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

- Reliability & Correctness
- Real Time & Scalability
- Openness
- Conditio sine qua non: Provability of information properties
- non-repudiability (”nicht-abstreitbar”)

Specific Security Goals (Terms)

- **Confidentiality** the property of information to be available only to anauthorized 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
- **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

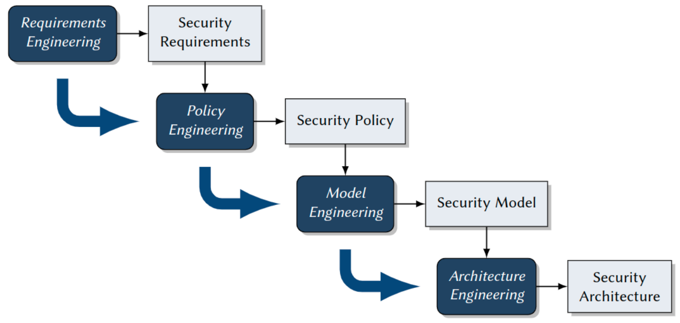
Security Goals in Practice

- ... are diverse and complex to achieve
- ... require multiple stakeholders to cooperate
- ... involve cross-domain expertise

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

Steps in Security Engineering



Security Requirements

Goal of Requirements Engineering: Methodology for identifying and specifying the desired security properties of an IT system.
Result:

- 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

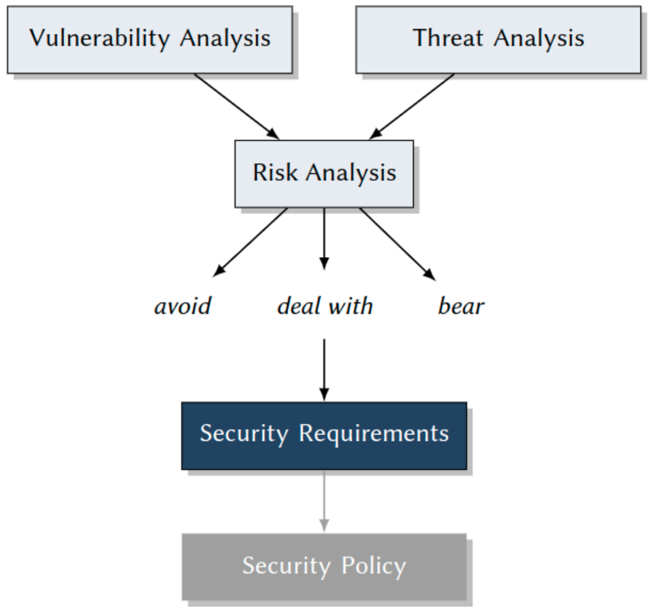
Influencing Factors

- 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)
- Criteria
- 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)

General Methodology: How to Come up with Security Requirements
Specialized steps in regular software requirements engineering:

1. Identify and classifyvulnerabilities.
2. Identify and classifythreats.
3. Match both, where relevant, to yieldrisks.
4. Analyze and decide which risks should bedealt with.

→ Fine-grained Security Requirements



Vulnerability Analysis

Goal: 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'sattack 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 flowin access control systems

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 by the project manager, should ever go into the product flyer.

Forbidden Information Flow Internal information about ProjectX goes into the product flyer!

Problem Analysis:

- Limited knowledge of users
  - limited horizon: knowledge about the rest of a system
  - limited problem awareness: see ”lack of knowledge”
  - limited skills
- 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

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
- Even better: most precise knowledge about the compiler used
  - how call conventions affect the stack layout
  - degree to which stack layout is deterministic

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!

Summary - Vulnerabilities

- Human
  - Laziness
  - Social engineering
  - Lack of knowledge (e.g. malware execution)
- Organizational
  - Key management
  - Physical access to rooms, hardware
- Technical
  - Weak security paradigms
  - Specification and implementation errors

Threat Analysis

Goal: Identification of

- Attack objectives and attackers
- Attack methods and practices (Tactics, 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

Exploitation of Vulnerabilities

Scenario 1: Insider Attack

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

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

- Trojan horses: Executable code with hidden functionality
- Viruses: Code for self-modification and self-duplication
- Logical bombs: Code that is activated by some event recognizable from the host (e. g. time, date, temperature, ...).
- Backdoors: 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
- Worms and worm segments: Autonomous, self-duplicating programs

Scenario 3: Outsider Attack

- Attack Method: Buffer Overflow
- Exploitation of implementation errors

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. automated 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 (from inside a network :nmap, from outside: by port scans)
  - 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 servers, utilities, libraries, OS modules
  - containing backdoors or smoke bombs for future attacks
- Results:
  - 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), e.g. syslog,kern.log,user.log,daemon.log,auth.log, ...
- 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
- Result: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, Refined methodology, 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: near zero

Countermeasures - Options:

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

Security Engineering

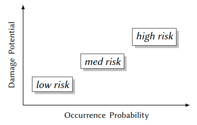
- New paradigms: policy-controlled systems → powerful software platforms
- New provable guarantees: formal security models → reducing specification errors and faults by design
- New security architectures → limiting bad effects of implementation errors and faults

Risk Analysis

Identification and Classification of scenario-specific risks

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

Risk Classification: Qualitative risk matrix/dimensions



Assessment

Damage Potential Assessment

- Cloud computing → loss of confidence/reputation
- Industrial plant control → damage or destruction of facility
- Critical public infrastructure → interrupted services, possible 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 level
- Traffic management → depending on terroristic threat level

Damage potential & Occurrence probability is highly scenario-specific

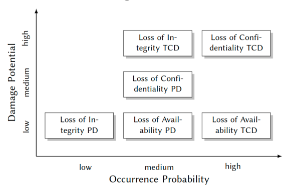
Depends on diverse, mostly non-technical side conditions → advisory board needed for assessment

Advisory Board Output Example

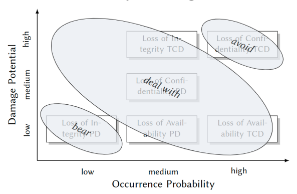
Object	Risk (Loss of...)	Dmg. Pot.	Rationale
PD	Confidentiality	med	Data protection acts
PD	Confidentiality	med	Certified software
PD	Integrity	low	Errors fast and easily detectable and correctable
PD	Integrity	low	Certified software, small incentive
PD	Availability	med	Certified software
PD	Availability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

PD = Personal Data; TCD = Technical Control Data

Resulting Risk Matrix



Identify 3 Regions



Form Risks to Security Requirements

- 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 Architecture

Security Policies: a preliminary Definition

- We have risks: Malware attack → violation of confidentiality and integrity of patient's medical records
- We infer security requirements: Valid information flows
- We 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:

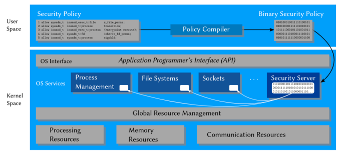
- informal (natural language) text
- formal model
- functional software specification
- executable code

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 →



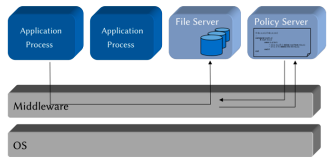
```
1 allow sysadm_t  insmod_exec_t:file     x_file_perms;
2 allow sysadm_t  insmod_t:process       transition;
3 allow insmod_t  insmod_exec_t:process  {entrypoint execute};
4 allow insmod_t  sysadm_t:fd            inherit_fd_perms;
5 allow insmod_t  sysadm_t:process       sigchld;
```

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** The security policy is only known and enforced by oneuser program → implemented in a user-space application
- **Application-level Security Architecture** The security 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** The security 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

Goal of Formal 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 (usually too complex) reality → get rid of insignificant details
- Precisionin 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
Taxonomy

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

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)

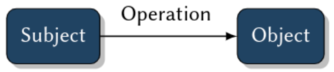
DAC vs. MAC

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.



There are

- **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]

- A really basic model to define access rights:
  - Who (subject) is allowed to do what (operation) on which object
  - Fundamental to OS access control since 1965
  - Formal paradigms: sets and functions
- 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 [1974] addresses the questions 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 ("⇔"). Despite a purely theoretical model: paved the way for practically implementing AC meta-information as tables, 2-dimensional lists, distributed arrays and lists.
Example

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

Implementation Notes

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

Access Control Lists (ACLs)

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

Capability Lists

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

What we actually Model:

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

Goal of ACF/ACM is 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 (snapshots) of an AC system
- Deterministic automata (state machines) → for modeling runtime changes of a protection state

This idea was pretty awesome. We need to understand automata, since from then on they were used for most security models.

Deterministic Automata

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

- 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.

Interpretation

- 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 $\langle op, (x_1, ..., x_k)\rangle \in \sum$ 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 $1 \leq i, j \leq k$.
- $\rightarrow$ We also call δ the state transition scheme (STS) of a model.
- Historically: äuthorization scheme"[Harrison et al., 1976].

State Transition Scheme (STS)

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

- $q = \{S_q, O_q, m_q\} \in Q, op \in OP$
- $r_1 ... r_m \in R$
- $x_{s1}, ..., x_{sm} \in S_q$ and $x_{o1}, ..., x_{om} \in O_q$ where s_i and o_i, $1 \leq i \leq m$, are vector indices of the input arguments: $1 \leq s_i, o_i \leq k$
- $p_1, ..., p_n$ 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 $q' = \sigma(q, \langle op, (x_1, ..., x_k)\rangle)$ 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 $\rightarrow$ 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 $\rightarrow$ define σ using the normalized form/programming syntax of the STS
3. Initialization: Define a well-known initial stateq $0 = \langle S_0, O_0, m_0\rangle$ of the system to model

1. Model Sets

- Subjects, objects, operations, rights:
  - Subjects: An unlimited number of possible students: $S \cong \mathbb{N}$
  - Objects: An unlimited number of possible solutions: $O \cong \mathbb{N}$
  - Operations:
    - (a) Submit $writeSolution(s_{student}, o_{solution})$
    - (b) Download $readSample(s_{student}, o_{sample})$
    - $\rightarrow OP = \{writeSolution, readSample\}$
  - Rights: Exactly one allows to execute each operation
    - $R \cong OP \rightarrow R = \{write, read\}$

2. State Transition Scheme: Effects of operations on protection state

```
        command writeSolution(s,o) ::= if write in m(s,o)
          then
            enter read into m(s,o);
          fi
        command readSample(s,o) ::= if read in m(s,o)
          then
            delete write from m(s,o);
          fi
```

3. Initialization

- By model definition: $q_0 = \langle S_0, O_0, m_0\rangle$
- For a course with (initially) three students:
  - $S_0 = \{sAnn, sBob, sChris\}$
  - $O_0 = \{oAnn, oBob, oChris\}$
  - m_0:
    - $m_0(sAnn, oAnn) = \{write\}$
    - $m_0(sBob, oBob) = \{write\}$
    - $m_0(sChris, oChris) = \{write\}$
    - $m_0(s, o) = \varnothing \Leftrightarrow s \neq o$
  - Interpretation: "There is a course with three students, each of whom has their own workspace to which she is allowed to submit (write) a solution."

Model Behavior

- Initial Protection State at beginning

m	oAnn	oBob	oChris
sAnn	write	∅	∅
sBob	∅	write	∅
sChris	∅	∅	write

- After $writeSolution(sChris, oChris)$

m	oAnn	oBob	oChris
sAnn	write	∅	∅
sBob	∅	write	∅
sChris	∅	∅	write, read

- After $readSample(sChris, oChris)$

m	oAnn	oBob	oChris
sAnn	write	∅	∅
sBob	∅	write	∅
sChris	∅	∅	read

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 student may (repeatedly) submit her respective solution.

Tricks in this Example

- The sample solution is not represented by a separate object $\rightarrow$ no separate column in the ACM.
- Instead, we smuggled the read right for it into the cell of each student's solution ...

HRU Model Analysis

Analysis of Right Proliferation $\rightarrow$ The HRU safety problem.
InputSequences

- ,,What is the effect of an input in a given state?" $\rightarrow$ a single state transition as defined by δ
- ,,What is the effect of an input sequence in a given state?" $\rightarrow$ a composition of sequential state transitions as defined by $\delta*$

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

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

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.

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 $safe(q_0, r)$.

all states in $\{\delta^*(q, \sigma^*) | \sigma^* \in \sum^*\}$ validated except for q'

m_q	o_1	o_2	o_3
s_1	$\{r_1, r_3\}$	$\{r_1, r_3\}$	$\{r_2\}$
s_2	$\{r_1\}$	$\{r_1\}$	$\{r_2\}$
s_3	∅	∅	$\{r_2\}$

$m_{q'}$	o_1	o_2	o_3	o_4
s_1	$\{r_1, r_3\}$	$\{r_1\}$	$\{r_2\}$	∅
s_2	$\{r_1, r_2\}$	$\{r_1\}$	$\{r_2\}$	$\{r_2\}$
s_3	∅	∅	∅	∅

- $r_3 \notin m_{q'}(s_1, o_2) \wedge r_3 \in m_q(s_1, o_1) \Rightarrow safe(q, r_3)$
- $r_2 \in m_{q'}(s_2, o_1) \wedge r_2 \notin m_q(s_2, o_1) \Rightarrow \neg safe(q, r_2)$
- $r_2 \in m_{q'}(s_2, o_4) \wedge o_4 \notin O_q \Rightarrow \neg safe(q, r_2)$

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 $m_{q'}$ ($\forall s \in S_{q'}, \forall o \in O_{q'} : ...$)
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 ($r \in m_q ...$).
4. Recursiv. proceed with 2. for any possible successor state q'' of q'

Safety Decidability

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
- $\rightarrow$,,Proofs teach us how to build things so nothing more needs to be proven." (W. E. Kühnhauser)

a mono-operational HRU model $\rightarrow$ 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
3. $\rightarrow$ safety is decidable

Proof:

- construct finite sequences ...$\rightarrow$
- Transform $\sigma_1 ... \sigma_n$ 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	5.
...	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 $\rightarrow$ 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 $\rightarrow$ goes as far as uselessness (only create files)
  - are efficient to analyze: algorithms and tools for safety analysis
  - $\rightarrow$ are always guaranteed to terminate
  - $\rightarrow$ 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) $\rightarrow$ 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
- $\rightarrow$ 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
   - $\rightarrow$ Runtime: polynomial in model size ($q_0 + STS$)
2. Simulation phase: The automaton is implemented and, starting with q_0, fed with inputs $\sigma =< op, x >$
   - $\rightarrow$ 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 $\rightarrow$ safety undecidability
- Performance: Model size 10 000 000 $\approx$ 417s

Achievements

- Find typical errors in security policies: Guide designers, who might know there's something wrong w. r. t. right proliferation, but not what and why!
- Increase our understanding of unsafety origins: By building clever heuristics, we started to understand how we might design specialized HRU models ($\rightarrow$ fixed STS, type system) that are safety-decidable yet practically (re-) usable

Summary HRU Models

Goal

- Analysis of right proliferation in AC models
- Assessing the computational complexity of such analyses

Method

- Combining ACMs and deterministic automata
- Defining $safe(q, r)$ based on this formalism

Conclusions

- Potential right proliferation: Generally undecidable problem
- $\rightarrow$ HRU model family, consisting of application-tailored, safety-decidable variants
- $\rightarrow$ Heuristic analysis methods for practical error-finding

The Typed-Access-Matrix Model (TAM)

- AC model, similar expressiveness to HRU
- $\rightarrow$ directly mapped to implementations of an ACM (DB table)
- Better suited for safety analyses: precisely statemodel properties for decidable safety

Idea

- Adopted from HRU: subjects, objects, ACM, automaton
- New: leverage the principle of strong typing (like programming)
- $\rightarrow$ safety decidability properties relate to type-based restrictions

How it Works:

- 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.

State Transition Scheme (STS) $\delta : Q \times \sum \rightarrow Q$ is defined by a set of specifications:

$$\begin{aligned} \delta(q, \langle op, (\langle x_1, t_1 \rangle, \dots, \langle x_k, t_k \rangle)\rangle) = \textbf{if } & r_1 \in m_q(x_{s_1}, x_{o_1}) \quad \wedge \\ & \dots \quad \wedge \\ & r_m \in m_q(x_{s_m}, x_{o_m}) \\ \textbf{then } & \\ & p_1 \circ \dots \circ p_n \end{aligned}$$

where

- $q = (S_q, O_q, type_q, m_q) \in Q, op \in OP$
- $r_1, ..., r_m \in R$
- $x_{s1}, ..., x_{sm} \in S_q, x_{o1}, ..., x_{om} \in Oq \backslash S_q$, and $t_1, ..., t_k \in T$ where s_i and $o_i, 1 \leq i \leq m$, are vector indices of the input arguments: $1 \leq s_i, o_i \leq k$
- $p_1, ..., p_n$ are TAM primitives

```
command op(x₁:t₁, …, xₖ:tₖ) ::= if $r_1 \in$ m($x_{s_1}, x_{o_1}$)  ∧
                                    …                    ∧
                                    $r_m \in$ m($x_{s_m}, x_{o_m}$)
                                then
                                    $p_1$;
                                    …
                                    $p_n$;
                                fi
```

Convenience Notation where

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

$$\begin{aligned}\delta(q, \langle op, (\langle x_1, t_1\rangle, \dots, \langle x_k, t_k\rangle)\rangle) = \text{if } & type_q(x_1) = t_1 & \wedge \\ & \dots & \wedge \\ & type_q(x_k) = t_k & \wedge \\ & r_1 \in m_q(x_{s_1}, x_{o_1}) & \wedge \\ & \dots & \wedge \\ & r_m \in m_q(x_{s_m}, x_{o_m}) & \\ \text{then } & & \\ & p_1 \circ \dots \circ p_n & \end{aligned}$$

TAM-specific

- Implicit Add-on:Type Checking
- where t_i are the types of the arguments $x_i, 1 \leq i \leq k$.

TAM-specific

- 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 $type : O \to T$ 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

Auxiliary analysis tools:

Parent- and Child-Types For any operation op with arguments $\langle x_1, t_1\rangle, ..., \langle x_k, t_k\rangle$ 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.

$$\begin{tikzcd} & w \arrow[dl] \arrow[dr] & \\ u \arrow[r] \arrow[loop left] & & v \end{tikzcd}$$

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.

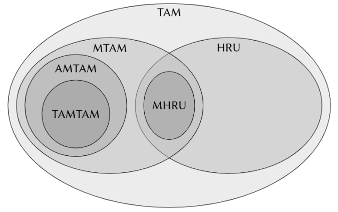
Crucial property acyclic, intuitively:

- 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



IBAC Summary

- Model identity-based AC policies (IBAC)
- Analyze them w.r.t. basic security properties (right proliferation)
- → Minimize specification errors
- → Minimize implementation errors
- Approach
  - Unambiguous policy representation through formal notation
  - Prediction and/or verification of mission-critical properties
  - Derivation of implementation concepts
- Model Range - Static models:
  - Access control function: $f : S \times O \times OP \to \{true, false\}$
  - Access control matrix (ACM): $m : S \times O \to 2^{OP}$
  - Static analysis: Which rights are assigned to whom, which (indirect) information flows are possible
  - Implementation: Access control lists (ACLs)
- Model Range - Dynamic models:
  - ACM plus deterministic automaton → Analysis of dynamic behavior: HRU safety
  - generally undecidable
  - decidable under specific restrictions: monotonous mono-conditional, static, typed, etc.
  - identifying and explaining safety-violations, in case such (are assumed to) exists: heuristic analysis algorithms
- Limitations
  - IBAC models are fundamental: KISS
  - IBAC models provide basic expressiveness only
- For more application-oriented policy semantics:
  - Large information systems: many users, many databases, files, ... → Scalability problem
  - Access decisions not just based on subjects, objects, and operations → Abstraction problem

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

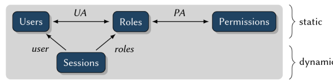
- Improved scalability and manageability
- application-oriented semantic: $roles \approx functions$ in organizations
- Models include smart abstraction: roles
- AC rules are specified based on roles instead of identities
- Users, roles, and rights for executing operations
- Access rules are based onrolesof users → on assignments

Basic RBAC model „$RBAC_0$" 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, that originate from the workflows and areas of responsibility in organizations
- Permissions P model rights for any particular access to a particular document
- user-role-relation $UA \subseteq U \times R$ defines which roles are available to users at any given time → must be assumed during runtime first before they are usable!
- permission-role-relation $PA \subseteq P \times R$ defines which permissions are associate with roles
- UA and PA describe static policy rules: Roles available to a user are not considered to possibly change, same with permissions associated with a role.

- Sessions S describe dynamic assignments of roles $\rightarrow$ a session $s \in S$ models when a user is logged in(where she may use some role(s) available to her as per UA):
  - The session-user-mapping user: $S \rightarrow U$ associates a session with its (öwning") user
  - The session-roles-mapping roles: $S \rightarrow 2^R$ 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 $P \rightarrow$ 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** $f_{RBAC_0} : U \times O \times OP \rightarrow \{true, false\}$ where

$f_{RBAC_0}(u, o, op) = \begin{cases} true, & \exists r \in R, s \in S : u = user(s) \land r \in roles(s) \land \langle\langle o, op\rangle, r\rangle \in PA \\ false, & \text{otherwise} \end{cases}$

RBAC96 Model Family

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

- Roles are often hierarchical $\rightarrow RBAC_1 = RBAC_0 + hierarchies$
- Role association and activation are often constrained $\rightarrow RBAC_2 = RBAC_0 + constraints$
- Both may be needed $\rightarrow RBAC_3 =$ consolidation: $RBAC_0 + RBAC_1 + RBAC_2$

RBAC 1: Role Hierarchies

Roles often overlap

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

Solution: Role hierarchy $\rightarrow$ 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
- Interpretation
  - Reflexivity: any role consists of (ïnherits") 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: $\forall r, r' \in R, \exists s \in S : r \leq r' \land r' \in roles(s) \Rightarrow r \in roles(s)$.

RBAC 2 : Constraints

Assuming and activating roles in organizations is often more restricted:

- Certain roles may not be active at the same time (same session) for any user
- Certain roles may not be together assigned to any user
- $\rightarrow$ 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, other permissions

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 $UA, PA, user, roles$)

RBAC Summary

- Scalability
- Application-oriented model abstractions
- Standardization (RBAC96) $\rightarrow$ tool-support for:
  - role engineering (identifying and modeling roles)
  - model engineering (specifying/validating a model config.)
  - static model checking (verifying consistency and plausibility of a model configuration)
- Still weak OS-support
  - $\rightarrow$ application-level integrations
  - $\rightarrow$ middleware integrations
- Limited dynamic analyses w.r.t. automaton-based models

Attribute-based Access Control Models

- 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; only assignable to users
- $\rightarrow$ Consider application-specific context of an 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 subjects or 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

- $f_{IBAC} : S \times O \times OP \rightarrow \{true, false\}$
- $f_{RBAC} : U \times O \times OP \rightarrow \{true, false\}$
- $f_{ABAC} : S \times O \times OP \rightarrow \{true, false\}$
- $\rightarrow$ 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,
- $A_S = V_S^1 \times ... \times V_S^n$ is a set of subject attributes, where each attribute is an n-tuple of values from arbitrary domains V_S^i, $1 \leq i \leq n$,
- $A_O = V_O^1 \times ... \times V_O^m$ is a corresponding set of object attributes, based on values from arbitrary domains V_O^j, $1 \leq j \leq m$,
- $att_S : S \rightarrow A_S$ is the subject attribute assignment function,
- $att_O : O \rightarrow A_O$ 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 $\rightarrow$ determines ACF!
- AAR is based on a set of first-order logic predicates Φ: $\Phi = \{\phi_1(x_{s1}, x_{o1}), \phi_2(x_{s2}, x_{o2}), ...\}$. 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)
- $f_{ABAC} : S \times O \times OP \rightarrow \{true, false\}$ 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.

ABAC Summary

- Scalability
- Application-oriented model abstractions
- Universality: ABAC can conveniently express IBAC, RBAC, MLS
- Still weak OS-support $\rightarrow$ application-level integrations
- Attribute semantics highly diverse, not normalizable $\rightarrow$ no common ,,standard ABAC"
- Limited dynamic analyses w.r.t. automaton-based models

Information Flow Models

Abstraction Level of AC Models: rules about subjects accessing objects. Adequate for

- Workflow systems
- Document/information management systems

Goal of Information Flow (IF) Models: Problem-oriented definition of policy rules for scenarios based on information flows(rather than access rights)
Lattices (refreshment)

- inf_C: ,,systemlow"
- sup_C: ,,systemhigh"
- has a source: $deg^-(inf_C) = 0$
- has a sink: $deg^+(sup_C) = 0$

Implementation of Information Flow Models

- Information flows and read/write operations are isomorphic
  - s has read permission o ⇔ information may flow from o to s
  - s has write permission o ⇔ information may 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 $\leq d$ ⇔ 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

We can now ...

- precisely define all information flows valid for a given policy
- define analysis goals for an IF model w.r.t.
  - Correctness: ∃ covert information flows? (transitivity of $\leq$, automation: graph analysis tools)
  - Redundancy: ∃ sets of subjects and objects with (transitively) equivalent information contents? (antisymmetry of $\leq$, automation: graph analysis tools)
- implement a model: through an automatically generated, isomorphic ACM(using already-present ACLs!)

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

Idea:

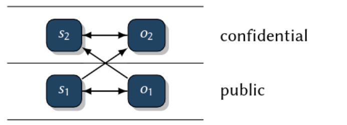
- entity sets S,O
- $lattice\langle C, \leq\rangle$ defines information flows by
  - C: classification/clearance levels
  - $\leq$: hierarchy of trust
- classification function cl assigns
  - clearance level from C to subjects
  - classification level from C to objects
- Model's runtime behavior is specified by a deterministic automaton

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
  - $M = \{m \mid m : S \times O \rightarrow 2^R\}$ is the set of possible ACMs,
  - $CL = \{cl \mid cl : S \cup O \rightarrow C\}$ 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 subjects and 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.



$\Longleftrightarrow$

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
- m provides an easy specification for model implementation

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.

Auxiliary Definition: The Basic Security Theorem for BLP (BLP BST)

- A convenient tool for proving BLP security
- Idea: let's look at properties of the finite and small model components → σ → STS

The 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 $\in m(s, o) \wedge \neg(cl'(s) \leq cl'(o)) \Rightarrow write \notin m'(s, o)$

Proof of Read Security

- Let $q = \sigma * (q_0, \sigma^+), \sigma^+ \in \sigma^+, q' = \delta(q, \sigma), \sigma \in \sigma, s \in S, o \in O$. With $q = \langle m, cl\rangle$ and $q' = m', cl'$, the BLP BST for read-security

- (a1) $read \notin m(s,o) \wedge read \in m'(s,o) \Rightarrow cl'(o) \leq cl'(s)$
- (a2) $read \in m(s,o) \wedge \neg(cl'(o) \leq cl'(s)) \Rightarrow read \notin m'(s,o)$
- Let's first introduce some convenient abbreviations for this:
  - $R := read \in m(s,o)$
  - $R' := read \in m'(s,o)$
  - $C' := cl'(o) \leq cl'(s)$
  - σ^+ is the set of finite, non-empty input sequences.
- Proposition: $(a1) \wedge (a2) \equiv read - security$
- Proof:
  $(a1) \wedge (a2) = R' \Rightarrow C' \equiv read \in m'(s,o) \Rightarrow cl'(o) \leq cl'(s)$,
  which exactly matches the definition of read-security for q'.
- Write-security: Same steps for $(b1) \wedge (b2)$.

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 $\Leftrightarrow \forall s \in S, o \in O : write \in m(s,o) \Rightarrow cl(s) \leq cl(o) \wedge co(o) \subseteq co(s)$
- old BLP: $\langle S, O, L, Q, \sigma, \delta, q_0 \rangle$
- With compartments: $\langle S, O, L, Comp, Q_{co}, \sigma, \delta, q_0 \rangle$ where $Q_{co} = M \times CL \times CO$ and $CO = \{co | co : S \cup O \rightarrow 2^{Comp}\}$

Example

- Let $co(o) = secret, co(o) = airforce$
- s_1 where $cl(s_1) = public, co(s_1) = \{airforce, navy\}$ can write o
- s_2 where $cl(s_2) = secret, co(s_2) = \{airforce, navy\}$ read/write o
- s_3 where $cl(s_3) = secret, co(s_3) = \{navy\}$ can do neither

BLP Model Summary

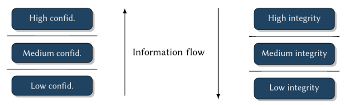
- Application-oriented modeling → hierarchical information flow
- Scalability → attributes: trust levels
- Modeling dynamic behavior → automaton with STS
- Correctness guarantees (analysis of)
  - consistency: BLP security, BST
  - completeness of IF: IFG path finding
  - presence of unintended IF: IFG 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

learn from BLP for designing and using security models

- Model composition from known model abstractions
  - Denning: IF modeling
  - ABAC: IF classes and compartments as attributes
  - MSL: modeling trust as a linear hierarchy
  - HRU: modeling dynamic behavior
  - ACM: implementing application-oriented policy semantics
- Consistency is an important property of composed models
- BLP is further extensible and refinable

The Biba Model

BLP upside down



- BLP → preserves confidentiality
- Biba → preserves integrity

OS Example

- Integrity: Protect system files from malicious user/software
- Class hierarchy (system, high, medium, low)
- every file/process/... created is classified → cannot violate integrity of objects
- Manual user involvement: resolving intended exceptions, e.g. install trusted application

Non-interference Models

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
  - Timing channels from server response times

Idea of NI models

- higher level of abstraction
- Policy semantics: which domains should be isolated based on their mutual impact

Consequences

- Easier policy modeling
- More difficult policy implementation → higher degree of abstraction

Example

- Fields: Smart Cards, Server System
- Different services, different providers, different levels of trust
- Shared resources
- Needed: isolation of services, restricted cross-domain interactions
- → Guarantee of total/limited non-interference between domains

NI Security Policies

Specify

- Security domains
- Cross-domain (inter)actions → interference

From convert channels to domain interference:

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,
- $\delta : Q \times \sigma \rightarrow Q$ 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.

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

Goals

- AC models: privilege escalation (→ HRU safety)
- BLP models: model consistency (→ BLP security)
- NI models: Non-interference between domains

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: $A^* \times 2^D \rightarrow A^*$ computes a subsequence of aa^* by removing such actions without an observable effect on any element of D' :

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

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) $\rightarrow$ expensive
  - State of the Art w.r.t. isolation completeness

Hybrid Models

Chinese-Wall Policies

for consulting companies

- Clients of any such company
  - Companies, including their business data
  - Often: mutual competitors
- Employees of consulting companies
  - Are assigned to clients they consult
  - Work for many clients $\rightarrow$ gather insider information
- Policy goal: No flow of (insider) information between competing clients

Why look at specifically these policies? Modeling

- Composition of
  - Discretionary IBAC components
  - Mandatory ABAC components
- Driven by real-world 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

Explicitly tailored towards Chinese Wall (CW) policies
Model Abstractions

- Consultants represented by subjects
- Client companies represented by objects, which comprise a company's business data
- Modeling of competition by conflict classes: two different clients are competitors $\Leftrightarrow$ their objects belong to the same class
- No information flow between competing objects $\rightarrow$ 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 (non-reflexive, symmetric, generally not transitive)
- In terms of ABAC:object attribute $att_O : O \rightarrow 2^O$, 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 relation $H \subseteq S \times O : \langle s, o \rangle \in H \Leftrightarrow s$ has previously consulted o
- In terms of ABAC: subject attribute $att_S : S \rightarrow 2^O$, such that $att_S(s) = \{o \in O | \langle s, o \rangle \in H\}$.

Brewer-Nash Security Model The Brewer-Nash model of the CW policy is a det. *automaton*$\langle S, O, Q, \sigma, \delta, q_0, R \rangle$ where
- S and O are sets of subjects (consultants) and (company data) objects,
- $Q = M \times 2^C \times 2^H$ is the state space where
  - $M = \{m | m : S \times O \rightarrow 2^R\}$ 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,
- $\delta : Q \times \sigma \rightarrow Q$ 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

Not shown: Discretionary policy portion $\rightarrow$ modifications in m to enable fine-grained rights management.
Restrictiveness

- Write Command: s is allowed to write $o \Leftrightarrow write \in m(s, o) \wedge \forall o' \in O : o' \neq o \Rightarrow \langle s, o' \rangle \notin H$
- Why so restrictive? $\rightarrow$ No transitive information flow!
- $\rightarrow$ 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
  - m_0: consultant assignments to clients, issued by management
  - C_0: according to real-life competition
  - $H_0 = \varnothing$

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".

Summary Brewer-Nash

What's remarkable with this model?

- Composes DAC and MAC components
- Simple model paradigms
  - Sets (subjects, objects)
  - ACM (DAC)
  - Relations (company conflicts, consultants history)
  - Simple ,,read" and ,,write" rule
  - $\rightarrow$ easy to implement
- Analysis goals
  - MAC: Model security
  - DAC: safety properties
- Drawback: Restrictive write-rule

Professionalization

- Remember the difference: trusting humans (consultants) vs. trusting software agents (subjects)
  - Consultants are assumed to be trusted
  - Systems (processes, sessions, ...) may fail
- $\rightarrow$ Write-rule applied not to humans, but to software agents
- $\rightarrow$ Subject set S models consultant's subjects (e.g. processes) in a group model
  - All processes of one consultant form a group
  - Group members
    - have the same rights in m
    - have individual histories
    - are strictly isolated w.r.t. IF

The Least-Restrictive-CW Model

Restrictiveness of Brewer-Nash Model:

- If $\langle o_i, o_k \rangle \in C$: no transitive information flow $o_i \rightarrow o_j \rightarrow o_k$, i.e. consultant(s) of o_i must never write to any $o_j \neq o_i$
- This is actually more restrictive than necessary: $o_j \rightarrow o_k$ and afterwards $o_i \rightarrow o_j$ would be 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
- $\rightarrow$ subject-/object-specific history, $\approx$attributes (,,lables")

LR-CW Model The Least-Restrictive 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

Inside the STS

- a reading operation: requires that no conflicting information is accumulated in the subject potentially increases the amount of information in the subject
- a writing operation: 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 labels)
  - Dependency of write permissions on earlier actions of other 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
- In example, a vector consists of 2 elements $\in C$ resulting in labels as:
  - $[\epsilon, \epsilon]$ (exclusively for inf_C)
  - $[DB, \epsilon]$ (for DB-objects or -consultants)
  - $[DB, Shell]$ (for subjects or objects containing information from both DB and Shell)

Why is the ,,Chinese Wall" policy interesting?

- One policy, multiple models:
- Brewer-Nash model demonstrates hybrid DAC-/MAC-/IFC-approach
- Least-Restrictive CW model demonstrates a more practical professionalization
- MLS-CW model demonstrates applicability of lattice-based IF modeling → semantically cleaner approach
- Applications: Far beyond traditional consulting scenarios...→ current problems in cloud computing!

Summary - Security Models

- Formalize informal security policies for the sake of
  - objectification by unambiguous calculi
  - explanation and proof of security properties by formal analysis techniques
  - foundation for correct implementations
- Are composed of simple building blocks (e.g. ACMs, sets, relations, functions, lattices, state machines) that are combined and interrelated to form more complex models

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

Model Engineering Principles

- Core model
- Core specialization
- Core extension
- Component glue

Core Model (Common Model Core)

- HRU: $\langle Q, \sum, \delta, q_0, \not R\rangle$
- $DRBAC_0$: $\langle Q, \sum, \delta, q_0, \not R, \not P, \not PA\rangle$
- DABAC: $\langle \not A, Q, \sum, \delta, q_0\rangle$
- TAM: $\langle Q, \sum, \delta, q_0, \not T, \not R\rangle$
- BLP: $\langle \not S, \not O, \not L, Q, \sum, \delta, q_0, \not R\rangle$
- NI: $\langle Q, \sum, \delta, \not\lambda, q_0, \not D, \not A, \not dom, \neq_{NI}, \not Out\rangle$
- $\rightarrow \langle Q, \sum, \delta, q_0\rangle$

Core Specialization

- HRU: $\langle Q, \sum, \delta, q_0, R\rangle \Rightarrow Q = 2^S \times 2^O \times M$
- $DRBAC_0$: $\langle Q, \sum, \delta, q_0, R, P, PA\rangle \Rightarrow Q = 2^U \times 2^{UA} \times 2^S \times USER \times ROLES$
- DABAC: $\langle A, Q, \sum, \delta, q_0\rangle \Rightarrow Q = 2^S \times 2^O \times M \times ATT$
- TAM: $\langle Q, \sum, \delta, q_0, T, R\rangle \Rightarrow Q = 2^S \times 2^O \times TYPE \times M$
- BLP: $\langle S, O, L, Q, \sum, \delta, q_0, R\rangle \Rightarrow Q = M \times CL$
- NI: $\langle Q, \sum, \delta, \lambda, q_0, D, A, dom, =_{NI}, Out\rangle$

Core Extensions

- HRU: $\langle Q, \sum, \delta, q_0, R\rangle \Rightarrow R$
- $DRBAC_0$: $\langle Q, \sum, \delta, q_0, R, P, PA\rangle \Rightarrow R, P, PA$
- DABAC: $\langle A, Q, \sum, \delta, q_0\rangle \Rightarrow A$
- TAM: $\langle Q, \sum, \delta, q_0, T, R\rangle \Rightarrow T, R$
- BLP: $\langle S, O, L, Q, \sum, \delta, q_0, R\rangle \Rightarrow S, O, L, R$
- NI: $\langle Q, \sum, \delta, \lambda, q_0, D, A, dom, =_{NI}, Out\rangle \Rightarrow \lambda, D, A, dom, =_{NI}, Out$
- $\rightarrow R, P, PA, A, T, S, O, L, D, dom, =_{NI}, \ldots$

Glue

- E.g. TAM: State transition scheme (types)
- E.g. DABAC: State transition scheme (matrix and predicates)
- E.g. Brewer/Nash Chinese Wall model: ,,$\wedge$" (simple, because $H + C \neq m$)
- E.g. BLP (much more complex, because rules restrict m by L and cl)
  - BLP read rule
  - BLP write rule
  - BST

Model Specification

Policy Implementation

- We want: A system controlled by a security policy
- We have: A (satisfying) formal model of this policy
- How to convert a formal model into an executable policy? → Policy specification languages
- How to enforce an executable policy in a system? → security mechanisms and architectures

Role of Specification Languages: Same as in software engineering

- 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

Approach

- Abstraction level: Step stone between model and security mechanisms
- → More concrete than models
- → More abstract than programming languages (,,what" instead of ,,how")
- 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)

Model Specification

CorPS

SELinux Policy Language

Summary

Security Mechanisms

Authorization

Access Control Lists

Capability Lists

Interceptors

Summary

Cryptographic Mechanisms

Encryption

Symmetric

Asymmetric

Cryptographic Hashing

Digital Signatures

Cryptographic Attacks

Identification and Authentication

Passwords

Biometrics

Cryptographic Protocols

SmartCards

Authentication Protocols

Summary

Security Architectures

Design Principles

Operating Systems Architectures

Nizza

SELinux

Distributed Systems Architectures

CORBA

Web Services

Kerberos

Summary