Security Models

Introduction

Bell-LaPadula model designed to capture a specific ‘military’ security policy.

At one time treated as ‘the model of security’.

However, security requirements dependent on the application; many applications do not need multi-level security.

We will now look at models for ‘commercial’ integrity policies.

We will also examine some theoretical foundations of access control.

Agenda

Biba model

Chinese Wall model

Clark Wilson Model

Harrison-Ruzo-Ullman model

- Reminder: Turing machines& decidability, NP-completeness

Information flow models

Enforcement monitors

Biba Model

Integrity policies prohibit the corruption of 'clean' high level entities by 'dirty' low level entities.

- Clean and dirty shorthand for high integrity and low integrity.
- Concrete meaning of integrity levels is application dependent.

Subjects and objects labelled with elements from a lattice $(L, \quad)$ of integrity levels by functions $f_S{:}S \quad L$ and $f_O{:}O \quad L$.

Information may only flow downwards in the integrity lattice; only information flows caused directly by access operations considered.

Biba model: state machine model similar to BLP; no single high-level integrity policy.

Biba With Static Integrity Levels

Simple Integrity Property (no write-up): If subject s can modify (alter) object o, then $f_S(s) \quad f_O(o)$.

Integrity -Property: If subject s can read (observe) object o, then s can have write access to some other object o' only if $f_O(o) \quad f_O(o')$.

Invoke Property: A 'dirty' subject s_1 must not touch a 'clean' object indirectly by invoking s_2: Subject s_1 can invoke subject s_2 only if $f_S(s_1) \quad f_S(s_2)$.

Biba: Dynamic Integrity Levels

Low watermark policies automatically adjust levels (as in the Chinese Wall model):

Subject Low Watermark Policy: Subject s can read (observe) an object o at any integrity level. The new integrity level of s is g.l.b.$(f_S(s), f_O(o))$.

Object Low Watermark Policy: Subject s can modify (alter) an object o at any integrity level. The new integrity level of o is g.l.b.$(f_S(s), f_O(o))$.

Biba for Protection Rings

Ring Property: A 'dirty' subject s_1 may invoke a 'clean' tool s_2 to touch a 'clean' object:

Subject s_1 can read objects at all integrity levels, modify objects o with $f_S(s_1) \quad f_O(o)$, and invoke a subject s_2 only if $f_S(s_1) \quad f_S(s_2)$.

The ring property is the opposite of the invoke property!

Captures integrity protection in operating systems based on protection rings.

Chinese Wall Model

In financial institutions analysts deal with a number of clients and have to avoid conflicts of interest.

Components:

- subjects: analysts
- objects: data item for a single client
- company datasets: $y:O$ C gives for each object its company dataset
- conflict of interest classes: companies that are competitors; $x: O$ $P(C)$ gives for each object o the companies with a conflict of interest on o
- 'labels': company dataset + conflict of interest class
- sanitized information: no access restrictions

Chinese Wall Model – Policies

Simple Security Property: Access is only granted if the object requested

- is in the same company dataset as an object already accessed by that subject;
- does not belong to any of the conflict of interest classes of objects already accessed by that subject.

Formally:

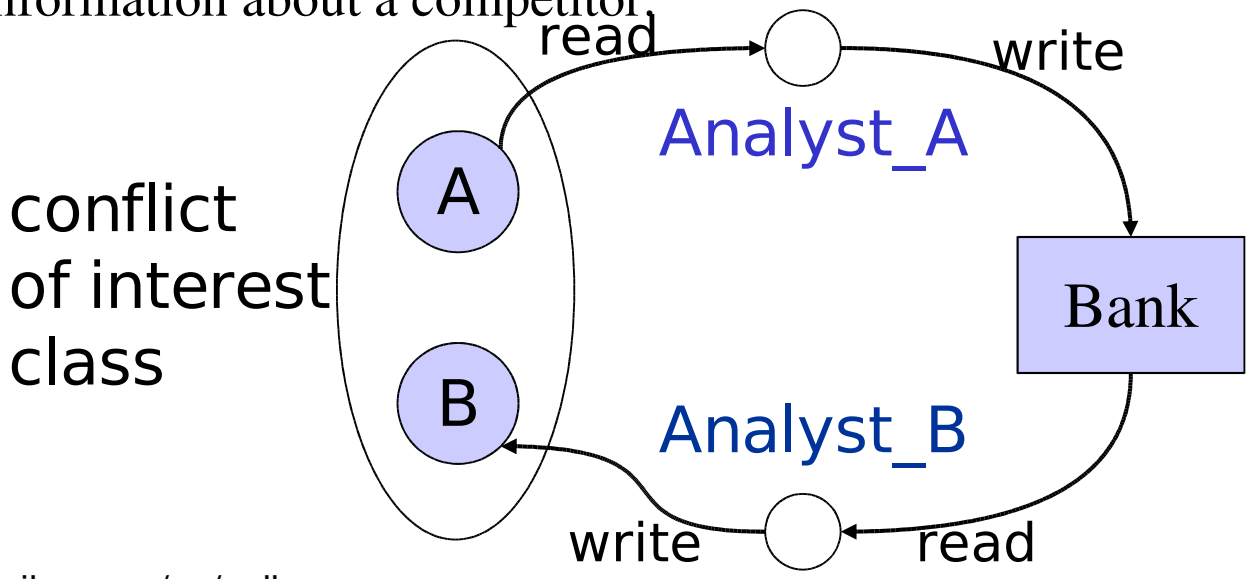
- $N = (N_{so})_{s \ S, o \ O}$, Boolean matrix, N_{so} = true if s has accessed o;
- ss-property: subject s gets access to object o only if for all objects o' with $N_{so'}$ = true, $y(o) = y(o')$ or $y(o) \quad x(o')$.

Chinese Wall: - Property

Indirect information flow: *A* and *B* are competitors having accounts with the same *Bank*.

Analyst_A, dealing with *A* and the *Bank*, updates the *Bank* portfolio with sensitive information about *A*.

Analyst_B, dealing with *B* and the *Bank*, now has access to information about a competitor.



Chinese Wall: - Property

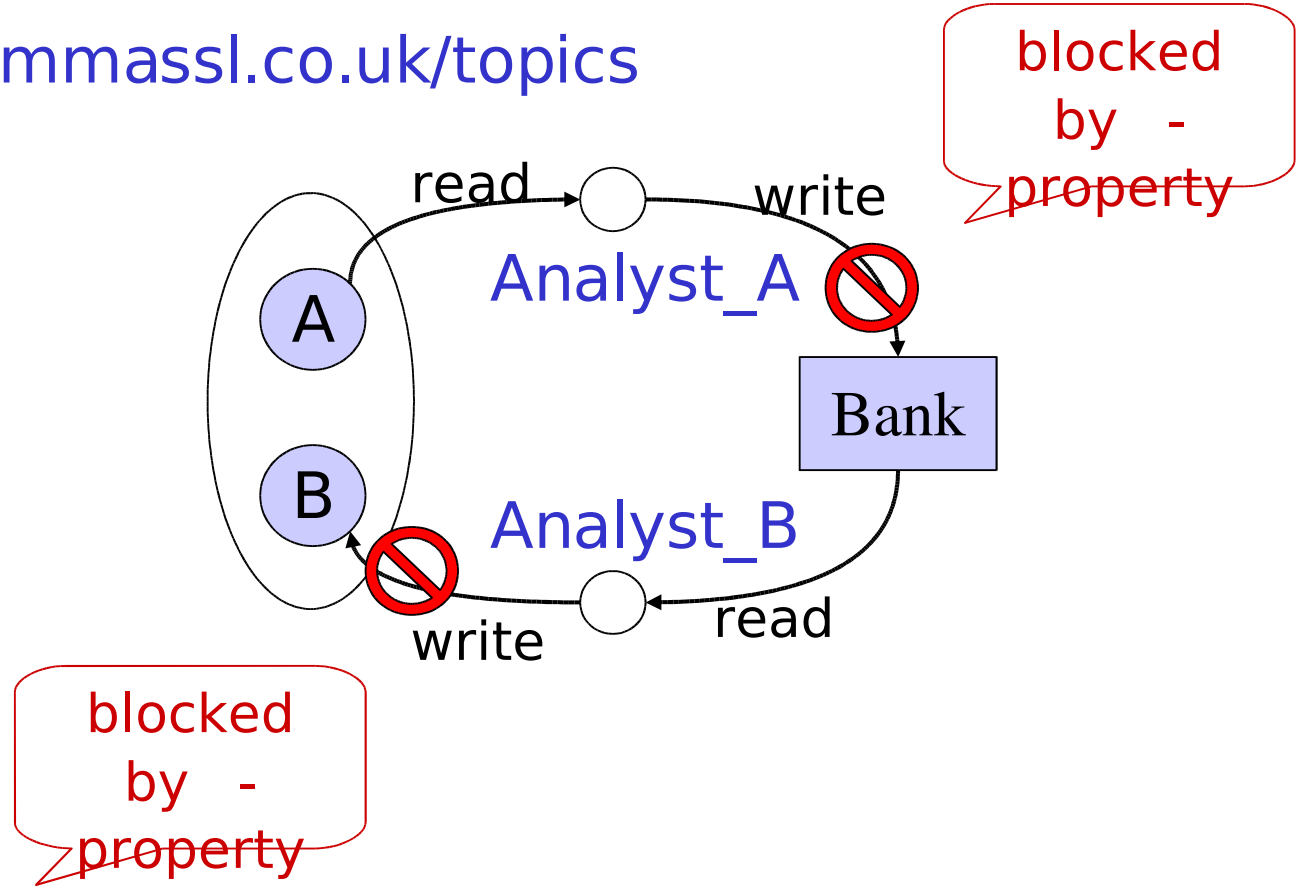
- Property: A subject *s* is permitted write access to an object only if *s* has no read access to any object *o'*, which is in a different company dataset and is unsanitized.

  – subject *s* gets write access to object *o* only if *s* has no read access to an object *o'* with *y(o)* *y(o')* or *x(o')* { }

Access rights of subjects change dynamically with every access operation.

Chinese Wall: - Property

www.gammassl.co.uk/topics



Clark-Wilson Model

Addresses security requirements of commercial applications. ‘Military’ and ‘commercial’ are shorthand for different ways of using computers.

Emphasis on integrity

- internal consistency: properties of the internal state of a system
- external consistency: relation of the internal state of a system to the outside world.

Mechanisms for maintaining integrity: well-formed transactions & separation of duties.

Clark-Wilson: Access Control

Subjects & objects are 'labeled' with programs.

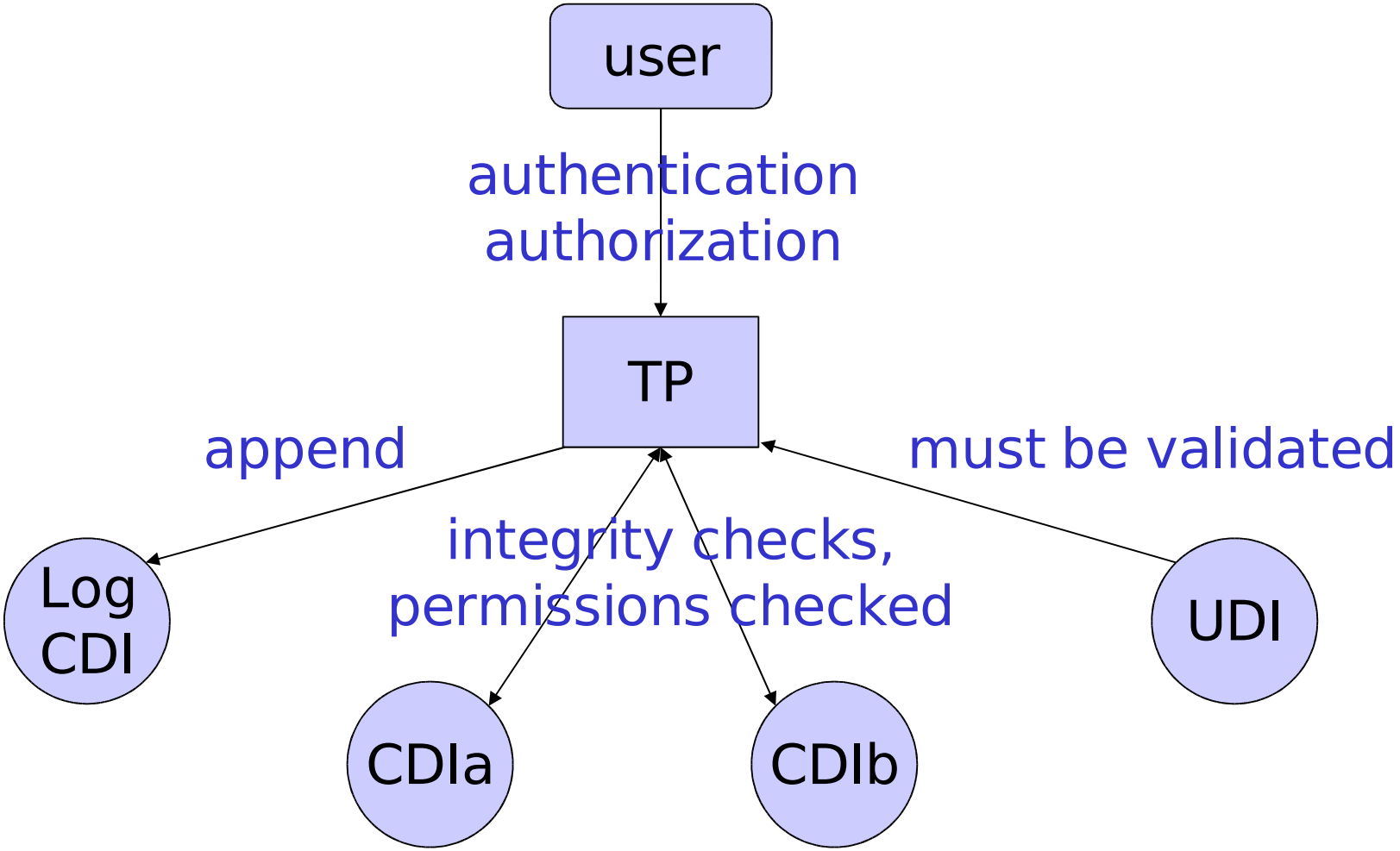
Programs serve as intermediate layer between subjects and objects.

Access control:

- define access operations (transformation procedures) that can be performed on each data item (data types).
- define the access operations that can be performed by subjects (roles).

Note the difference between a general purpose operating system (BLP) and an application oriented IT system (Clark-Wilson).

Access Control in CW



CW: Certification Rules

Five certification rules suggest how one should check that the security policy is consistent with the application requirements.

- CR1: IVPs (initial verification procedures) must ensure that all CDIs (constrained data items) are in a valid state when the IVP is run.
- CR2: TPs (transformation procedures) must be certified to be valid, i.e. valid CDIs must always be transformed into valid CDIs. Each TP is certified to access a specific set of CDIs.
- CR3: Access rules must satisfy any separation of duties requirements.
- CR4: All TPs must write to an append-only log.
- CR5: Any TP that takes an UDI (unconstrained data item) as input must either convert the UDI into a CDI or reject the UDI and perform no transformation at all.

CW: Enforcement Rules

Describe mechanisms within the computer system that should enforce the security policy:

- ER1: For each TP maintain and protect the list of entries (CDIa,CDIb,...) giving the CDIs the TP is certified to access.
- ER2: For each user maintain and protect the list of entries (TP1, TP2,...)} specifying the TPs user can execute.
- ER3: The system must authenticate each user requesting to execute a TP.
- ER4: Only subjects that may certify an access rule for a TP may modify the respective list; this subject must not have execute rights on that TP.

Interlude: Turing Machines & Decidability

Reminder

Is it better to have a very expressive policy language or a simple language where it is easier to check the impact of policy decisions?

In an expressive language, it is easier to capture the intended policy (what we think we want) but more difficult to verify that the policy actually does what it is supposed to do.

In a simple language we may not be able to capture our intentions precisely, but there are efficient means to check what the policy does

How Difficult Can It Get?

Mathematical complexity theory provides a formal framework for stating the complexity of problems.

In this framework we can give precise meanings to "easy", "difficult", or "impossible".

Having precise definitions does not imply that those definitions are "correct" and you are entitled to disagree with the mapping from informal terms to formal definitions.

How Difficult Will It Get?

It should not be necessary to understand the details of the formal models to be able to follow this lecture.

Once you start looking deeper into security you are likely to come across the concepts that will be presented so it helps if you can appreciate their meaning.

It helps to know if you are dealing with a problem that is impossible to solve.

Turing Machines

Turing machine: abstract model of a computer.

Finite set of "control" states that includes an initial state q_0 and a final state q_F.

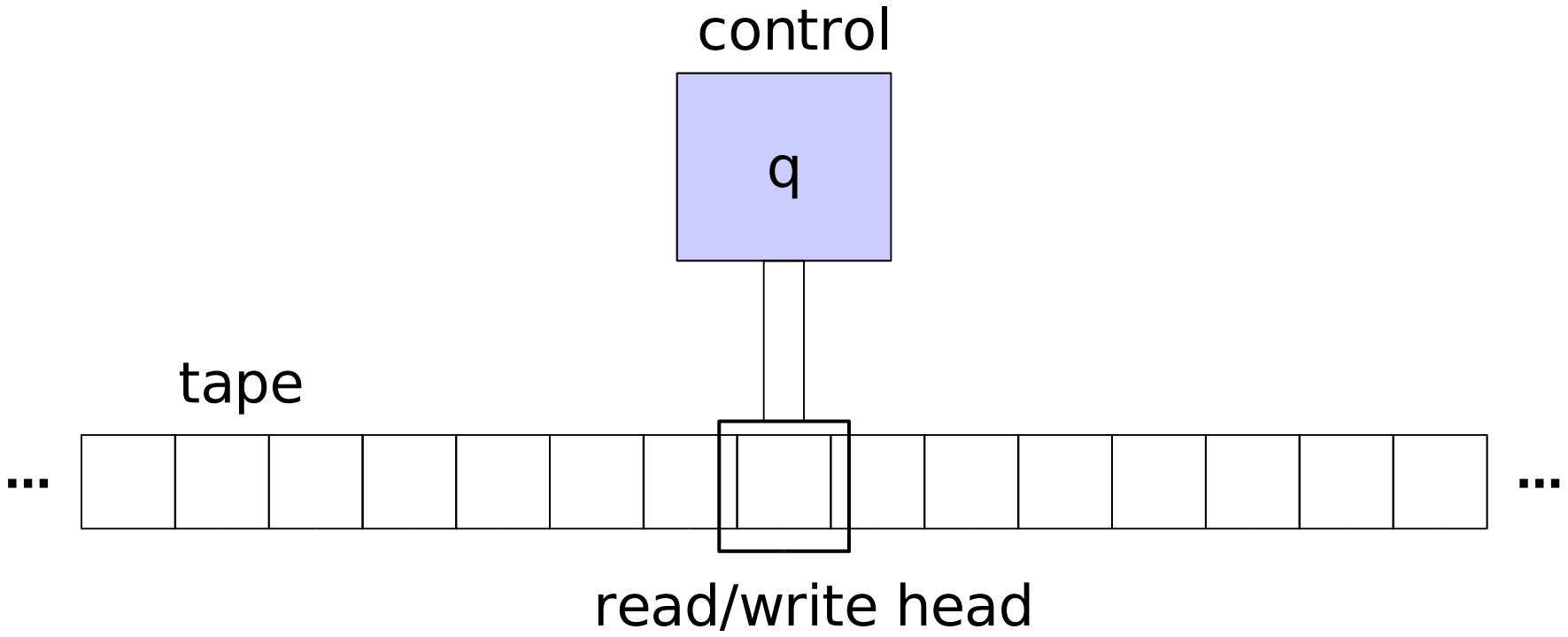
Infinite read/write tape with a read/write head.

Finite set of tape symbols.

State transition function that takes the current state and tape symbol under the read/write-head and computes.

- the next state,
- the movement of the read/write head,
- the symbol to be written to the tape (if any).

Turing Machines



Turing Machines

"Application": define a formal language; all words that can be recognized by the Turing machine belong to the language.

Words are strings of tape symbols.

To recognize a word w, load it on the tape, start the machine in its initial state q_0, and wait if it gets into its final state q_F.

When the machine gets into its final state it halts.

Will the Machine Halt?

By definition, for words in the language the TM will halt; for words not in the language, the TM will loop forever.

Can we decide whether the TM will halt?

If we can construct a second TM that recognizes the complement of the original language (halts exactly on inputs where the first TM loops) we run both TMs in parallel and know the answer once one of them halts.

Decidability

Languages where we can recognize the complement are decidable.

Otherwise, the language is undecidable.

Can we decide whether a TM will halt?

Construct a universal Turing machine (UTM):

- Input: description of a TM and an input word;
- UTM simulates the TM on its tape and halts if the TM halts on the given input.

Question: Can we build another TM that recognizes the complement of the language defined by the UTM?

Undecidability

The Halting Problem: Given an arbitrary Turing machine and an arbitrary input word, will the Turing machine halt?

Theorem (Turing): The halting problem is undecidable.

Undecidable problems are "impossible" to solve; there cannot exist an algorithm that solves all instances of the problem.

There may well be algorithms that solve all relevant instances of the problem.

“Easy” Problems

Languages where membership can be decided in polynomial time (in the size of the problem instance) fall into a class called *P*.

Problems that correspond to languages in *P* are the “easy” problems in (theoretical) computer science

In theoretical cryptography, “easy” algorithms are modelled as polynomial-time TMs.

Attraction: The combination of polynomial-time TMs is again a polynomial-time TM.

“Easy” Problems

It is still possible that such “easy” problems are difficult to solve in practice, e.g. if the problem instances are very large or if the polynomial that bounds execution time has a high degree.

Example for an easy problem: given a plaintext m, a ciphertext c, and a (guessed) key k, check whether $c = e_k(m)$.

Notation: $e_k(m)$ - encryption of m under key k.

Non-deterministic TMs

Non-deterministic Turing machines: several instances of the TM are running in parallel.

The class *NP*: languages where membership can be decided in polynomial-time by a non-deterministic TM.

Problems that correspond to languages in *NP* but not in *P* are the "difficult" problems; a brute-force approach that tries all possible solutions will eventually find the correct one.

Difficult Problems

Example for a (potentially) difficult problem: given a plaintext m, a ciphertext c, find a key k so that $c = e_k(m)$.

Making this check for each guessed key is easy, a brute-force attack searching through the entire key space is "difficult".

In practice, difficult problems may be easy to solve if problem instances are not too large or have some benign characteristics.

NP-Completeness

A problem in *NP* is *NP*-complete if an algorithm for its solution could be "easily" adapted to solve any other problem in *NP*.

NP-complete problems are the "most difficult" problems in *NP*.

The travelling salesman problem is, for example, *NP*-complete.

Even *NP*-complete problems can have instances that are easy to solve.

P = NP? Still an open problem.

Access Control

Scenario: security policy given as access control matrix; beyond read/write operations there are also operations for changing access rights (discretionary access control) and for creating new subjects and objects.

Question: Given a policy, can we answer the question "Will this particular principal ever be allowed to access this resource?"

Access rights can change so we have to do more than simply check the current access control matrix.

Harrison-Ruzo-Ullman Model

Harrison-Ruzzo-Ullman model (HRU, 1976): defines authorisation systems where we can explore answers to our question.

Components of the HRU model:

- set of subjects S
- set of objects O
- set of access rights R
- access matrix $M = (M_{so})_{s \in S, o \in O}$: entry M_{so} is a subset of R defining the rights subject s has on object o

Primitive Operations in HRU

Six primitive operations for manipulating subjects, objects, and the access matrix:

- enter $\underline{r}$ into M_{so}
- delete $\underline{r}$ from M_{so}
- create subject s
- delete subject s
- create object o
- delete object o

Commands in HRU model (examples):

command create_file(s,f)
    create f
    enter $\underline{o}$ into $M_{s,f}$
    enter $\underline{r}$ into $M_{s,f}$
    enter $\underline{w}$ into $M_{s,f}$
end

command grant_read(s,p,f)
    if $\underline{o}$ in $M_{s,f}$
    then enter $\underline{r}$ in $M_{p,f}$
end

HRU Policies

Policy management question: Is this subject allowed to access this object?

The HRU access matrix describes the state of the system; commands effect changes in the access matrix.

HRU can model policies for allocating access right; to verify compliance with a given policy, you have to check that no undesirable access rights can be granted.

‘Leaking’ of Rights in HRU

An access matrix *M* is said to leak the right *r* if there exists a command *c* that adds *r* into a position of the access matrix that previously did not contain *r*.

M is safe with respect to the right *r* if no sequence of commands can transform *M* into a state that leaks *r*.

Do not expect the meaning of ‘leak’ and ‘safe’ to match your own intuition.

Safety Properties of HRU

The safety problem cannot be tackled in its full generality.

Theorem. Given an access matrix *M*, a right *r*, and a set of commands, verifying the safety of *M* with respect to *r* is undecidable.

There does not exist a general algorithm that answers our policy question for all instances of the HRU model.

For restricted models, the chances of success are better.

Restricted Models

Mono-operational commands contain a single operation:

Theorem. Given a mono-operational authorisation system, an access matrix *M*, and a right *r*, verifying the safety of *M* with respect to *r* is decidable.

With two operations per command, the safety problem is undecidable.

Limiting the size of the authorisation system also makes the safety problem tractable.

Theorem. The safety problem for arbitrary authorisation systems is decidable if the number of subjects is finite.

HRU – Summary

Do not memorize the details of HRU.

Do not memorize the individual undecidability theorems for the HRU variants.

Remember the important message: The more expressive the security model, the more difficult it is to verify security.

You don't have to know the basics of computational complexity theory for this course but it helps to appreciate the challenges in formally verifying security.

Information Flow Models

Similar framework as BLP: objects are labeled with security classes (form a lattice), information may flow upwards only.

Information flow described in terms of conditional entropy (equivocation information theory)

Information flows from *x* to *y* if we learn something about *x* by observing *y*:

- explicit information flow: *y:= x*
- implicit information flow: IF *x=0* THEN *y:=1*
- covert channels

Proving security is undecidable.

Non-interference Models

A group of users, using a certain set of commands, is non-interfering with another group of users if what the first group does with those commands has no effect on what the second group of users can see.

Take a state machine where low users only see outputs relating to their own inputs. High users are non-interfering with low users if the low users see the same no matter whether the high users had been providing inputs or not.

Active research area in formal methods.

Execution Monitors

Security Policies (again)

Three classes of security policies:

Access control: restricts what operations principals can perform on objects.

Information flow: restricts what principals can infer about objects from observing system behaviour.

Availability: restrict principals from denying others the use of a resource.

Execution Monitoring

The practicality of a security policy depends on whether it is enforceable and at what cost.

Execution Monitoring (EM): enforcement mechanisms that monitor execution steps of a target system and terminate the target's execution if it is about to violate the security policy being enforced.

EM includes security kernels, reference monitors, firewalls, most other operating system, …

Beyond EM

Enforcement mechanisms that use more information than is available only from observing the steps of a target's execution only are outside EM.

Information provided to an EM mechanism is insufficient to predict future steps the target might take, alternative possible executions, or all possible target executions.

Compilers and theorem-provers that analyze a static representation of a target to deduce information about all of its possible executions are not EM mechanisms.

Beyond EM

Mechanisms that modify a target before executing it are also outside EM.

The modified target must be equivalent to the original, except for aborting executions that violate the security policy of interest.

A definition of equivalence is thus required to analyze this class of mechanisms.

In-line reference monitors and reflection techniques fall in this category.

Executions & Properties

We are interested in the executions of a target system.

Executions are sequences of steps, e.g. machine instructions.

We use Ψ to denote the set of all executions, finite and infinite, of our target system

Let $\sigma\ [\ ..i\ ]$ denote the first i steps of σ.

A set Γ of executions is called a property if membership of an element is determined by the element alone, not by other elements.

EM & Properties

A security policy must be a property to have an enforcement mechanism in EM.

Not every security policy is a property.

Some security policies cannot be defined as a predicate on individual executions.

Information flow policies: information flows from "high" to "low" if a low user can somehow detect actions by a high user.

We have to compare executions where the high user is active with executions where the high user is inactive.

EM & Properties

Not every property is EM enforceable.

Enforcement mechanisms in EM cannot look into the future when making decisions on an execution.

Consider an execution that reaches a state that satisfies the security policy but goes through "insecure" states

An EM has to prohibit such an insecure prefix of a secure execution.

Safety & Liveness

In the HRU model, we were talking about "safe" access matrices.

In discussions about security you may find further references to safety & liveness.

Safety properties: nothing bad can happen.

Liveness properties: something good will happen eventually.

A property Γ is called a safety property if we have for every finite or infinite execution σ

$$\sigma \notin \Gamma \Rightarrow (\exists i : (\forall \tau \in \Psi : \sigma[..i]\tau \notin \Gamma)).$$

EM & Safety

Non EM-Enforceable Security Policies: If the set of executions for a security policy is not a safety property, then that policy does not have an enforcement mechanism from EM.

EM enforcement mechanisms enforce security policies that are safety properties.

It is not the case that all safety properties have EM enforcement mechanisms.

Safety & Security Policies

Access control defines safety properties: partial executions that end with attempting an unacceptable operation will be prohibited.

Information flow does not define sets that are properties; so information flow cannot be a safety property and in turn cannot be enforced by EM.

Availability is not a safety property: any partial execution can be extended in a way that allows a principal to access the resource.

Availability defined in respect to a Maximum Waiting Time (MWT) is a safety property; once an execution has waited beyond MWT, any extension will also wait beyond MWT.

The 3rd Design Principle

If you design complex systems that can only be described by complex models, finding proofs of security becomes difficult.

In the worst case (undecidability), no universal algorithm exists that verifies security in all cases.

If you want verifiable security properties, you are better off with a security model of limited complexity.

Such a model may not describe all desirable security properties, but you may gain efficient methods for verifying 'security'.

In turn, you are advised to design simple systems that can be adequately described in the simple model.

The more expressive a security model is, both with respect to the security properties and the systems it can describe, the more difficult it is usually to verify security properties.

Summary

The theoretical foundations for access control are relevant in practice.

It helps to know in which complexity class your policy language and enforcement algorithm put you in.

Powerful description languages may leave you with undecidable enforcement problems.

Much of current efforts on policy languages in ‘trust management’ and web services access control revolves around these issues.

Further Reading

D.F.C. Brewer and M.J. Nash, The Chinese Wall Security Policy, Proceedings of the 1989 IEEE Symposium on Security and Privacy, pages 206-214, 1989

Clark, D.R. and Wilson, D.R., A Comparison of Commercial and Military Computer Security Policies, Proceedings of the 1987 IEEE Symposium on Security and Privacy, pages 184-194, 1987

Goguen, J.A. and Meseguer, J., Security Policies and Security Models, Proceedings of the 1982 IEEE Symposium on Security and Privacy, pages 11-20, 1982

Fred B. Schneider: Enforceable security policies: ACM Transactions on Information System Security, Vol. 3, No. 1, S. 30-50, 2000.

ESORICS 2000 (Springer Lecture Notes in Computer Science 1895): Checking secure interactions of Smart Card Applets and Verification of a Formal Security Model for Multiapplicative Smart Cards