STSs
  - new platforms: SELinux, TrustedBSD, PostgreSQL, ...
- Is an example of a hybrid model: IF + AC + ABAC

The Biba Model

BLP upside down

- BLP → preserves confidentiality
- Biba → preserves integrity



OS Example: file/process/... created is classified → cannot violate integrity of objects

Non-interference Models (NI)

Problems: Covert Channels & Damage Range (Attack Perimeter)

Covert Channel Channels not intended for information transfer at all, such as the service program's effect on the system load.

- AC policies (ACM, HRU, TAM, RBAC, ABAC): colluding malware agents, escalation of common privileges
  - Process 1: only read permissions on user files
  - Process 2: only permission to create an internet socket
  - both: communication via covert channel
- MLS policies (Denning, BLP, Biba): indirect information flow exploitation (can never prohibitany possible transitive IF ...)
  - Test for existence of a file
  - Volume control on smartphones

Idea of NI models

- higher level of abstraction
- which domains should be isolated based on their mutual impact
- → Easier policy modeling
- → More difficult implementation → higher degree of abstraction
- Needed: isolation of services, restricted cross-domain interactions
- → Guarantee of total/limited non-interference between domains

NI Security Policies

Security domains & Cross-domain actions

Non-Interference Two domains do not interfere with each other iff no action in one domain can be observed by the other.

NI Security Model An NI model is a det. automaton $\langle Q, \sigma, \delta, \lambda, q_0, D, A, dom, \approx_{NI}, Out\rangle$ where

- Q is the set of (abstract) states,
- $\sigma = A$ is the input alphabet where A is the set of (abstract) actions,
- $\delta : Q \times \sigma \to Q$ is the state transition function,
- $\lambda : Q \times \sigma \to Out$ is the output function,
- $q_0 \in Q$ is the initial state,
- D is a set of domains,
- $dom : A \to 2^D$ is adomain function that completely defines the set of domains affected by an action,
- $\approx_{NI} \subseteq D \times D$ is a non-interference relation,
- Out is a set of (abstract) outputs.

NI Security Model is also called Goguen/Meseguer-Model.

BLP written as an NI Model

- BLP Rules:
  - write in class public may affect public and confidential
  - write in class confidential may only affect confidential
- NI Model:
  - $D = \{d_{pub}, d_{conf}\}$
  - write in d_{conf} does not affect d_{pub}, so $d_{conf} \approx_{NI} d_{pub}$
  - $A = \{writeInPub, writeInConf\}$
  - $dom(writeInPub) = \{d_{pub}, d_{conf}\}$
  - $dom(writeInConf) = \{d_{conf}\}$

NI Model Analysis

Purge Function Let $aa^* \in A^*$ be a sequence of actions consisting of a single action $a \in A \cup \{\epsilon\}$ followed by a sequence $a^* \in A^*$, where ϵ denotes an empty sequence. Let $D' \in 2^D$ be any set of domains. Then, purge: $A^* \times 2^D \to A^*$ computes a subsequence of aa^* by removing such actions without an observable effect on any element of D' :

- $purge(aa^*, D') = \begin{cases} a \circ purge(a^*, D'), \exists d_a \in dom(a), d' \in D' : d_a \approx_I d' \\ purge(a^*, D'), \quad \text{otherwise} \end{cases}$
- $purge(\epsilon, D') = \epsilon$

where $\approx_I$ is the complement of $\approx_{NI}$: $d_1 \approx_I d_2 \Leftrightarrow \neg(d_1 \approx_{NI} d_2)$.

NI Security For a state $q \in Q$ of an NI model $\langle Q, \sigma, \delta, \lambda, q_0, D, A, dom, \approx_{NI}, Out\rangle$, the predicate ni-secure (q) holds iff $\forall a \in A, \forall a^* \in A^* : \lambda(\delta^*(q, a^*), a) = \lambda(\delta^*(q, purge(a^*, dom(a))), a)$.

Interpretation

1. Running an NI model on $\langle q, a^*\rangle$ yields $q' = \delta^*(q, a^*)$.
2. Running the model on the purged input sequence so that it contains only actions that, according to $\approx_{NI}$, actually have impact on $dom(a)$ yields $q'_{clean} = \delta^*(q, purge(a^*, dom(a)))$
3. If $\forall a \in A : \lambda(q', a) = \lambda(q'_{clean}, a)$, than the model is called NI-secure w.r.t. q($ni-secure(q)$).

Comparison to HRU and IF Models

HRU Models

- Policies describe rules that control subjects accessing objects
- Analysis goal: right proliferation
- Covert channels analysis: only based on model implementation

IF Models

- Policies describe rules about legal information flows
- Analysis goals: indirect IFs, redundancy, inner consistency
- Covert channel analysis: same as HRU

NI Models

- Rules about mutual interference between domains
- Analysis goal: consistency of $\approx_{NI}$ and dom
- Implementation needs rigorous domain isolation (e.g. object encryption is not sufficient) → expensive
- State of the Art w.r.t. isolation completeness

Hybrid Models

Chinese-Wall Policies (CW)

e.g. for consulting companies

Policy goal: No flow of (insider) information between competing clients

- Composition of
  - Discretionary IBAC components
  - Mandatory ABAC components
- by real demands: iterative refinements of a model over time
  - Brewer-Nash model
  - Information flow model
  - Attribute-based model
- Application areas: consulting, cloud computing

The Brewer-Nash Model

tailored towards Chinese Wall

Model Abstractions

- Consultants represented by subjects
- Client companies represented by objects
- Modeling of competition by conflict classes: two different clients are competitors ⇔ their objects belong to the same class
- No information flow between competing objects → a „wall" separating any two objects from the same conflict class
- Additional ACM for refined management settings of access permissions

Representation of Conflict Classes

- Client company data: object set O
- Competition: conflict relation $C \subseteq O \times O : \langle o, o'\rangle \in C \Leftrightarrow o$ and o' belong to competing companies
- object attribute $att_O : O \to 2^O$, such that $att_O(o) = \{o' \in O | \langle o, o'\rangle \in C\}$

Representation of a Consultant's History

- Consultants: subject set S
- History $H \subseteq S \times O : \langle s, o\rangle \in H \Leftrightarrow s$ has previously consulted o
- subject attribute $att_S : S \to 2^O$, such that $att_S(s) = \{o \in O | \langle s, o\rangle \in H\}$

Brewer-Nash Security Model is a deterministic $automaton \langle S, O, Q, \sigma, \delta, q_0, R\rangle$ where

- S and O sets of subjects (consultants) and objects (company data),
- $Q = M \times 2^C \times 2^H$ is the state space where
  - $M = \{m | m : S \times O \to 2^R\}$ is the set of possible ACMs,
  - $C \subseteq O \times O$ is the conflict relation: $\langle o, o'\rangle \in C \Leftrightarrow o$ and o' are competitors,
  - $H \subseteq S \times O$ is the history relation: $\langle s, o\rangle \in H \Leftrightarrow s$ has previously consulted o,
- $\sigma = OP \times X$ is the input alphabet where
  - $OP = \{read, write\}$ is a set of operations,
  - $X = S \times O$ is the set of arguments of these operations,
- $\delta : Q \times \sigma \to Q$ is the state transition function,
- $q_0 \in Q$ is the initial state,
- $R = \{read, write\}$ is the set of access rights.

Brewer-Nash STS

- Read (similar to HRU notation) command read(s,o)::=if read $\in$ m(s,o) $\wedge \forall \langle o', o\rangle \in C : \langle s, o'\rangle \notin H$ then $H := H \cup \{\langle s, o\rangle\}$ fi
- Write command write(s,o)::=if write $\in$ m(s,o) $\wedge \forall o' \in O : o' \neq o \Rightarrow \langle s, o'\rangle \notin H$ then $H := H \cup \{\langle s, o\rangle\}$ fi

→ modifications in m to enable fine-grained rights management.
Restrictiveness:

- Write Command: s is allowed to write $o \Leftrightarrow write \in m(s, o) \wedge \forall o' \in O : o' \neq o \Rightarrow \langle s, o'\rangle \notin H$
- → s must never have previously consulted any other client
- any consultant is stuck with her client on first read access

Brewer-Nash Model

- Initial State q_0, $H_0 = \varnothing$
- m_0: consultant assignments to clients, issued by management
- C_0: according to real-life competition

Secure State $\forall o, o' \in O, s \in S : \langle s, o\rangle \in H_q \wedge \langle s, o'\rangle \in H_q \Rightarrow \langle o, o'\rangle \notin C_q$
Corollary: $\forall o, o' \in O, s \in S : \langle o, o'\rangle \in C_q \wedge \langle s, o\rangle \in H_q \Rightarrow \langle s, o'\rangle \notin H_q$

Secure Brewer-Nash Model Similar to „secure BLP model".

- difference: trusting humans vs. trusting software agents
- → Write-rule applied not to humans, but to software agents
- → Subject set S models consultant's subjects in a group model
  - all processes of one consultant form a group

The Least-Restrictive-CW Model

Restrictiveness of Brewer-Nash Model:

- If $\langle o_i, o_k\rangle \in C$: no transitive information flow $o_i \rightarrow o_j \rightarrow o_k$
- more restrictive than necessary: $o_j \rightarrow o_k$ and later $o_i \rightarrow o_j$ fine
- Criticality of an IF depends on existence of earlier flows.

Idea LR-CW: Include time as a model abstraction!

- $\forall s \in S, o \in O$: remember, which information has flown to entity
- → subject-/object-specific history, ≈attributes („lables")

Least-Restrictive CW model of the CW policy is a deterministic $automaton \langle S, O, F, \zeta, Q, \sigma, \delta, q_0\rangle$ where

- S and O are sets of subjects (consultants) and data objects,
- F is the set of client companies,
- $\zeta : O \rightarrow F$ („zeta") function mapping each object to its company,
- $Q = 2^C \times 2^H$ is the state space where
  - $C \subseteq F \times F$ is the conflict relation: $\langle f, f'\rangle \in C \Leftrightarrow f$ and f' are competitors,
  - $H = \{Z_e \subseteq F | e \in S \cup O\}$ is the history set: $f \in Z_e \Leftrightarrow e$ contains information about $f(Z_e$ is the „history label" of $e)$,
- $\sigma = OP \times X$ is the input alphabet where
  - $OP = \{read, write\}$ is the set of operations,
  - $X = S \times O$ is the set of arguments of these operations,
- $\delta : Q \times \sigma \rightarrow Q$ is the state transition function,
- $q_0 \in Q$ is the initial state

- reading: requires that no conflicting information is accumulated in the subject potentially increases the amount of information in the subject
- writing: requires that no conflicting information is accumulated in the object potentially increases the amount of information in the object

Model Achievements

- Applicability: more writes allowed in comparison to Brewer-Nash
- Paid for with
  - Need to store individual attributes of all entities (history)
  - Need of write permissions on earlier actions of subjects
- More extensions:
  - Operations to modify conflict relation
  - Operations to create/destroy entities

An MLS Model for Chinese-Wall Policies

Conflict relation is

- non-reflexive: no company is a competitor of itself
- symmetric: competition is always mutual
- not necessarily transitive: any company might belong to more than one conflict class → Cannot be modeled by a lattice

Idea: Labeling of entities

- Class of an entity (subject or object) reflects information it carries
- Consultant reclassified whenever a company data object is read
- → Classes and labels:
- Class set of a lattice $C = \{DB, Citi, Shell, Esso\}$
- Entity label: vector of information already present in each business branch

Practical Security Engineering

Goal: Design of new, application-specific models

- Identify common components → generic model core
- Core specialization
- Core extension
- Glue between model components

Model Engineering

- Core **model** (Common Model Core) → $\langle Q, \sum, \delta, q_0\rangle$
- Core **specialization**
  - HRU: $Q = 2^S \times 2^O \times M$
  - RBAC: $Q = 2^U \times 2^{UA} \times 2^S \times USER \times ROLES$
  - DABAC: $Q = 2^S \times 2^O \times M \times ATT$
  - TAM: $Q = 2^S \times 2^O \times TYPE \times M$
  - BLP: $Q = M \times CL$
  - NI: -
- Core **extension**
  - HRU: R
  - $DRBAC_0$:R, P, PA
  - DABAC: A
  - TAM: T, R
  - BLP: S, O, L, R
  - NI: $\lambda, D, A, dom, =_{NI}, Out$
- Component **glue**
  - TAM: State transition scheme (types)
  - DABAC: State transition scheme (matrix, predicates)
  - Brewer/Nash Chinese Wall model: „$\wedge$" (simple)
  - BLP (much more complex, rules restrict m by L and cl)

Model Specification

Policy Implementation (Language) to bridge the gap between

- Abstractions of security models (sets, relations, ...)
- Abstractions of implementation platforms (security mechanisms such as ACLs, krypto-algorithms,...)
- Foundation for Code verification or even more convenient: Automated code generation

Abstraction level: Step stone between model and security mechanisms

- → More concrete than models
- → More abstract than programming languages
- Expressive power: Domain-specific for representing security models only
- → Necessary: adequate language paradigms
- → Sufficient: not more than necessary (no dead weight)

Domains

- Model domain, e.g. AC/IF/NI models (TAM, RBAC, ABAC)
- Implementation domain (OS, Middleware, Applications)

DYNAMO: A Dynamic-Model-Specification Language

formerly known as „CorPS: Core-based Policy Specification Language"
Language Domain: RBAC models
Language Paradigms: Abstractions of (D)RBAC models

- Users, roles, permissions, sessions
- State transition scheme (STS)

Language Features: Re-usability and inheritance

- Base Classes: Model family (e.g. $DRBAC_0, DRBAC_1, \ldots$)
- Policy Classes: Inherit definitions from Base Classes

DYNAMO compiler: Translates specification into XML and C++ Classes

SELinux Policy Language

Language Domain I/R/A-BAC models, IF(NI) models
Model Domain: BAC, MLS, NI
Application Domain: OS-level security policies
Implementation Domain: Operating systems access control
Language paradigms

- OS Abstractions: Users, processes, files, directories, sockets, ...
- model paradigms: Users, rights, roles, types, attributes, ...

Tools

- Specification: Policy creating and validation
- Policy compiler: Translates policy specifications
- Security server: Policy runtime environment in OS kernel security architecture
- LSM hooks: Support policy enforcement in OS kernel security architecture

Technology

- Policy compiler → translates specifications into loadable binaries
- Security architecture → implementation of Flask architecture

Basic Language Concepts

- Definition of types (a.k.a. „domains")
- Labeling of subjects (e.g. processes) with „domains" → $passwd_t$
- Labeling of objects (e.g. files, sockets) with „types" → $shadow_t$
- AC: defined by permissions between pairs of types
- Dynamic interactions: transitions between domains

Policy Rules

- Grant permissions: allow rules
- Typical domains: $user_t$, bin_t, $passwd_t$, $insmod_t$, $tomCat_t$, ...
- Classes: OS abstractions (process, file, socket, ...)
- Permissions: read, write, execute, getattr, signal, transition, ...

The Model Behind: 3 Mappings

- Classification $cl : S \cup O \rightarrow$ C where C = $\{process, file, dir, \ldots\}$
- Types $type : S \cup O \rightarrow$ T where T = $\{user_t, passwd_t, bin_t, \ldots\}$
- Access Control Function (Type Enforcement) $te : T \times T \times C \rightarrow 2^R$
- $\rightarrow ACM : T \times (T \times C) \rightarrow 2^R$

Idea only: SELinux RBAC

Users and Roles

- User ID assigned on login
- RBAC rules confine type associations „Only users in role $doctor_r$ may transit to domain $edit - epr_t$"
- → fine-grained domain transitions
- → Attributes in SELinux-style RBAC: User ID, Role ID
- Specification → Tool → Binary → Security Server

Model abstractions

- TE: MAC rules based on types
- ABAC:MAC rules based on attributes
- RBAC: MAC rules based on roles
- Additionally: BLP-style MLS

Other Policy Specification Languages

- XACML (eXtensibleAccess Control Markup Language)
- NGAC (Next Generation Access Control Language)
- SEAL (Label-based AC policies)
- Ponder (Event-based condition/action rules)
- GrapPS (Graphical Policy Specification Language)
- GemRBAC (Role-based AC models)
- PTaCL (Policy re-use by composition)

Security Mechanisms

Security Models Implicitly Assume

- Integrity of model implementation
  - Model state
  - Authorization scheme
- Integrity of model operations call
  - Parameters of authorization scheme ops
  - Completeness and total mediation of their invocation
- AC, IF: no covert chanels
- NI: Rigorous domain isolation

→ job of the ,,Trusted Computing Base" (TCB) of an IT system

Trusted Computing Base (TCB) The set of functions of an IT system that are necessary and sufficient for implementing its security properties → Isolation, Policy Enforcement, Authentication ...

Security Architecture The part of a system's architecture that implement its TCB → Security policies, Security Server (PDP) and PEPs, authentication components, ...

Security Mechanisms Algorithms and data structures for implementing functions of a TCB → Isolation mechanisms, communication mechanisms, authentication mechanisms, ...

→ TCB - runtime environment for security policies

- (some) TCB functions are integrated in today's commodity OSes
  - Isolation
  - Subject/object authentication
- Complex models additionally require implementation of
  - Authorization schemes
  - Roles, lattices, attributes
  - → stronger concepts and mechanisms
  - OS level: Security Server (SELinux, OpenSolaris)
  - Middleware level: Policy Objects (CORBA, DBMSs)
  - Application level: user level reference monitors (Flume), user level policy servers (SELinux)

Security mechanisms: A Visit in the Zoo: ...

- In OSes
  - Authenticity
    - Of subjects: login
    - Of objects: object management, e.g. file systems
  - Confidentiality and integrity: Access control lists
- In middleware layer (DBMSs, distributed systems)
  - Authentication server (Kerberos AS) or protocols (LDAP)
  - Authorization: Ticket server (Kerberos TGS)
- In libraries and utilities
  - Confidentiality, integrity, authenticity
    - Cryptographic algorithms
    - Certificate management for PKIs
    - Isolation (Sandboxing)

Authorization

Lampson, HRU, RBAC, ABAC, BLP, CW → ACMs

Access Control Lists und Capability Lists

Lampson's ACM: Sets S, O, R and ACM $m : S \times O \to 2^R$
Properties of an ACM

- Large (e.g. ,,normal" file server: $|m| >> 1$ TByte)
- Sparsely populated
- Subject and object identifications in OSes generally are
  - Not numerical
  - Not consecutive
- Rows and columns are created and destroyed dynamically

Idea: Distributed ACM Implementation

1. Split matrix into vectors; Column/Row vectors
2. Attach vectors to subjects resp. objects

- Column vectors
  - Describe every existing right wrt. an object
  - vector associated to object, part of object's metadata
  - → Access control lists (ACLs)
- Row vectors
  - Describe every existing right wrt. a subject
  - Associated to its subject, part of subject's metadata
  - → capability lists

ACLs

- Associated to exactly one object
- Describes every existing right wrt. object by a set of tuples
- Implemented e.g. as list, table, bitmap
- Part of object's metadata (generally located in inode)

Create and Delete an ACL

- Together with creation and deletion of an object
- Initial rights are create operation parameters → discretionary access control
- Initial rights issued by third party→ mandatory access control

Modify an ACL

- Add or remove tuples (subject identification, right set)
- Owner has right to modify ACL → discretionary access control
- Third party has right to modify ACL → mandatory access control
- Right to modify ACL is part of ACL → universal

Check Rights

- Whenever an object is accessed
- Search granting tuple in ACL

Negative Rights

- Dominate positive rights
- represented by tuples (subject identification, negative rights set)
- Rights of subject: difference of positive and negative rights

Example: ACLs in Unix

- 3 elements per list list
- 3 elements per right set

→ 9 bits coded in 16-bit-word (PDP 11, 1972)

	read	write	exec
owner	y	y	n
group	y	n	n
others	n	n	n

Operations on Capability Lists

Create and Delete

- Together with creation and deletion of a subject
- Initial rights same as parent → inherited
- Constraints by
  - Parent → discretionary access control
  - Capability → mandatory access control

Modification: Add or remove tuples (object identification, right set)
Passing on Capabilities, options:

- Emission and call-back by capability owner → discretionary access control
- Emission and call-back by third party → mandatory access control
- Emission and call-back controlled by capability itself → universal

δs in Administration

ACLs: Located near objects → finding all rights of a subject expensive
Example BLP: re-classification of a subject → update every ACL with rights of this subject
Group models; e.g.

- BLP: subjects with same classification
- Unix: subjects belonging to project staff

Role models (role: set of rights); e.g. set of rights wrt. objects with same classification

δs in Distributed Systems

- No encapsulation of subject ids/ACLs in single trustworthy OS
- No encapsulation of cap. lists in a single trustworthy OS kernel
  - Authentication and management on subject's system
  - Transfer via open communication system
  - Checking of capabilities and subject ids on object's system

Vulnerabilities and Counteractions

- Subject's system may fake subject ids
- Consequence: Reliable subject authentication required → authentication architectures (e.g. Kerberos)
- Non-trustworthy subject systems modify capabilities
  - → cryptographic sealing of capabilities such that
  - Issuer can be determined
  - Modification can be detected
  - sealing e.g. by digital signatures
- Non-trustworthy subject systems pass capabilities to third parties or are copied by third parties while in transit → personalized
- Exploit stolen capabilities by forging subject id
  - → cryptographically sealed personalized capabilities
  - → reliable subject authentication required
  - → authentication architectures

Expressive Power of ACLs and Capability Lists

- Efficient data structures for implementing ACMs/ACFs
- Located in OSs, middleware, DBMSe, application systems
- Correctness, tamperproofness, total S/O interaction mediation vital for enforcing access control → implementation by strong architectural principles
- Assume reliable authentication of subjects and objects → support by further security mechanisms
- Are too weak to implement complex security policies
- Not sufficient for implementing more complex security policies → Authorization schemes

Interceptors

Policy implementation by algorithms instead of lists

- Tamperproof runtime environments for security policies
- In total control of subject/object interactions (Observation, Modification, Prevention)

General Architectural Principle: Separation of

- (Replaceable) strategies
- (Strategy-independent) mechanisms

Applied to Interceptors → 2 Parts

- Runtime environment for security policies (strategies)
  - often called ,,policy decision point" (PDP)
- Interception points (mechanisms)
  - often called ,,policy enforcement points" (PEP)

Summary

- RTE for security policies in policy-controlled systems
  - SELinux: ,,Policy Server"
  - CORBA: ,,Policy Objects"
- Architecture: separation of responsibilities
- Strategic component State and authorization scheme
- Policy enforcement: total policy entities interaction mediation
- Generality: implement a broad scope of policies (computable)
  - → rules based on checking digital signatures
  - → interceptor checks/implements encryption

Cryptographic Security Mechanisms

Encryption: Transformation of a plaintext into a ciphertext

- 2 functions encrypt, decrypt
- 2 keys k1, k2
- $text = decrypt_{k2}(encrypt_{k1}(text))$ or simply
- $text = \{\{text\}_{k1}\}_{k2}$ (if encryption function is obvious)
- Symmetric schemes (secret key): one single key: $k1 = k2$
- Asymmetric schemes (public key): two different keys: $K1 \neq K2$

Kerkhoff's Principle

1. Encryption functions (algorithms) are publicly known
   - → many experts look at it
   - → quality advantage assumed
2. Keys are secret
   - → encryption security depends on
   - Properties of algorithms
   - Confidentiality of keys

Symmetric Encryption Schemes

- Encryption and decryption with same key
- → security based on keeping key secret
- Example: shift letters of a ciphertext forward by K positions

Application Examples

1. Confidentiality of Communication (Assumptions)
   - Sender and receiver share key k , which has to be established before communication, Authentically, Confidentially
   - Nobody else must know $k(secretkey)$
2. Authentication: client to server (by shared secret key)
   - Each client shares an individual and secret key k_{client} with server
   - Server and clients keep key secret
   - Server reliably generates a nonce (=never sent once before)
3. Sealing of Documents, e.g. Capabilities
   - 1 key owner → owner may
     - seal document
     - check whether seal is sound
   - Group of key owners → each group membermay
     - Seal document
     - Check whether seal was impressed by group member
     - → nobody in this group can prove it was him or not
   - Outside the group → nobody can do any of these things

Algorithms: Block and Stream Ciphers

- Block cipher
  - Decompose plaintext into blocks of equal size (e.g. 64 bits)
  - Encrypt each block
  - e.g. Data Encryption Standard (DES) obsolete since 1998
  - e.g. Advanced Encryption Standard (AES) (128bits length)
- Stream cipher
  - Encrypt each digit of a plaintext stream by a cipher digit stream (e.g. by XOR)
  - Cipher digit stream: pseudo-random digit stream

Asymmetric Encryption Schemes

- → key pair $(k1, k2) = (k_{pub}, k_{sec})$ where
- $decrypt_{k_{sec}}(encrypt_{k_{pub}}(text)) = text$
- Conditio sine qua non: Secret key not computable from public key

Application Examples

1. Confidentiality of Communication (compare symmetric encryption schemes)
   - Sender shares no secret with receiver → No trust between sender and receiver necessary
   - Sender must know public key of receiver → public-key-Infrastructures (PKIs) containing key certificates
2. Authentication: using public key
   - Each client owns an individual key pair (k_{pub}, k_{sec})
   - Server knows public keys of clients (PKI)
   - Clients are not disclosing secret key
   - Server reliably generates nonces
   - Properties
     - Client and server share no secrets
     - No key exchange before communication
     - No mutual trust required
     - But: sender must know public key of receiver
     - → PKIs
3. Sealing of Documents, compare sealing using secret keys
   - ∃ just 1 owner of secret key → only she may seal contract
   - Knowing her public key,
     - → everybody can check contract's authenticity
     - → everybody can prove that she was the sealer
     - → repudiability: digital signatures

Consequence of Symmetric vs. Asymmetric Encryption

- **Sym** shared key, integrity and authenticity can be checked only by key holders → message authentication codes (MACs)
- **Asym** integrity and authenticity can be checked by anyone holding public key (only holder of secret key could have encrypted the checksum) → digital signatures

Key Distribution for Symmetric Schemes

- Asymmetric encryption is expensive
- Key pairs generation (High computational costs, trust needed)
- Public Key Infrastructures needed for publishing public keys
- → Use asymmetric key for establishing communication
- Use symmetric encryption for communication

RSA Cryptosystem (Rivest/Shamir/Adleman)

Attractive because $encrypt = decrypt$ → universal:

1. Confidentiality
2. Integrity and authenticity (non repudiability, digital signatures)

For $n \in \mathbb{N}$ we search 2 primes p and q such that $n = p * q$

- → hard problem because for factorization, prime numbers are needed
- There are many of them, approx. $7 * 10^{151}$
- Finding them is extremely expensive: Sieve of Eratosthenes
- Optimization: Atkin's Sieve, $O(n^{1/2+O(1)})$
- Until today, no polynomial factorization algorithm is known
- Until today, nobody proved that such algorithm cannot exist...

Precautions in PKIs: Prepare for fast exchange of cryptosystem

Cryptographic Hash Functions

Discover violation of integrity of data, so that integrity of information is maintained.

- Checksum generation by cryptographic hash functions
- Checksum encryption
- Integrity check by
  - Generating a new checksum
  - Decryption of encrypted checksum
  - Comparison of both values

Method of Operation: Map data of arbitrary length to checksum of fixed length such that $Text1 \neq Text2 \Rightarrow hash(Text1) \neq hash(Text2)$ with high probability

- 160 - Bit checksums: RIPEMD-160 (obsolete since 2015)
- Secure Hash Algorithm (SHA-1, published NIST 1993)
- Larger Checksums: SHA-256, SHA-384, SHA-512
- 128-Bit: Message Digest (MD5 (1992)) (no longer approved)
- MD5: belongs to IPsec algorithm group, used also in SSL

Digital Signatures

- assert author of a document (signer) → Authenticity
- discover modifications after signing → Integrity → non repudiability

Approach

- Create signature
  - Integrity: create checksum → cryptographic hash function
  - Authenticity: encrypt checksum → use private key of signer
- Check signature
  - Decrypt checksum using public key of signer
  - Compare result with newly created checksum

Cryptographic Attacks

Ciphertext Only Attacks (weakest assumptions)

- Known: ciphertext CT
- Wanted: plaintext T, Ke, Kd, algorithm
- Typical assumptions
  - CT was completely generated by one Ke
  - Known algorithm
  - Observation of packet sequences in networks
  - Listening into password-based authentication

Known Plaintext Attacks

- Known: T and CT (respectively parts thereof)
- Wanted: Ke, Kd, algorithm
- Listening into challenge/response protocols
  - Server → Client: nonce
  - Client → Server: $\{nonce\}_{Ke}$
- countermeasure often: Client → Server:$\{nonce + Time\}_{Ke}$

Chosen Plaintext Attacks

- Known: T and CT where T can be chosen by attacker
- Wanted: Ke, Kd (algorithm often known)
- Authentication in challenge/response protocols
  - Attacker (malicious server) tries to find client's private key
  - sends tailored nonces
- Authentication by chosen passwords
  - Attacker tries to find login password
  - Generates passwords & compare encryptions with pw DB

Chosen Ciphertext Attacks

- Known: T, CT and Kd, CT can be chosen, T can be computed
- wanted: Ke → successful attacks allow forging digital signatures
- Attack by
  - (within limits) Servers while authenticating clients
  - (within limits) Observers of such authentications
  - In a PK cryptosystem: Everybody knowing Kd

Goals of Cryptographic Algorithms

- To provide security properties such as
  - Integrity, confidentiality, non-repudiability
  - Of communication
  - Of resources such as files, documents, program code
- Especially: implement assumptions made by security models like
  - Authenticity, integrity, confidentiality of
  - Model entities (subjects, objects, roles, attributes)
  - Model implementations

Beware: Many Pitfalls!

- Weaknesses of mathematical foundations → unproved assumptions
- Weaknesses of algorithms → cryptographic attacks
- Weaknesses of key generation → e.g. weak prime numbers
- Weaknesses of mechanism use → co-existence of mechanisms

Identification and Authentication

To reliably identify people, systems,. . . .
Approaches: Proof of identity by

- By proving knowledge of simple secret → passwords
- By biophysicproperties → biometrics
- By proving knowledge of simple secret → cryptographic protocols

Passwords

- Used For: Authentication of humans to IT systems
- Verified Item: Knowledge of simple secret
- Convenient
- Easy to guess / compute (RainbowCrack: $104 * 10^9$ hash/second)
  - → password generators
  - → password checkers (min. 8 chars, . . .)
- Problem of careless handling (password on post-it)
- Input can easily be observed (see EC PINs)

→ Confidential communication with authenticating system

Biometrics

- Used For: Authentication of humans to IT systems
- Verified Items: Individual properties like voice, hand/retina, finger
- Verification: By comparing probe with reference pattern
- Pros: (prospectively) Difficult to counterfeit
  - Convenient, no secrets to remember, cannot be lost
  - Difficult to intentionally pass on
- Contras: Fundamental technical problems
  - Comparison methods with reference fuzzy techniques
  - False Non-match Rate: authorized people are rejected
  - False Match Rate: not authorized people are accepted
  - Susceptible environmental conditions (noise, dirt, fractured)
  - Social Barriers, Acceptance
- Fundamental weaknesses in distributed systems → Secure communication to authenticating system required (personal data)
- Reference probes are personal data → Data Protection Act
- Reaction time on security incidents → Passwords, smartcards can be exchanged easily

Cryptographic Protocols

SmartCards

- Used For: Authentication of humans to IT systems
- Verified Item: Knowledge of complex secret
  - Secret part of asymmetric key pair
  - Symmetric key
- Verification
  - Challenge/response protocols
  - Goal: Proof that secret is known
  - Contrary to password authentication, no secret exposure

Vehicle for Humans: SmartCards

- Small Computing Devices encompassing Processor(s), RAM, Persistent memory, Communication interfaces
- What They Do
  - Store and keep complex secrets (keys)
  - Run cryptographic algorithms
    - Response to challenges in challenge/response protocols
    - Encrypt incoming nonces
  - Launch challenges to authenticate other principals
    - Generate nonces, verify response

Properties

- no secret is exposed
  - → no trust in authenticating system required
  - → no trust in network required
- Besides authentication other features possible → digital signatures, credit card, parking card . . .
- Weak verification of card right to use card (PIN, password) → some cards have finger print readers
- Power supply for contactless cards

Authentication Protocols

- Used For: Authentication between IT systems
- Method: challenge/response-scheme
- Based on symmetric & asymmetric key

The 2 fundamental Scenarios

1. After one single authentication, Alice wants to use all servers in a distributed system of an organization.
2. Alice wants authentic and confidential communication with Bob. Authentication Server serves session keys to Bob and Alice

Needham-Schroeder Authentication Protocol (for secret keys)

- establish authentic and confidential communication between 2

→ confidentiality, integrity, authenticity

1. Authentication of Alice to Bob → Bob knows other end is Alice
2. Authentication of Bob to Alice → Alice knows other end is Bob
3. Establish fresh secret: a shared symmetric session key

Fundamental

- Common trust in same authentication server
- Client-specific secret keys (K_{AS}, K_{BS})

Message Semantics

1. $A \rightarrow S : A, B, N_A$: A requests session key for B from S
2. $S \rightarrow A : \{N_A, B, K_{AB}, \{K_{AB}, A\}_{KBS}\}_{KAS}$: S responds encrypted with K_{AS} such that only A is able to understand
   - nonce proves that 2. is a reply to 1. (fresh)
   - session key K_{AB}
   - ticket for B; encryption proves K_{AB} was generated by S
3. $A \rightarrow B : \{K_{AB}, A\}_{KBS}$: A ticket to B; encryption as challenge
4. $B \rightarrow A : \{N_B\}_{KAB}$: B decrypts ticket & verifies if A knows K_{AB}
5. $A \rightarrow B : \{N_B - 1\}_{KAB}$: A proves by using K_{AB} that he was the sender of 3. (response)
   - Authentication of A to B: only A can decrypt 2.
   - Authentication of B to A: only B can decrypt 3.
   - A and B now also share a secret session key

Authentication Servers

- Common trust in server by all principals → closed user group
- Server shares individual secret with each principal (sym key)

Needham-Schroeder Authentication Protocol for public keys

- establish authentic and confidential communication between Principals
- Premise: Trust
  - Individually in issuer of certificate (certification authority)
  - → much weaker than secret key based authentication
- Message Semantics
  1. $A \rightarrow S : A, B$: A requests public key of B
  2. $S \rightarrow A : \{PK_B, B\}_{SK_S}$: S sends certificate; A knows public key of CA
  3. $A \rightarrow B : \{N_A, A\}_{PK_B}$: A sends challenge to B
  4. $B \rightarrow S : B, A$: B requests public key of A
  5. $S \rightarrow B : \{PK_A, A\}_{SK_S}$: S responds (see 2.)
  6. $B \rightarrow A : \{N_A, N_B\}_{PK_A}$: B proves it is B and challenges A
  7. $A \rightarrow B : \{N_B\}_{PK_B}$: A replies and proves it is A

- Authentication of A to B: 6. together with 7.
- Authentication of B to A: 3. together with 6.
- From where key certificates are obtained is irrelevant

Certificate Servers: Basis of Authentication

- Key certificates
  - Digitally signed mappings (name ↔ public key)
  - Issued by certification authorities (CA)
- Certificate servers
  - Manage certificate data base
  - Need not be trustworthy

δs between Secret Key and Public Key Authentication

- Secret Key Authentication
  - Requires common trust in AS, a-priori key exchange and mutual trust in keeping session key secret
  - Allows for message authentication codes
  - Require online AS
  - accumulation of secrets at AS → dangerous, server always online
  - n keys for authenticating n principals
  - $O(n^2)$ session keys for n communicating parties
- Public Key Authentication
  - Requires knowledge of public keys → PKIs
  - Allows for digital signatures
  - Allow for local chaching of certificates
  - n keys for authenticating n principals
  - $O(n)$ keys for n communicating parties if PKs are used
  - $O(n^2)$ key for n comm. parties if session keys are used
  - Certificate management: PKIs, CAs, data bases, ...

Security Architectures

Security architectures have been around for a long time ...

- Architecture Components (Buildings, walls, windows,...)
- Architecture (Component arrangement and interaction)
- Build a stronghold such that security policies can be enforced
  - Presence of necessary components/mechanisms
  - Totality of interaction control („mediation")
  - Tamperproofness
  - → architecture design principles

Check your trust in

- Completeness of access mediation (and its verification!)
- Policy tamperproofness(and its verification!)
- TCB correctness (and its verification!)

Problem Areas PDPs/PEPs are

- Scattered among many OS components → Problem of architecture
- Not robust
  - Not isolated from errors within the entire OS
  - Especially in dynamically loaded OS modules
  - → Problem of security architecture implementation
- OSes/Middleware/Applications are big
- Only a small set of their functions logically belongs to the TCB
- → architecture design such that TCB functions are collected
  - not bypassable (total access mediation),
  - isolated (tamperproofness),
  - trustworthy (verifiable correctness) core
  - → architecture such that these properties are enforced

Architecture Design Principles

Definitions of fundamental security architecture design principles

- Complete
- Tamperproof
- Verifiably correct
- control of all security-relevant actions in a system

The Reference Monitor Principles

There exists an architecture component that is

- RM1 Involved in any subject/object interaction → total mediation property
- RM2 Well-isolated from the rest of the systems → tamperproofness
- RM3 Small and well-structured enough to analyze correctness by formal methods → verifiability

architecture component built along these: „Reference Monitor"

- 1 PDP (policy implementation)
- many PEPs (interceptors, policy enforcement)

Reference Monitor

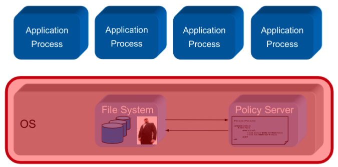
- Core component of a TCB
- Typically encloses
  - Security policy implementation(s) (PDP)
    - ∗ Model state (e.g. ACM, subject set, entity attributes)
    - ∗ Model behavioral logic (e.g.authorization scheme)
  - Enforcement mechanisms: PEPs
- Typically excludes (due to complexity and size, RM 3)
  - Authentication
  - Cryptographic mechanisms
  - Sometimes also model state (e.g.ACLs)

Consequences of (RM 3) for TCBs

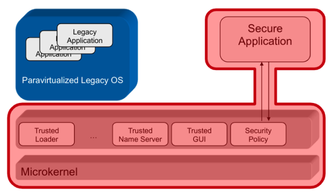
- Few functions → small size (LoC)
- Simple functions → low complexity
- Strong isolation
- Precisely known perimeter

Implementation Layers

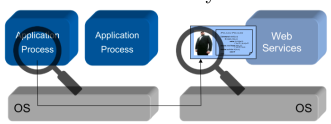
Monolithic OS Kernel



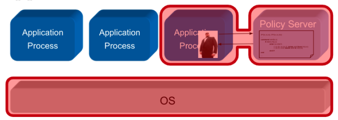
Microkernel Architecture (Nizza)



Middleware-level Policy



Application



- Numerous rather weak implementations in Middleware, Applications...
- Stronger approaches in Microkernel OSes, Security-focused OS

Nizza

- RM1 - RM3 (Especially: Small TCB)
- Maintain functionality of
  - Contemporary legacy OSes
  - Legacy Applications („legacy" = unmodified for security)

Concepts/Reference monitor principles:

- Separation of OS, Applications into security-critical vs. non-critical components → precise identification of (minimal) TCB
- Maintain functionality → Paravirtualization of standard legacy OS

OS View

- Trustworthy microkernel
- Trustworthy basic services
- Not trustworthy (paravirtualized) legacy OS

Application View

- Vulnerability increases with growing complexity → reduce vulnerability of security-critical code by
- Software functionality separation
- Isolation of functional domains
- Example: Email Client
  - Non-critical: reading/composing/sending emails
  - Critical: signing emails (email-client ↔ Enigmail Signer)
- Code size of TCB reduced by 2 orders of magnitude
- Functionality of legacy OSes and applications preserved
- (Moderate) performance penalties
- Paravirtualization of legacy OS
- Decomposition of trusted applications

Security Enhanced Linux (SELinux)

- State-of-the-art OS
- State-of-the-art security paradigms
- → Policy-controlled (Linux) (Security-aware) OS kernel

Security Policies in SELinux

- Implementation by new OS abstractions
- Somewhat comparable to „process" abstraction
- Specification of a...
  - process is a program: algorithm implemented in formal language
  - security policy is a security model: rule set in formal language
- Runtime environment (RTE) of a ...
  - process is OSprocess management → RTE for application-level programs
  - security policy is OS security Server → RTE for kernel-level policies

SELinux Architecture

- Policy-aware Security Server (policy decision point, PDP) → Policy RTE in kernel's protection domain
- Interceptors (policy enforcement points, PEPs) → Total interaction control in object managers

Implementation Concepts

- Reference Monitor Principles
  - Total mediation of security-relevant interactions → placement of PEPs: Integration into object managers
  - Tamperproofness of policy implementation → placement of PDP: Integration into kernel
- Policy Support
  - Authenticity of entities: Unique subject/object identifiers
  - Policy-specific entity attributes (type, role, MLS label)
- Problem in Linux,

- Subject identifiers (PIDs) or object identifiers (i-node numbers) are
  - neither unique
  - nor are of uniform type
- → security identifier (SID)
- Policy-specific subject/object attributes (type, role) are not part of subject/object metadata → security context
- → Approach: Extensions of process/file/socket...-management

Authenticity of Entities

- Object managers help: implement injective mapping SEO → SID
  - SID created by security server
  - Mapping of SIDs to objects by object managers

Entity Attributes

- sec. policy implements injective mapping SID → security context
- sec. contexts creation according to policy-specific labeling rules
- Entry in SID → security context mapping table

Security Context contains

- Standard entity attributes such as user ID, Role, Type
- Policy-specific entity attributes such as Confidentiality/clearance level (e.g. MLS label)
- is implemented as a text string with policy-dependent format

Problem: Security contexts of persistent Entities

- Policies not aware of persistency of entities → persistency of security contexts is job of object managers
- Layout of object metadata is file system standard → security contexts cannot be integrated in i-nodes (their implementation: policy-independent)

Solution

- Persistent objects additionally have persistent SID : „PSID"
- OMs map these to SID
- 3 invisible storage areas in persistent memory implementing
  - Security context of file system itself (label)
  - Bijective mapping: inode → PSID
  - Bijective mapping: PSID → security context

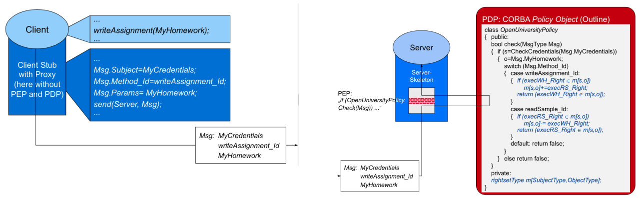
Access Vector Cache(AVC)

- Located in object managers (user level) resp. in Security Server (kernel level)
- Caches access decisions

RM Evaluation of SELinux

- Compliance with Reference Monitor Principles
- Total Mediation Property (placement of PEPs) done manually
- Tamperproofness of Policy Implementation
  - Fundamental problem in monolithic software architectures
  - → TCB implementation vulnerable from entire OS kernel code
  - Security server, All object managers, Memory management,...
  - It can be done: Nizza
- Verifiability
  - Size and complexity of policy → analysis tools
  - Policy's RTE claim to be universal
  - Completeness of PEPs
  - Policy isolation

Security Architectures of Distributed Systems CORBA



Kerberos

Distributed Authentication and Authorization Architecture with closed user groups(→ static sets of subjects)

- Distributed system run by single organization
- Workstations and Servers
- 2 Kerberos servers
  - Authentication Server (AS)
  - Authorization Server (TGS)
- Authentication Server (AS)
  - Authenticates users; Based on key shared between user and AS. Result: authenticator (electronic ID card)
  - Authorizes use of TGS. Based on key shared between AS and TGS. Result: ticket (capability) for TGS
- Ticket Granting Server (TGS): Issues tickets for all servers
  - Based on key shared between TGS and respective server
  - Result: ticket(s) for server(s)
- Kerberos database
  - Contains for each user and server a mapping $\langle user, server \rangle$ → authentication key
  - Used by AS
  - Is multiply replicated (availability, scalability)

Typical Use Case

1. Authentication, then request for TGS ticket
2. Authenticator, TGS-Ticket
3. Request for further server tickets
4. Server tickets
5. Service request: Servers decide based on

Inside Kerberos Tickets

- Tickets issued by Ticket Granting Server
- Specify right of one client to use one server (capability)
- Limited lifetime (to make cryptographic attacks difficult)
  - balance between secure and convenient
  - Short: inconvenient but more secure (if stolen soon expires)
  - Long: insecure but more convenient (no frequent renewal)
- Can be used multiply while valid
- Are sealed by TGS with key of server

Provisions against Misuse

- Tampering by client to fabricate rights for different server → guarantee of integrity by MAC using $K_{TGS/Server}$
- Use by third party intercepting ticket → personalization by Name and network address of client together with Limited lifetime &Authenticator of client

Authenticators

- Proof of identity of client to server
- Created using $SessionKey_{Client/Server}$
  - → can be created and checked only by
  - Client (without help by AS, client knows session key)
  - Server
  - TGS (trusted)
- Can be used exactly once → prevent replay attacks by checking freshness

Kerberos Login

1. Alice tells her name
2. Alice's workstation requests authentication
3. The AS
   - Create fresh timestamp
   - Create session key for Alice communication with the TGS
   - Create Alice ticket for TGS and encrypt it with $K_{AS/TGS}$
   - Encrypts everything with $K_{Alice/AS}$ (only Alice can read the session key and the TGS-Ticket)
4. Alice's workstation
   - $TGS, Timestamp, SessionKey_{Alice/TGS}, Ticket_{Alice/TGS}$
   - Requests Alice's password
   - Get $K_{Alice/AS}$ from password using cryptographic hash
   - Uses it to decrypt above message from AS

- Result: Alice's workstation has
  - Session key for TGS session: $SessionKey_{Alice/TGS}$
  - Ticket for TGS: $Ticket_{Alice/TGS}$
  - The means to create an authenticator

Using a Server

Authentication (bidirectional)

1. Authentication of Client (to server)
   - (Assumption) Alice has session key
   - (Assumption) Alice has server ticket

   (a) Alice assembles authenticator A_{Alice}
   (b) Alice sends $Ticket_{Alice/Server}, A_{Alice}$ to Server
   (c) Server decrypts ticket and thus gets session key; thus it can decrypt A_{Alice} and check
      - Freshness
      - Compliance of names in ticket and authenticator
      - Origin of message and network address in authenticator
2. Authentication of Servers (to client)
   - send $\{Timestamp + 1\}_{SessionKey_{Alice/Server}}$ to Alice
   - only by principal that knows $SessionKey_{Alice/Server}$
   - only by server that can extract the session key from the ticket

Getting a Ticket for a Server

- Are valid for a pair $\langle client, server \rangle$
- Are issued (but for TGS-Ticket itself) only by TGS
- Ticket request to TGS: $(server, TGS_{ticket}, authenticator)$

TGS:

- Checks $Ticket_{Client/TGS}$ and $authenticator$
- Generates $SessionKey_{Client/Server}$ for client & server
- Generates $Ticket_{Client/Server}$
- Encrypts both using shared session key $\{Server, SessionKey_{Client/Server}, Ticket_{Client/Server}\}_{SessionKey_{Client/TGS}}$

Identity-based access control models (IBAC)

- ACF: $f_{ABAC} : S \times O \times OP \rightarrow \{true, false\}$
- ACM: $m : S \times O \rightarrow 2^{OP}$ so $\forall s \in S, \forall o \in O : op \in m(s,o) \Leftrightarrow f(s,o,op)$.

HRU (Harrison-Ruzzo-Ullman) Safety

iff, beginning with q, there is no sequence of commands that enters r in an ACM cell where it did not exist in q

TAM Security Model $\langle Q, \sum, \delta, q_0, T, R\rangle$

- $Q = 2^S \times 2^O \times TYPE \times M$ where S and O are subjects set and objects set as in HRU, where $S \subseteq O$, $TYPE = \{type | type : O \rightarrow T\}$ is a set of possible type functions, M is the set of possible $ACMs$ as in HRU,
- $\sum = OP \times X$ is the input alphabet where OP is a set of operations as in HRU, $X = O^k$ is a set of k-dim. vectors of arguments of these operations,
- $\delta : Q \times \sum \rightarrow Q$ is the state transition function,
- $q_0 \in Q$ is the initial state,
- T is a static (finite) set of types,
- R is a (finite) set of access rights.

Theorem 5

Safety of a ternary, acyclic, monotonous TAM model (TAMTAM) is decidable in polynomial time

Role-based access control models (RBAC)

$RBAC_0 = \langle U, R, P, S, UA, PA, user, roles\rangle$ where

- U is a set of user identifiers,
- R is a set of role identifiers,
- P is a set of permission identifiers,
- S is a set of session identifiers,
- $UA \subseteq U \times R$ is a many-to-many user-role-relation,
- $PA \subseteq P \times R$ is a m:m permission-role-relation,
- $user : S \rightarrow U$ function mapping sessions to users,
- $roles : S \rightarrow 2^R$ function mapping sessions to roles

- $RBAC_1 = RBAC_0 + hierarchies$
- $RBAC_2 = RBAC_0 + constraints$
- $RBAC_3 = RBAC_0 + RBAC_1 + RBAC_2$

- **Separation of duty** mutually exclusive roles
- **Quantitative constraints** max. number of use
- **Temporal constraints** time/date/week/...

- $RBAC_1 : \langle U, R, P, S, UA, PA, user, roles, RH\rangle$
- $RBAC_2 : \langle U, R, P, S, UA, PA, user, roles, RE\rangle$
- $RBAC_3 : \langle U, R, P, S, UA, PA, user, roles, RH, RE\rangle$
- $RH \subseteq R \times R$ is a partial order of hierarchy
- RE logical expressions over the other model components

Attribute-based access control models (ABAC)

Model $\langle S, O, AS, AO, attS, attO, OP, AAR\rangle$ where

- S, O are subject/object identifiers,
- $A_S = V_S^1 \times \cdots \times V_S^n$ is a set of subject attributes,
- $A_O = V_O^1 \times \cdots \times V_O^m$ set of object attributes,
- $att_S : S \rightarrow A_S$ subject attribute assignment func.,
- $att_O : O \rightarrow A_O$ object attribute assignment function,
- OP is a set of operation identifiers,
- $AAR \subseteq \Phi \times OP$ is the authorization relation.

Denning Security Model

a tuple $\langle S, O, L, cl, \bigoplus\rangle$ where

- $L = \langle C, \leq\rangle$ is a lattice where C is a set of classes, $\leq$ is a dominance relation
- $cl : S \cup O \rightarrow C$ is a classification function,
- $\bigoplus : C \times C \rightarrow C$ is a reclassification function.

BLP Security Model

automaton $\langle S, O, L, Q, \sum, \sigma, q_0, R\rangle$ where

- S and O are (static) subject and object sets,
- $L = \langle C, \leq\rangle$ is a (static) lattice consisting of classes set C, the dominance relation $\leq$,
- $Q = M \times CL$ is the state space where
  - $M = \{m | m : S \times O \rightarrow 2^R\}$ set of possible ACMs,
  - $CL = \{cl | cl : S \cup O \rightarrow C\}$ classify entities in $S \cup O$,
- $\sum$ is the input alphabet,
- $\sigma : Q \times \sum \rightarrow Q$ is the state transition function,
- $q_0 \in Q$ is the initial state,
- $R = \{read, write\}$ is the set of access rights.

NI Security Model

automaton $\langle Q, \sigma, \delta, \lambda, q_0, D, A, dom, \approx_{NI}, Out\rangle$ where

- Q is the set of (abstract) states,
- $\sigma = A$ is the input alphabet, A set of actions,
- $\delta : Q \times \sigma \rightarrow Q$ is the state transition function,
- $\lambda : Q \times \sigma \rightarrow Out$ is the output function,
- $q_0 \in Q$ is the initial state,
- D is a set of domains,
- $dom : A \rightarrow 2^D$ is adomain function that completely defines the set of domains affected by an action,
- $\approx_{NI} \subseteq D \times D$ is a non-interference relation,
- Out is a set of (abstract) outputs.

NI Security Model is also called Goguen/Meseguer-Model.

Brewer-Nash Security Model

is a deterministic $automaton\langle S, O, Q, \sigma, \delta, q_0, R\rangle$ where

- S, O sets of subjects (consultants) & objects (data),
- $Q = M \times 2^C \times 2^H$ is the state space where
  - $M = \{m | m : S \times O \rightarrow 2^R\}$ set of possible ACMs,
  - $C \subseteq O \times O$ is the conflict relation,
  - $H \subseteq S \times O$ is the history relation,
- $\sigma = OP \times X$ is the input alphabet where
  - $OP = \{read, write\}$ is a set of operations,
  - $X = S \times O$ arguments of these operations,
- $\delta : Q \times \sigma \rightarrow Q$ is the state transition function,
- $q_0 \in Q$ is the initial state,
- $R = \{read, write\}$ is the set of access rights.

Least-Restrictive CW model

automaton $\langle S, O, F, \zeta, Q, \sigma, \delta, q_0\rangle$ where

- S and O are sets of subjects and data objects,
- F is the set of client companies,
- $\zeta : O \rightarrow F$ mapping each object to its company,
- $Q = 2^C \times 2^H$ is the state space where
  - $C \subseteq F \times F$ is the conflict relation,
  - $H = \{Z_e \subseteq F | e \in S \cup O\}$ is the history set,
- $\sigma = OP \times X$ is the input alphabet where
  - $OP = \{read, write\}$ is the set of operations,
  - $X = S \times O$ arguments of these operations,
- $\delta : Q \times \sigma \rightarrow Q$ is the state transition function,
- $q_0 \in Q$ is the initial state

Discretionary Access Control (DAC)

Individual users specify access rules to objects within their area of responsibility (at their discretion).

Mandatory Access Control (MAC)

System designers and administrators specify system-wide rules, that apply for all users and cannot be sidestepped.

Non-Interference

Two domains do not interfere with each other iff no action in one domain can be observed by the other.

Trusted Computing Base (TCB)

The set of functions of an IT system that are necessary and sufficient for implementing its security properties $\rightarrow$ Isolation, Policy Enforcement, Authentication ...

Security Architecture

part of a system's architecture that implement its TCB $\rightarrow$ Security policies, PDP and PEPs, ...