Information Security Group

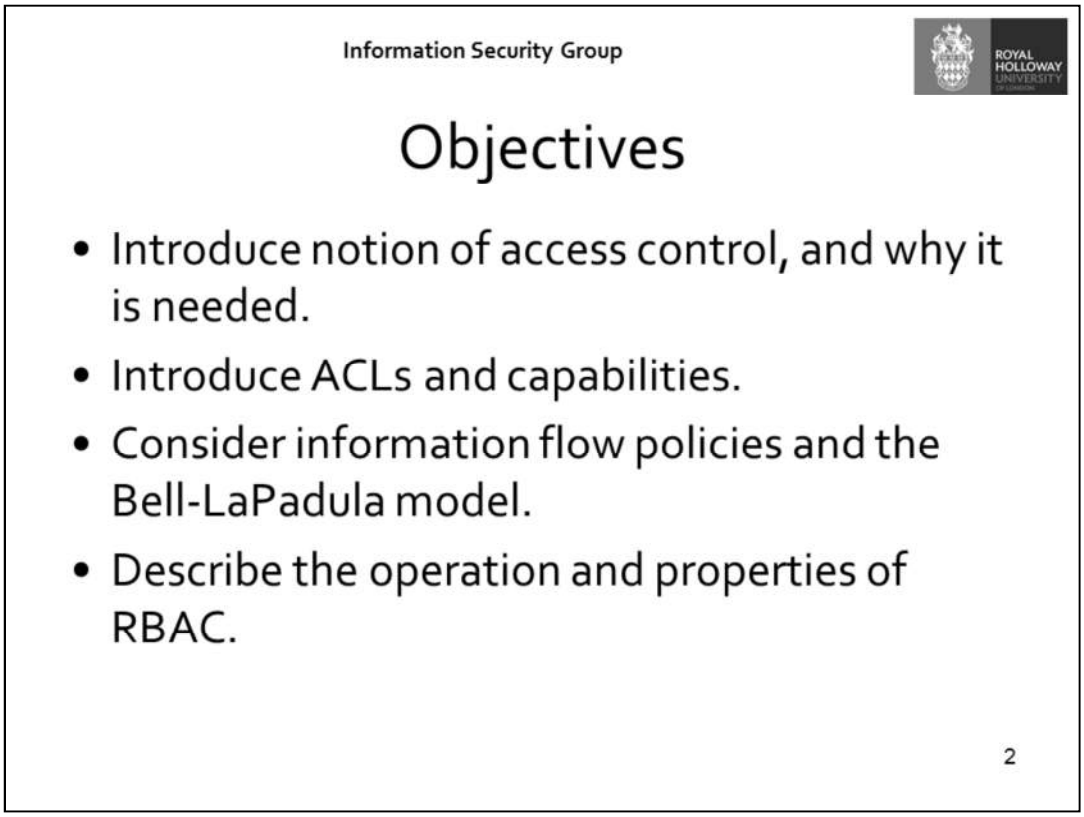
IY5512 Computer Security
Part 6: Authorisation

Chris Mitchell

me@chrismitchell.net

http://www.chrismitchell.net

1



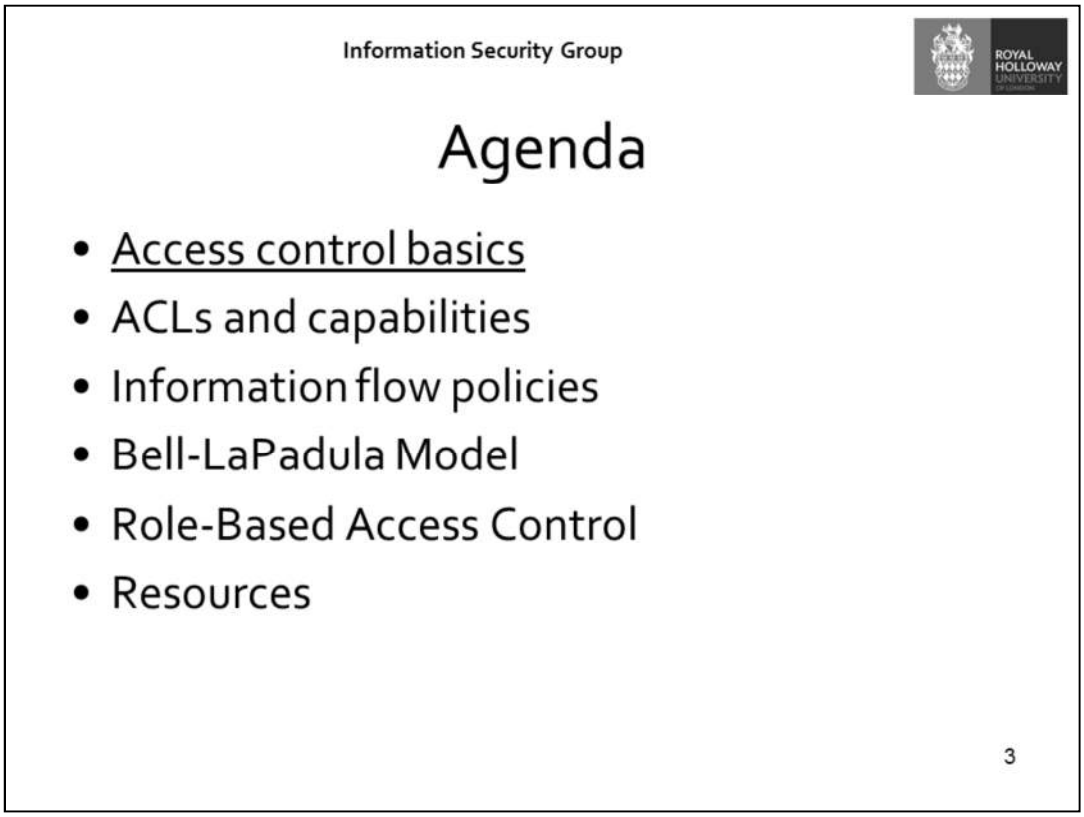
This part of the course addresses the following topics.

• We introduce the notion of an access control system, and why access control is needed.

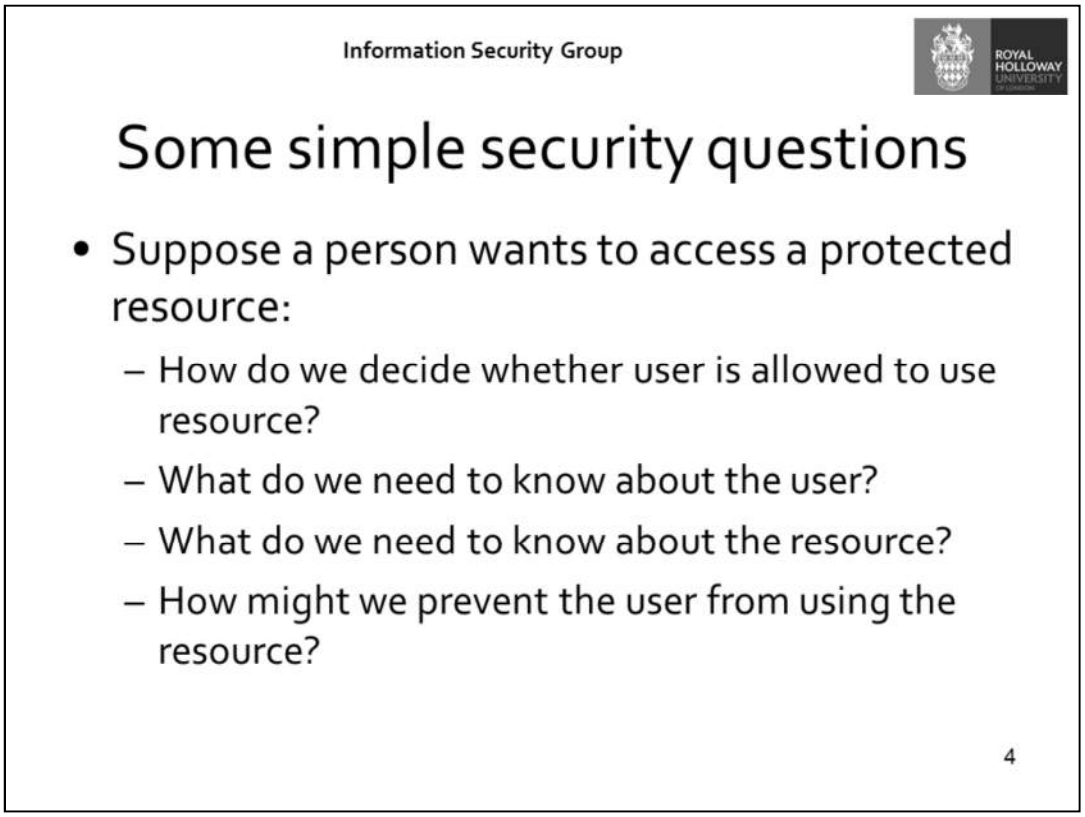
• We introduce fundamental access control models, including Access Control Lists (ACLs) and capabilities.

• We consider information flow policies and the Bell-LaPadula model.

• We describe the operation and properties of Role-Based Access Control (RBAC).

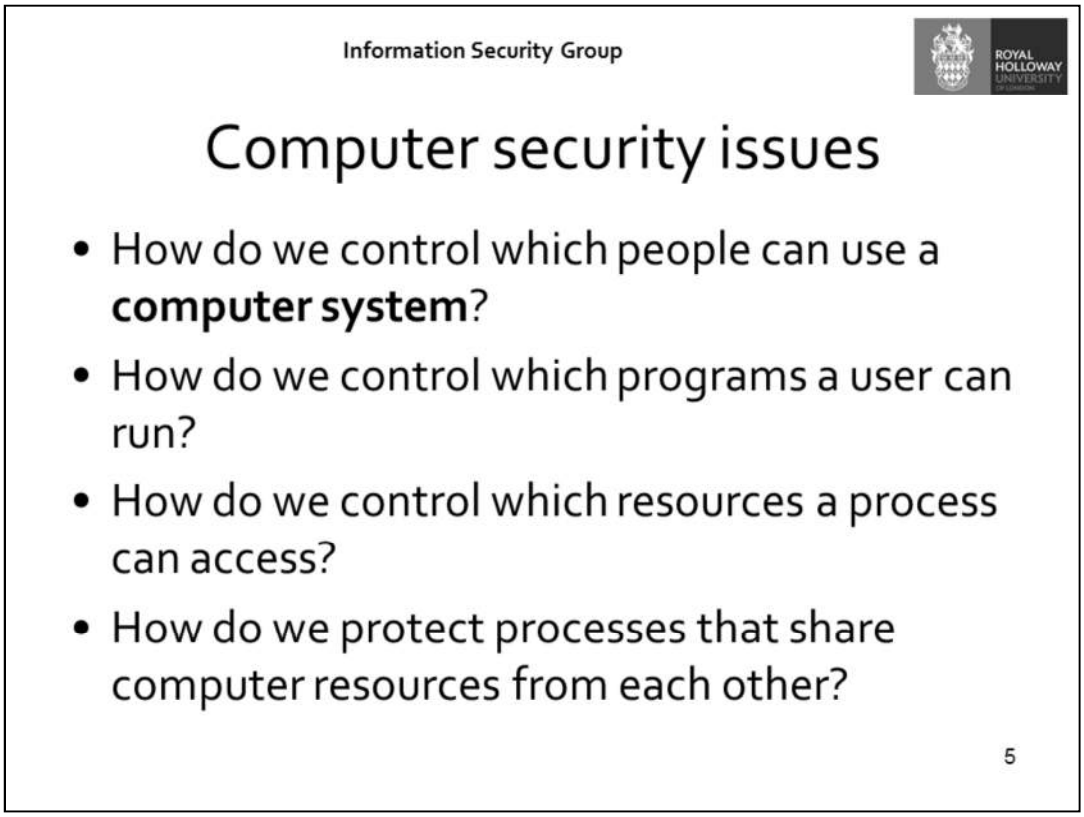


We start with the basics …



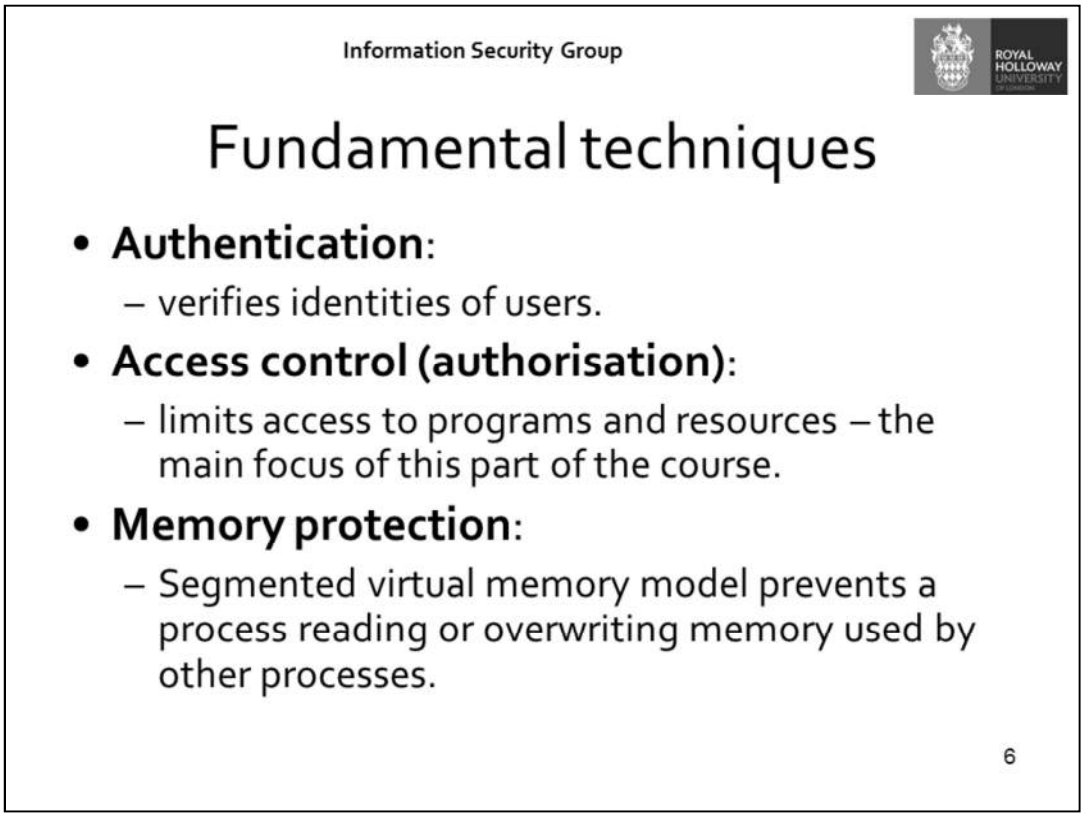
Suppose that a (process acting on behalf of a) human being wants to access a protected resource on a computer. Some fundamental questions need to be answered.

- How should we decide whether that user is allowed to use the resource?
- What do we need to know about the user?
- What do we need to know about the resource?
- How might we prevent the user from using the resource?



In a computer system, the following issues need to be addressed:

- How do we control which people can use the computer system?
- How do we control which programs a user can run?
- How do we control which resources a process can access?
- How do we protect processes that share computer resources from each other?

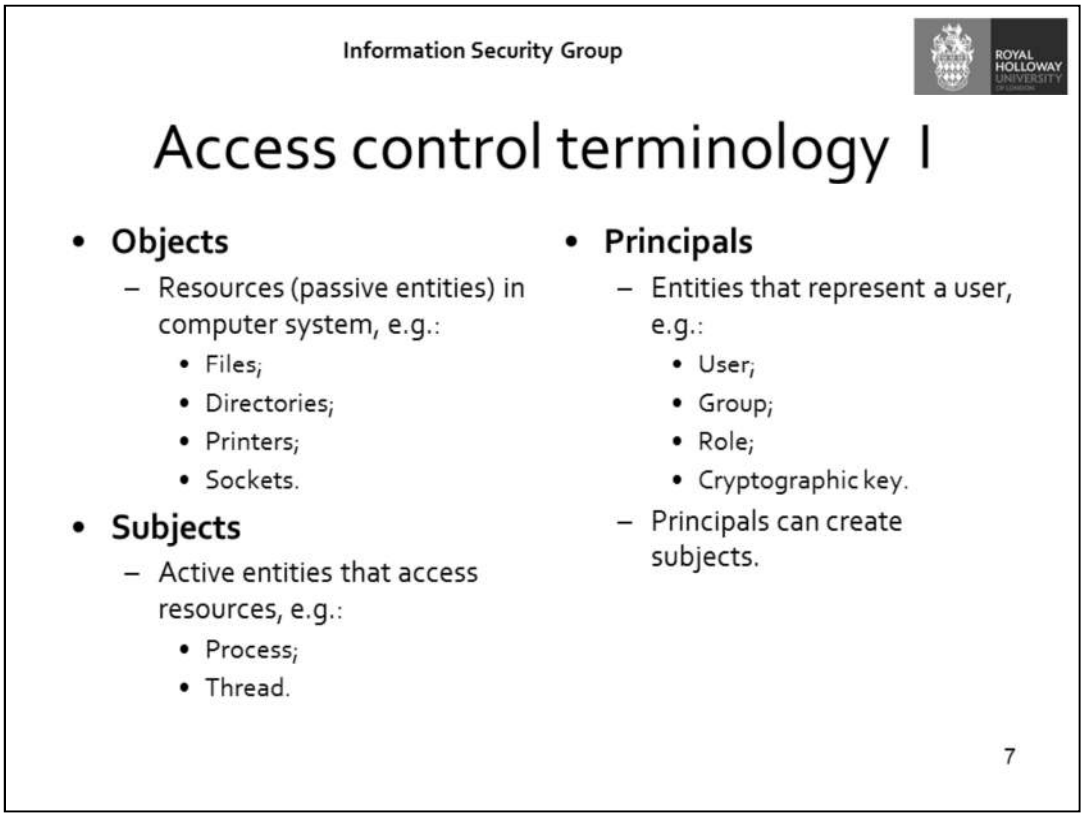


The following techniques can be used to help control access to resources:

• **Authentication** – corroborates (verifies) the identities of users, as discussed in the previous part of the course.

• **Access control** (authorisation) – limits access to programs and resources to those authorised to have access.

• **Memory protection** – a segmented virtual memory model prevents a process reading or overwriting memory used by other processes.

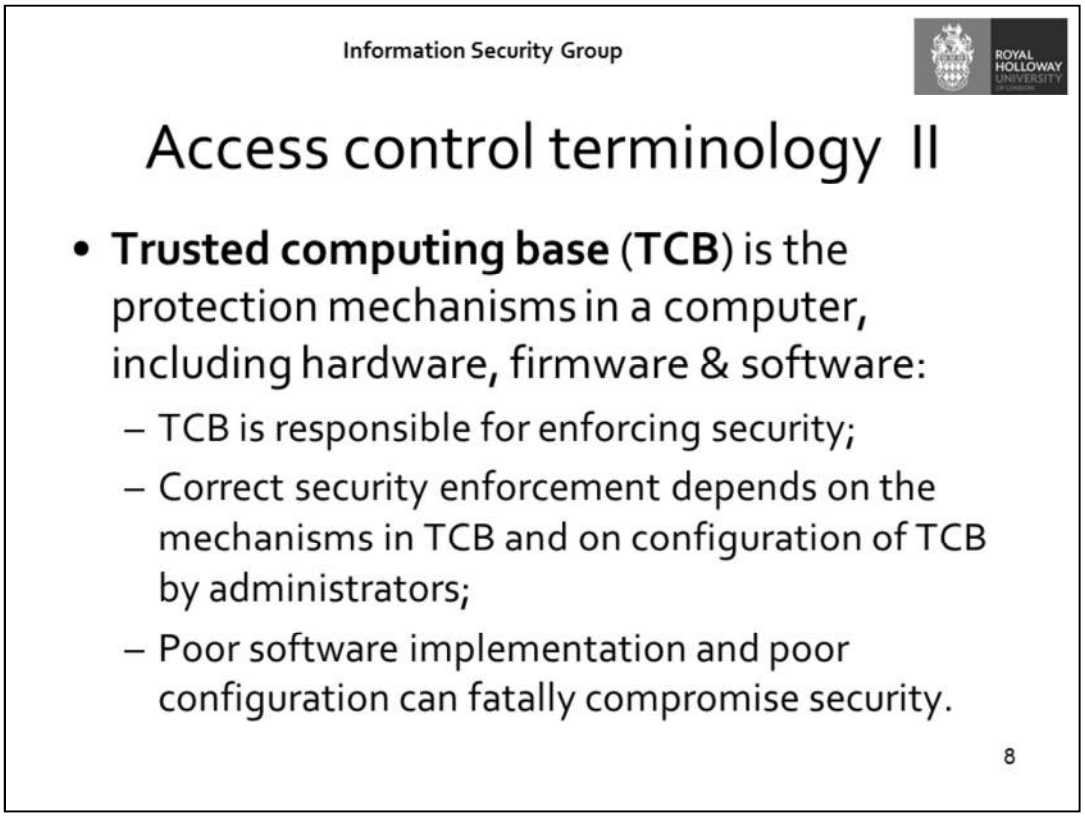


We use the following terminology when discussing access control systems. There are three fundamental types of entity in an access control system:

• **Subjects** – i.e. active entities that wish to access resources (subjects), e.g. processes or threads.

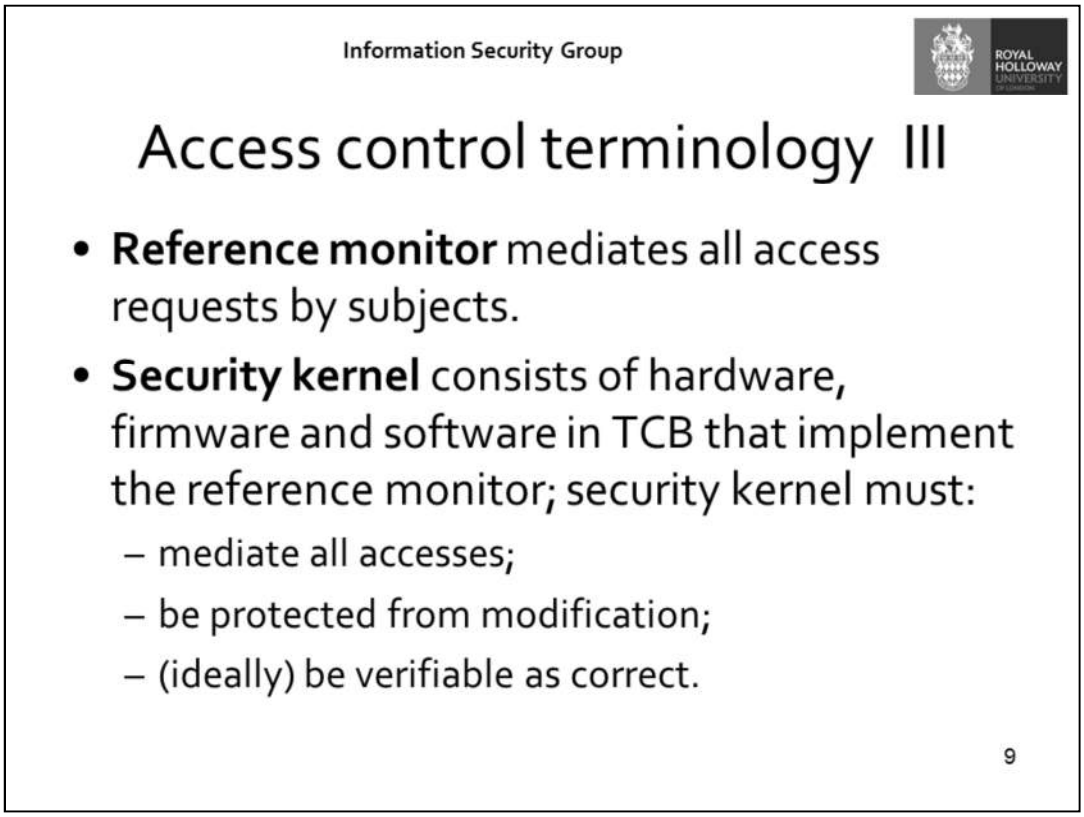
• **Objects** – i.e. resources which subjects may wish to access, e.g. files, directories, printers, sockets, networks, …

• **Principals** – i.e. entities that represent a human user. (We can think of them as the users although, of course, users are not part of the computer system). Typically subjects act on behalf of principals, and access control decisions are made on the basis of the identities of the object and the principal.



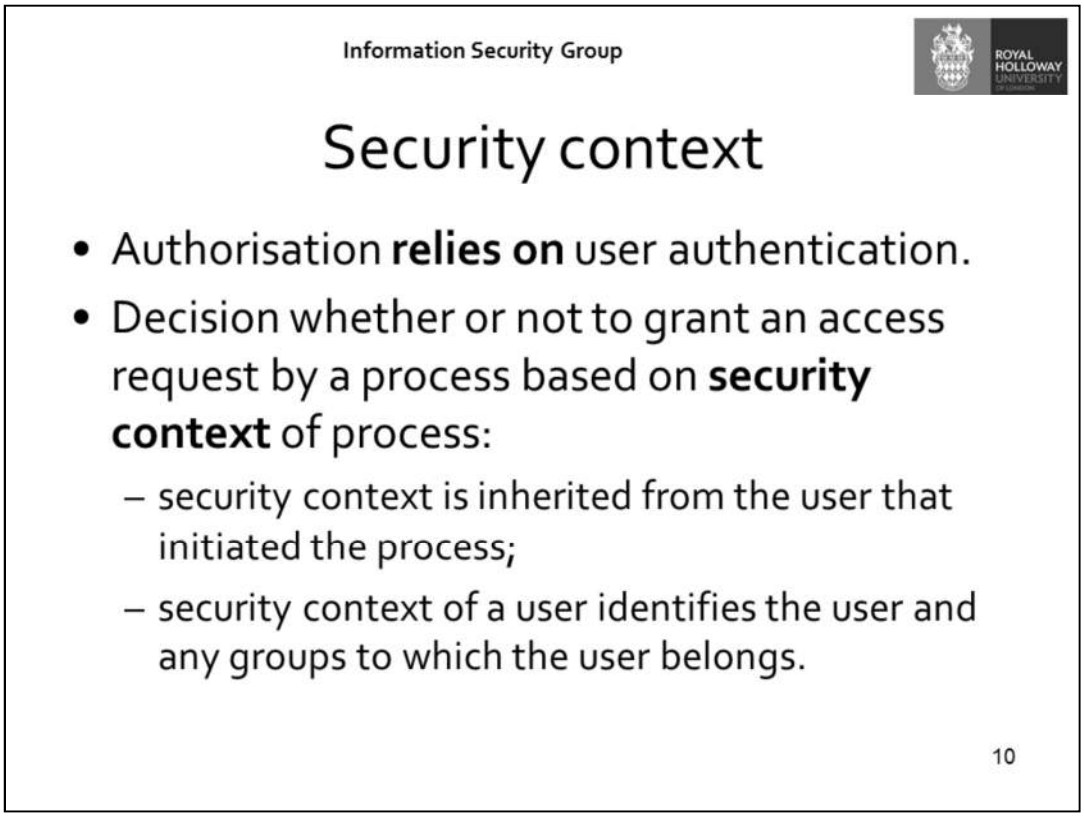
The **trusted computing base (TCB)** is a general term for all the protection mechanisms within a computer system, including the hardware, firmware and software.

The TCB is responsible for enforcing security (in the form of an implicit or explicit **security policy**). The ability of the TCB to correctly enforce a security policy depends on the mechanisms within the TCB and on the correct configuration of the TCB by administrators. Clearly, poor software implementation and poor configuration can fatally compromise security.



The **reference monitor** is an abstract concept used to describe the functionality that mediates all access requests by subjects. That is, the reference monitor is the part of a system responsible for authorisation (access control) checks.

The **security kernel** consists of all the hardware, firmware and software elements of a TCB that are responsible for implementing the reference monitor. The security kernel must mediate all attempts by subjects to access objects (resources), be protected from modification, and (ideally) be verifiable as correct.



Authorisation (access control) depends on robust user authentication.

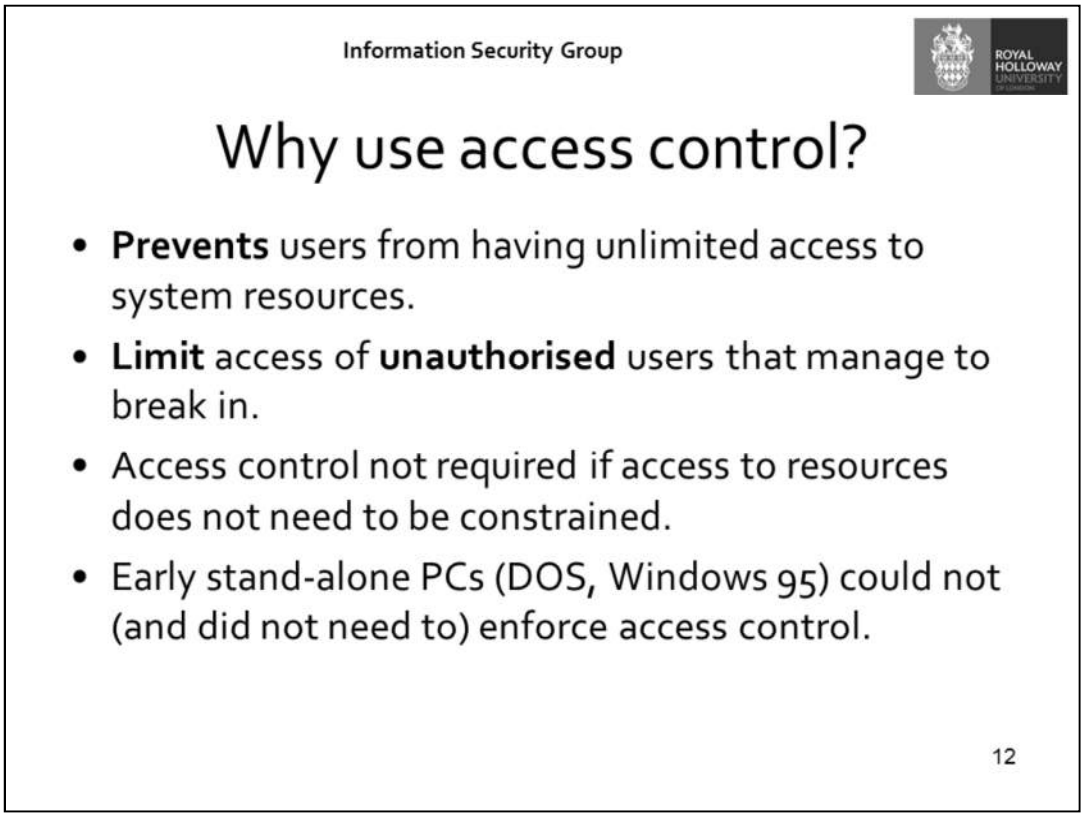
The decision to grant an access request made by a process is based on the security context of the process. The security context is inherited from the user that initiated the process. The security context of a user usually identifies the user and any security groups to which the user belongs.

Hence if the authentication process is broken, then the access control system is also effectively broken, since the security context will not be reliable.

Access control is a generic term for the process(es) by which a computer system controls the interaction between users and system resources.

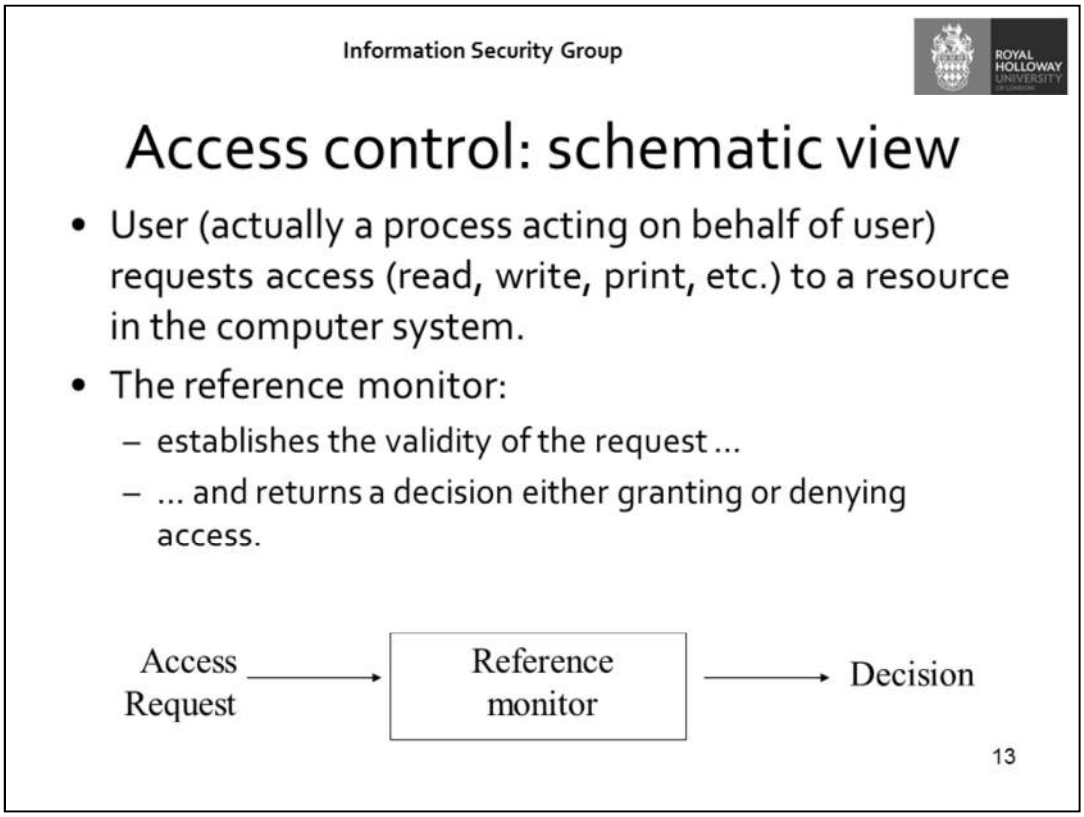
An access control system may implement (part of) a specific security policy. This policy may be determined by both organisational requirements and statutory requirements (e.g. covering personally identifiable information (PII), such as medical records).

Policy requirements relevant to access control include confidentiality (restrictions on read access) and integrity (restrictions on write access). Other requirements typically addressed by access control systems in practice include ‘executability’, i.e. restrictions on executing a file as a program.

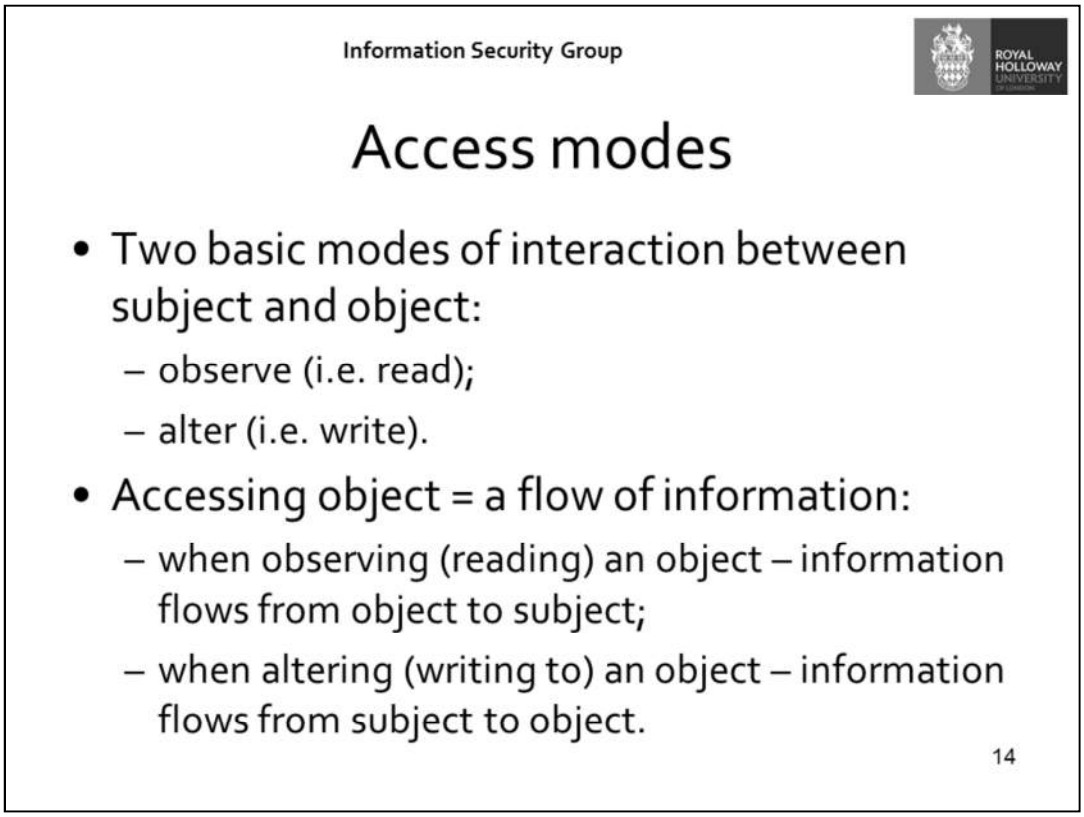


An access control system **prevents** users from having unlimited access to system resources, and helps to limit the access of **unauthorised** users that manage to break in.

Of course, an access control system is not required if access to resources does not need to be constrained. Early stand-alone PCs (DOS, Windows 95) could not (and arguably did not need to) enforce access control, since the (implicit) security model for the PC was that of a single user, and all the resources on the PC belonged to that user.



A high-level view of an access control system involves a user (actually, a process acting on behalf of a user) requesting access (read, write, print, etc.) to a resource in the computer system. The reference monitor establishes the validity of the request, and returns a decision either granting or denying access to the process.

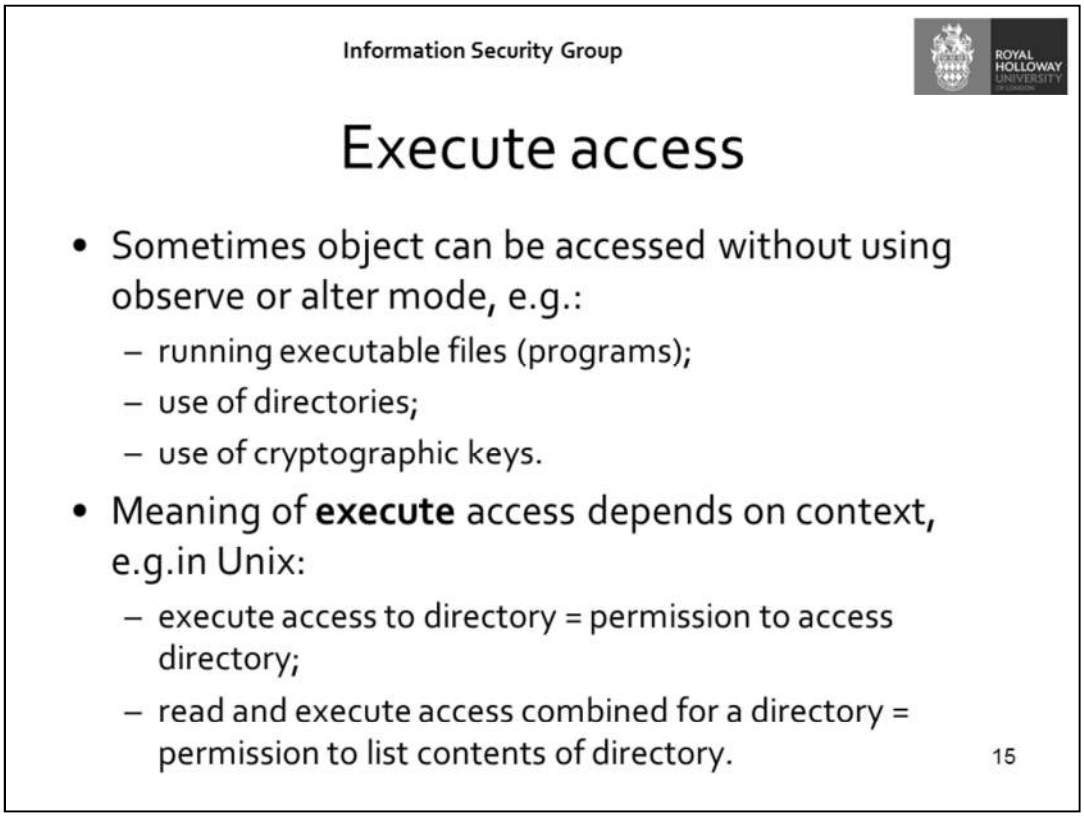


There are two fundamental modes of interaction between a subject and an object, namely **Observe** and **Alter**.

Accessing an object can be regarded as initiating a flow of information.

• A subject may observe (read) an object. In this case information flows from the object to the subject.

• A subject may alter (write to) an object. In this case information flows from the subject to the object.

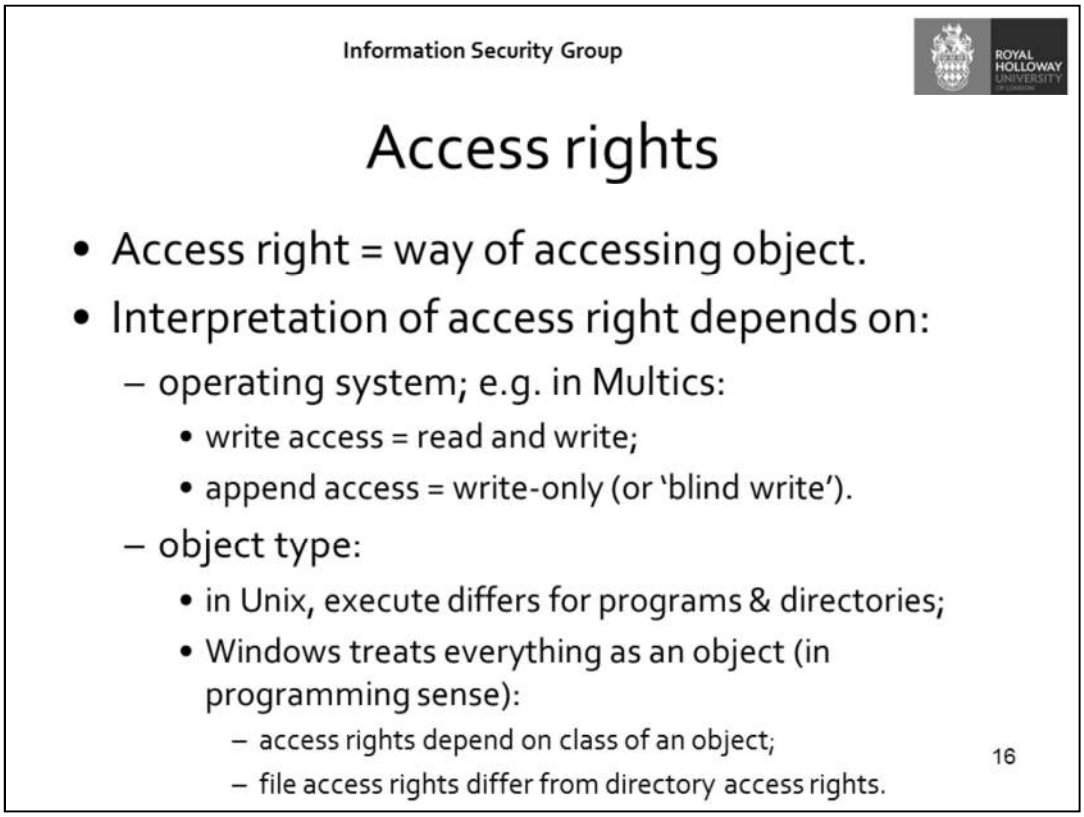


Sometimes an object can be accessed without using either observe or alter mode. Examples of this include executable files (programs), directories, and cryptographic keys.

The **execute** access right means different things in different contexts and in different systems:

In Unix:

- to execute a file both `r` and `x` must be granted;
- to enter a directory (i.e. to access a file in a directory), `x` must be granted;
- to list the contents of a directory `r` and `x` must be granted;
- to create a file in a directory `w` and `x` must be granted.



Access rights define particular ways of accessing an object. The interpretation of an access right may differ between operating systems. In Multics, write access allows a subject to both read and write, and append access is write-only (or 'blind write").

The interpretation of an access right may depend on the object to which it applies:

• as noted on the previous slide, execute means something different in Unix when applied to programs and directories;

• Windows treats everything as an object (in the programming sense). Access rights are dependent on the class to which an object (in the access control sense) belongs. File access rights are different from directory access rights etc.

Differing terminology is also used for the notion of access right – e.g. right, permission, operation.

Certain operations on objects are administrative in nature, and are controlled by separate administrative access control rules. These operations involve, for example:

- changing access control data structures;
- changing the access rights of a user for a particular resource.

Control over operations which change access rights is often related to **ownership** of the resource. A special command can be used to change the access rights for an object (`chmod` in Unix), or by granting control access rights and privileges to users (change permission and take ownership privilege in Windows).

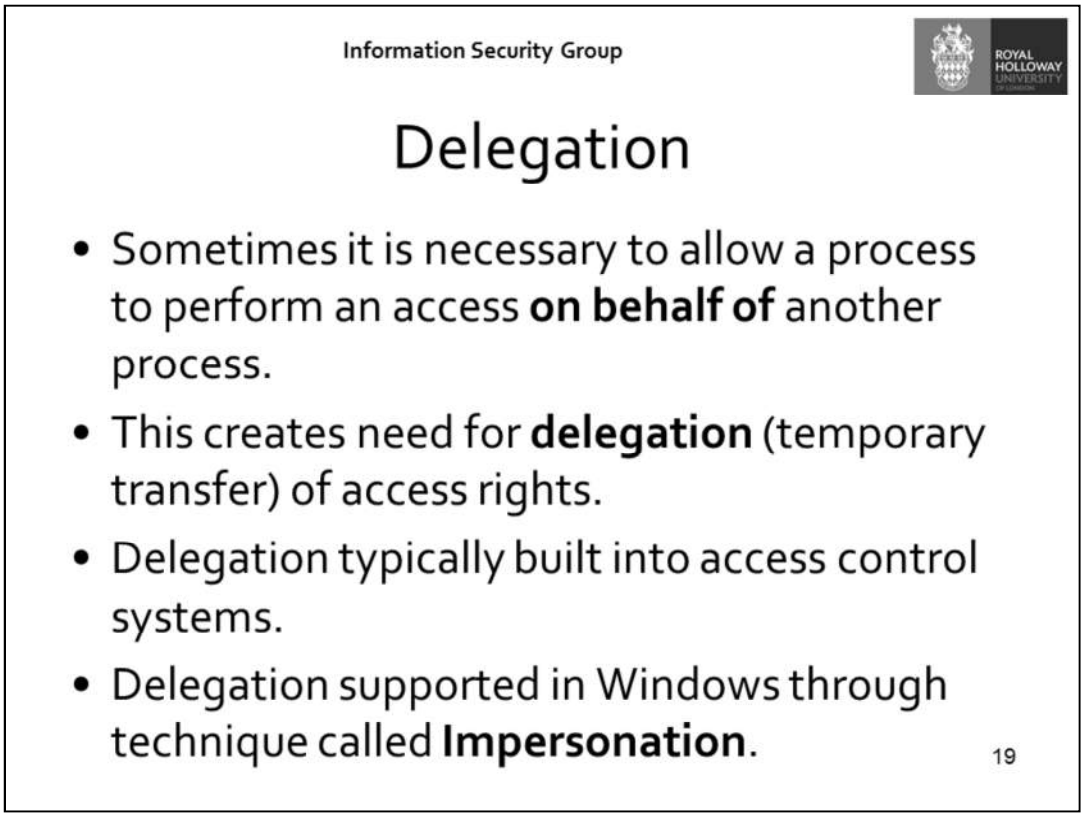
Access control mechanisms typically exist to enforce access control **policies**. Two fundamental types of access control can be identified.

Discretionary access control policies are based on identities (or other characteristics of users) – the term is used because access is **at the discretion** of the owner of a resource.

- Ownership of resources is typically important;
- Unix access control enforces such a policy;
- The use of discretionary access control is common in commercial systems.

Mandatory access control policies are independent of user identities:

- The characteristics of resources are important;
- Access is only allowed if user and object belong to same security domain;
- Common in government/military systems.

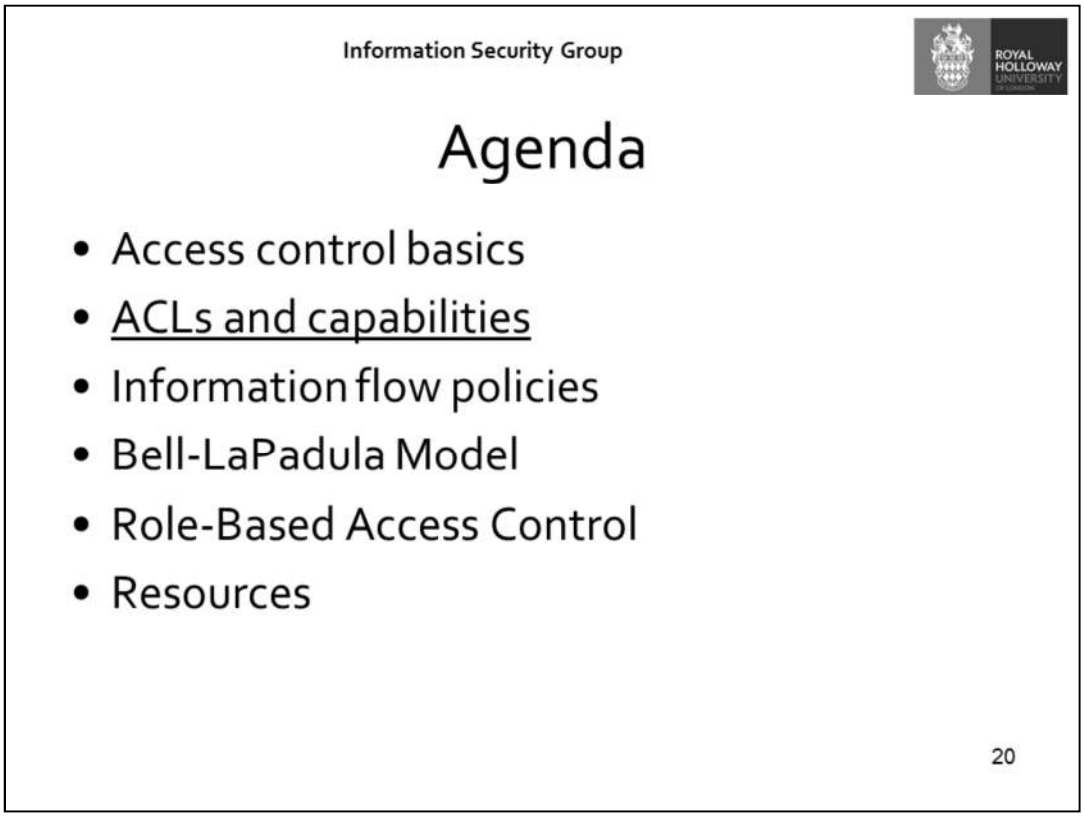


Sometimes it is necessary to allow one process to perform an access to a resource **on behalf of** another process.

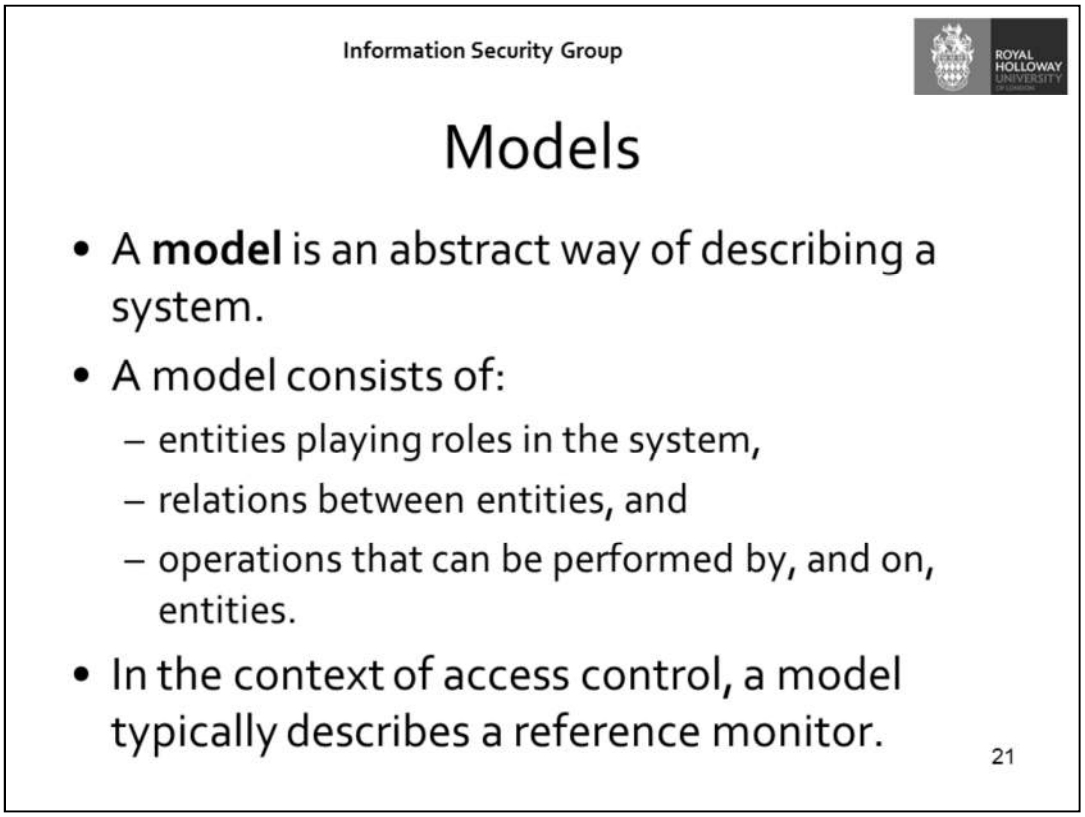
This creates a requirement for **delegation** (i.e. temporary transfer) of access rights.

Functionality for delegation is typically incorporated into access control systems.

Delegation is supported in Windows through a technique called **Impersonation**. [This is discussed in part 7b of this course].



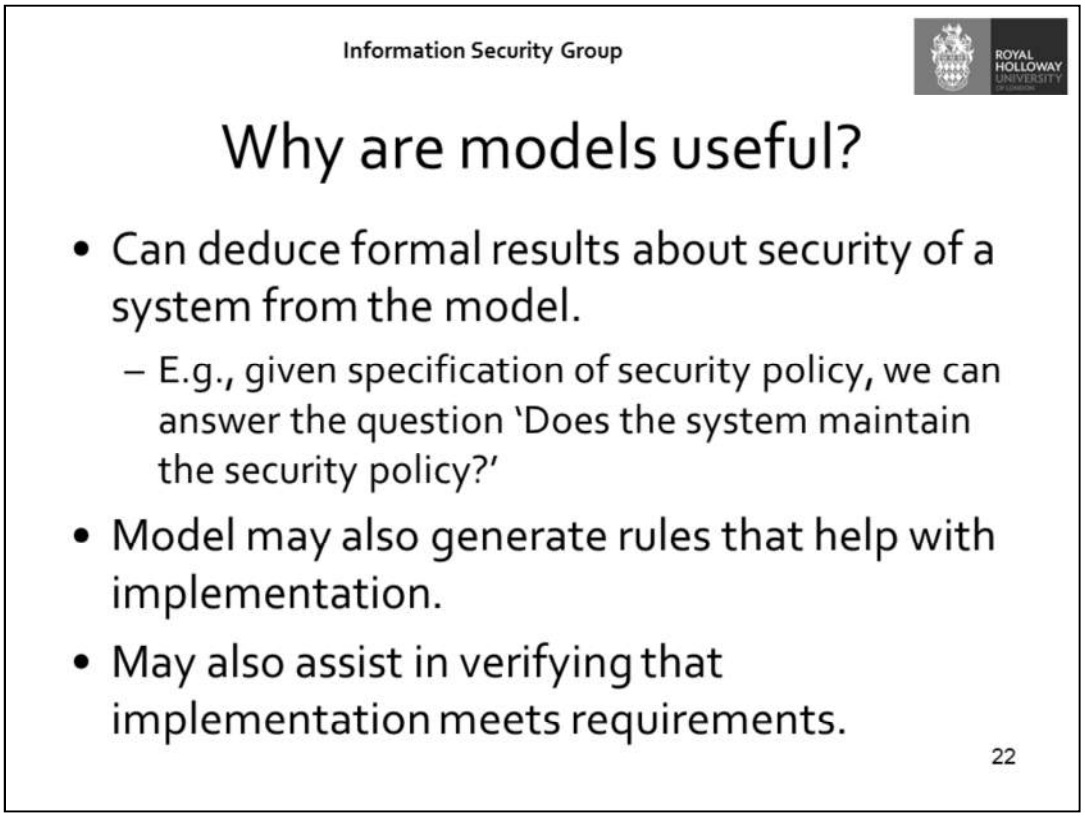
We next look at some fundamentally important models for access control. Some of these models are widely used in practice.



In general, a **model** is an abstract way of describing a system. A model is made up of elements that are used to represent the system, specifically entities (active and passive) that play a role in the system, relations between entities, and operations that can be performed by and on entities.

In the context of access control, a model typically describes a reference monitor.

One definition of an access control model is: A '*model has the ability to **represent abstractly** the elements of computer systems and of security that are relevant to a treatment of classified information stored in a computer system*' [due to Bell and LaPadula, 1976].



It might be possible to deduce formal results from the model that make statements about the security of the system.

For example, given a specification of security policy, we can answer the question 'Does the system maintain the security policy?'

A model may also generate rules that can provide a blueprint for an implementation. It also may assist in verifying that an implementation meets requirements.

Information Security Group

Access control matrix I

- Simplest access control model.
- Introduced by Lampson (1972) and extended by Harrison, Ruzzo and Ullman (1976-8):
  - Columns indexed by objects;
  - Rows indexed by principals;
  - Matrix entries are (sets of) access operations;
- Foundation of many theoretical security models.

Principals \ Objects	trash	a.out	allfiles.txt
jason	{r,w}	{r,w,x}	{r,w}
mick		{r,x}	{r}

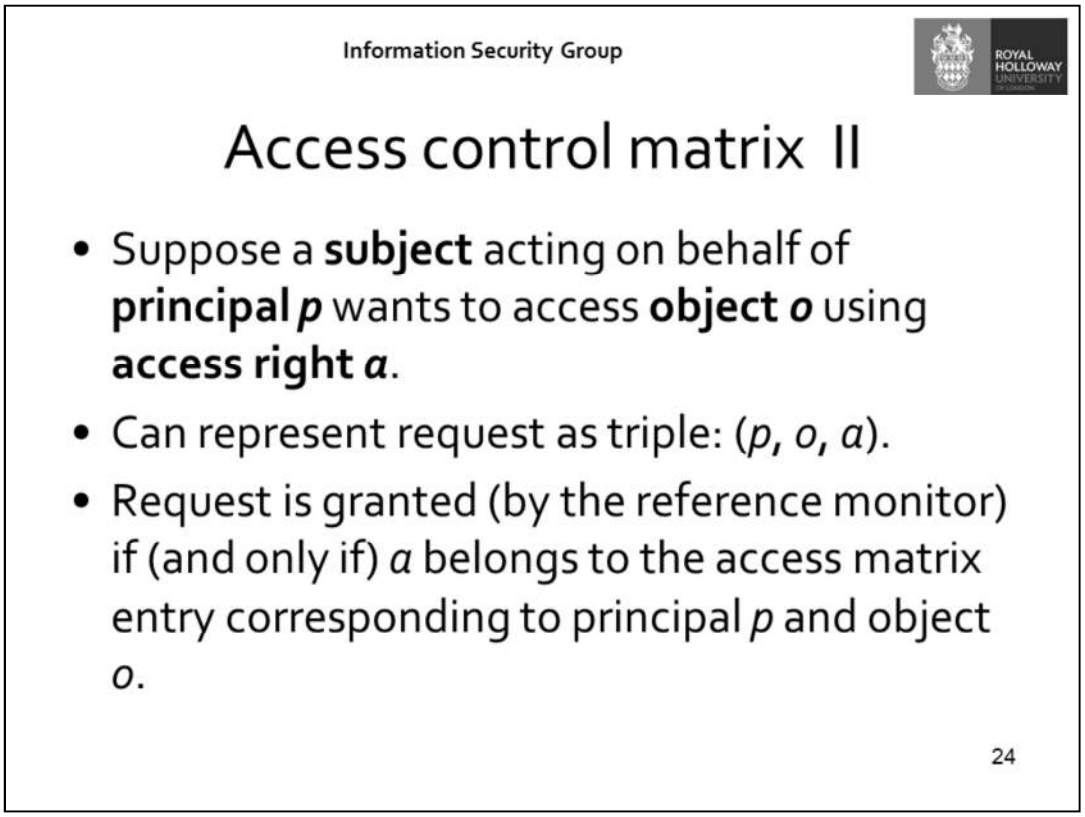
23

The simplest access control model is probably the access control matrix. The notion of an access control matrix was introduced by Lampson (1972) and was extended by Harrison, Ruzzo and Ullman (1976-8). In an access control matrix the columns are indexed by objects and the rows are indexed by the principals. The matrix entries are (sets of) access operations which a subject, acting on behalf of the specified principal, may perform on the object.

This model is the foundation of many theoretical security models.

In the example matrix (for two principals and three objects), principal Jason has read and write access to the 'trash' object, but principal Mick has no access rights to this object. Jason has read, write and execute access to the a.out object.

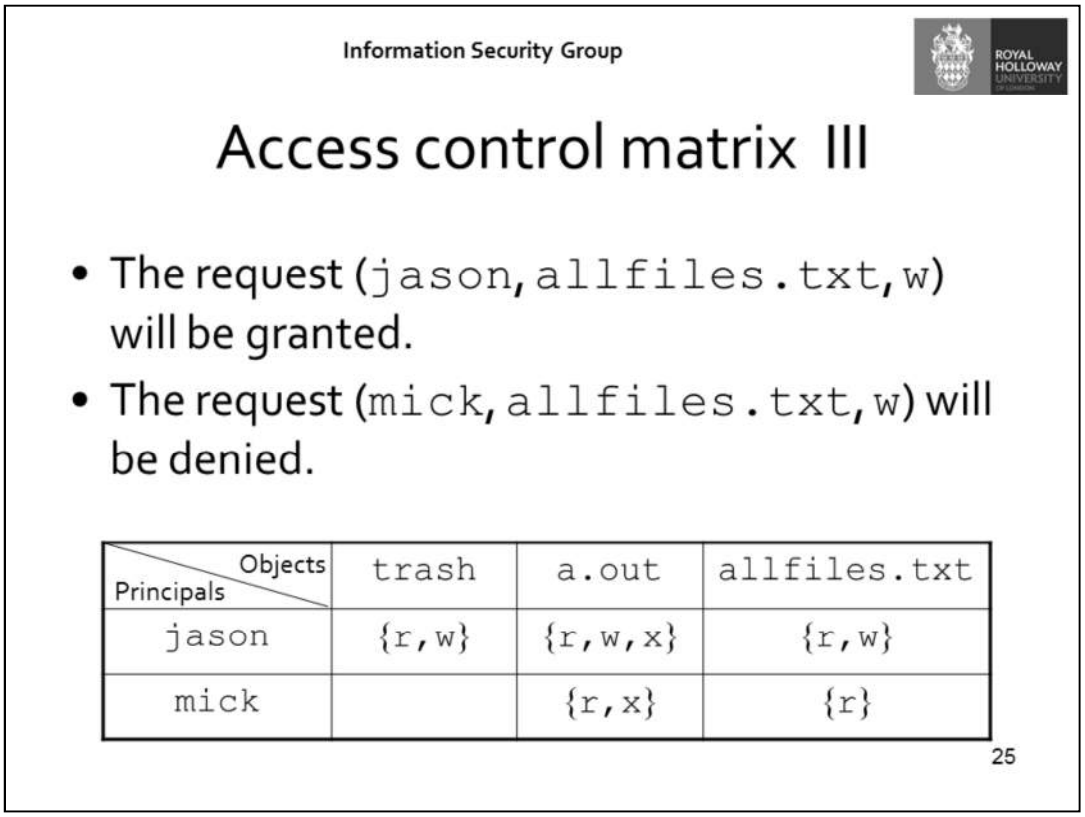
It is important to note that the notion of an access control matrix was never intended to be implemented. Its main role is to provide a way of thinking about the access control problem. That is, once we have a formal model of access control, we can start to describe real life systems using the model.



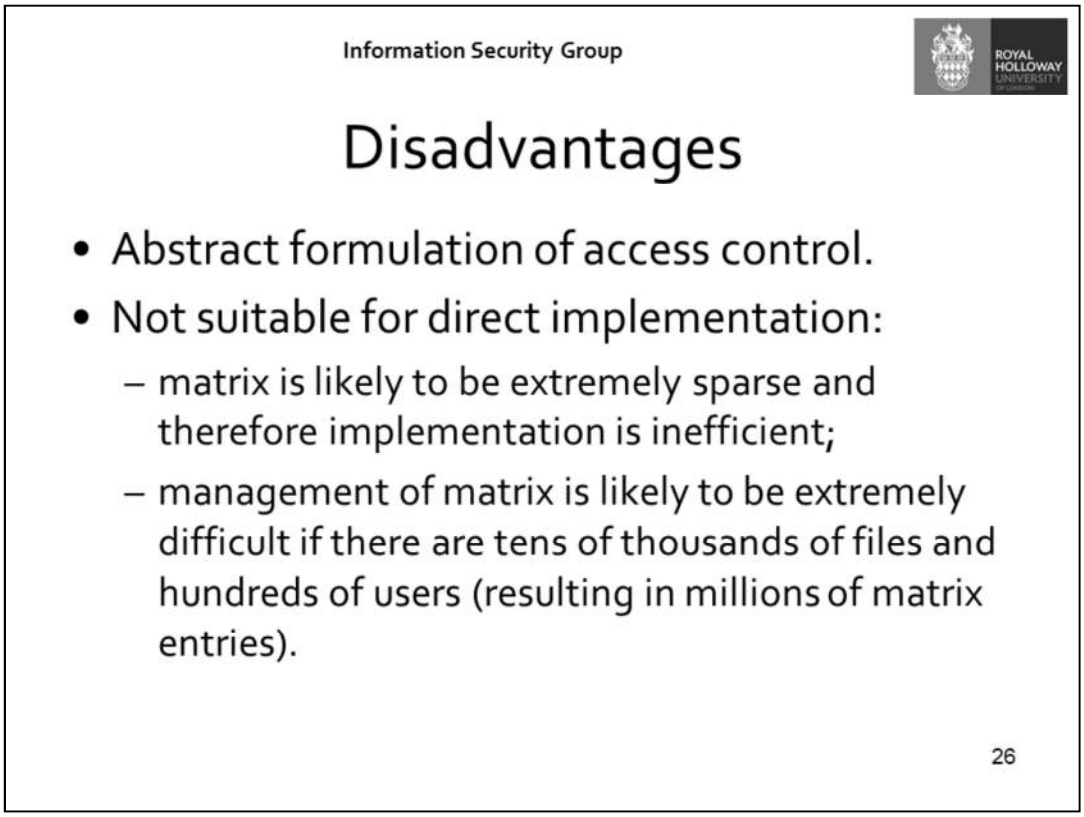
In this model of access control, an access request can be regarded as a triple (*p*, *o*, *a*). This indicates that a subject acting on behalf of principal *p* wants to access object *o* with access type *a*.

A request is granted (by the reference monitor) if and only if *a* belongs to the access matrix entry corresponding to principal *p* and object *o*.

That is, we now have a mathematical way of describing how access control works. Once we have such a mathematical (formal) model, we have the basis to mathematically prove results about how the system operates.



In the example shown, the request (`jason`, `allfiles.txt`, `w`) will be granted, whereas the request (`mick`, `allfiles.txt`, `w`) will be denied.

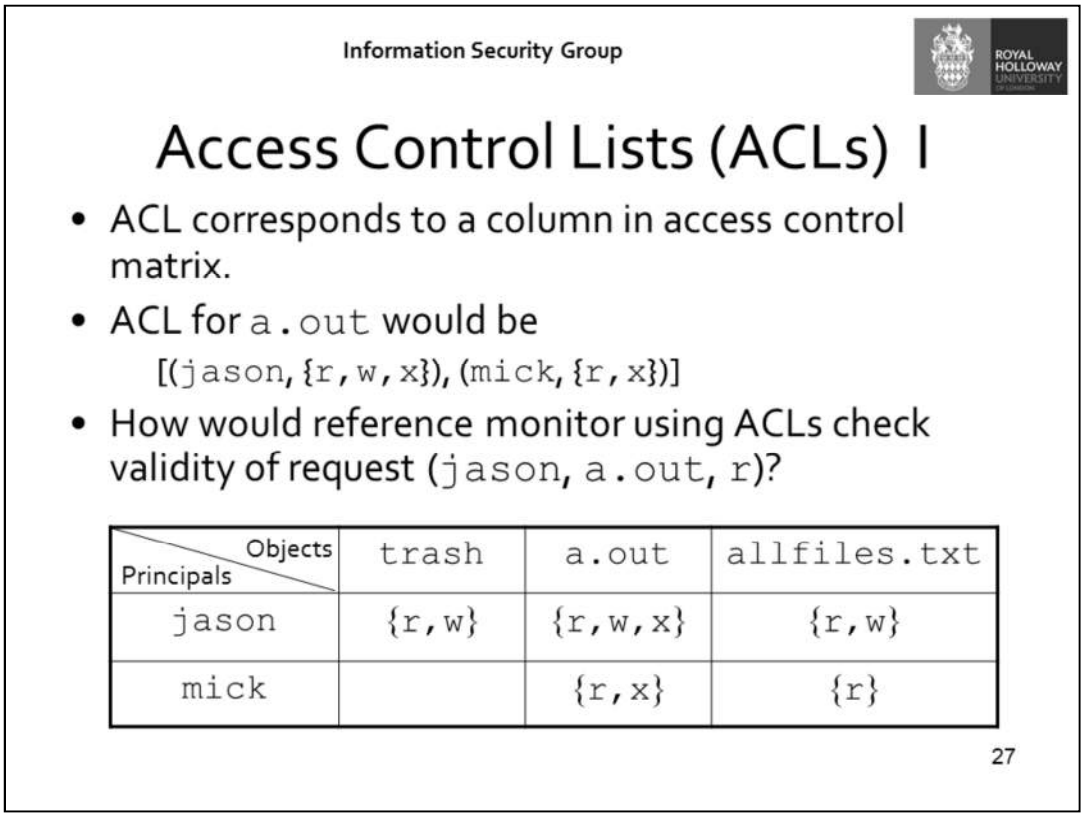


The access control matrix is an abstract formulation of access control. It is not suitable for direct implementation, for the following reasons.

• The matrix is likely to be extremely sparse, and therefore implementation is inefficient;

• Management of the matrix is likely to be extremely difficult if there are tens of thousands of files and hundreds of principals, i.e. users, resulting in millions of matrix entries.

Checking access rights needs to be very fast, since otherwise it could dramatically slow the overall performance of a system. This is because access checks will often need to be made very frequently during normal operation.



One widely used practical model for access control is the Access Control List (ACL). An ACL corresponds to a column in the access control matrix. Each object maintains a list of the principals permitted to have access (and, in each case, what the access rights are for that principal).

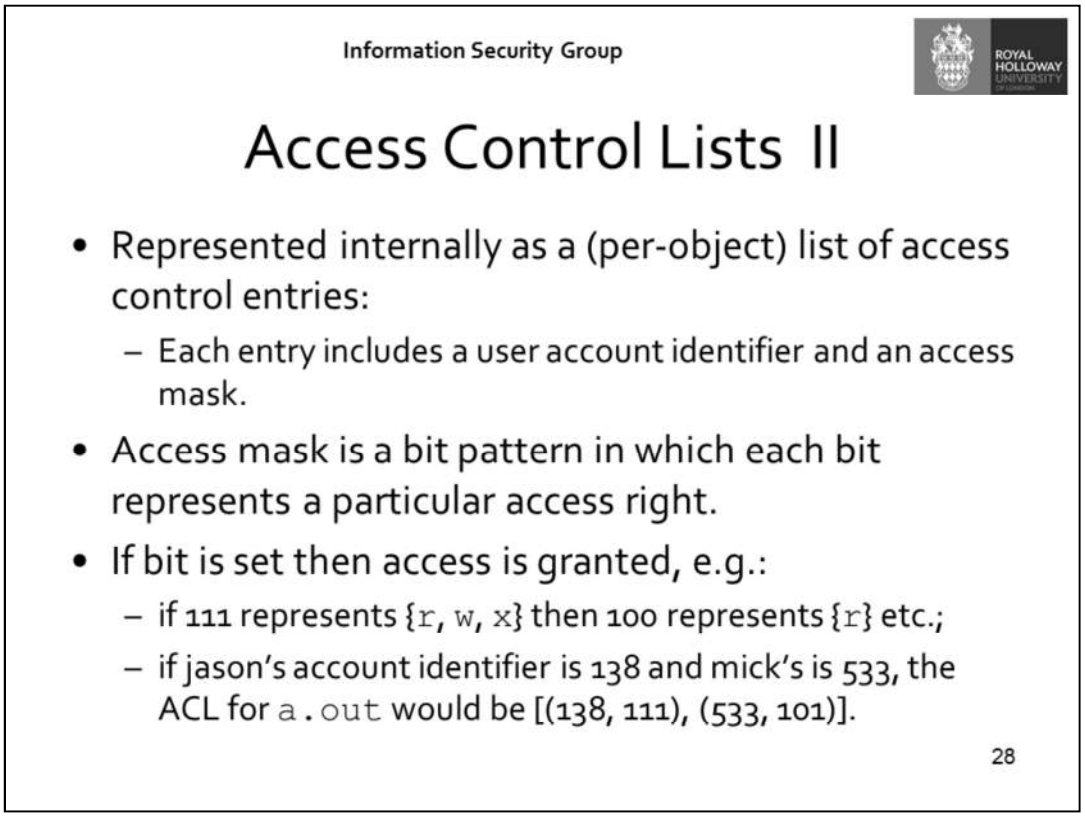
The ACL for `a.out` (as in the previous example) would be

[(`jason`, {`r`, `w`, `x`}), (`mick`, {`r`, `x`})]

How would a reference monitor that uses ACLs check the validity of the request (`jason`, `a.out`, `r`)?

It is important to note that a **subject** is requesting access to an object, whereas the decision is made on the basis of the **principal** associated with the subject. This means that all the subjects (processes) acting on behalf of a single principal (user) have the same access rights.

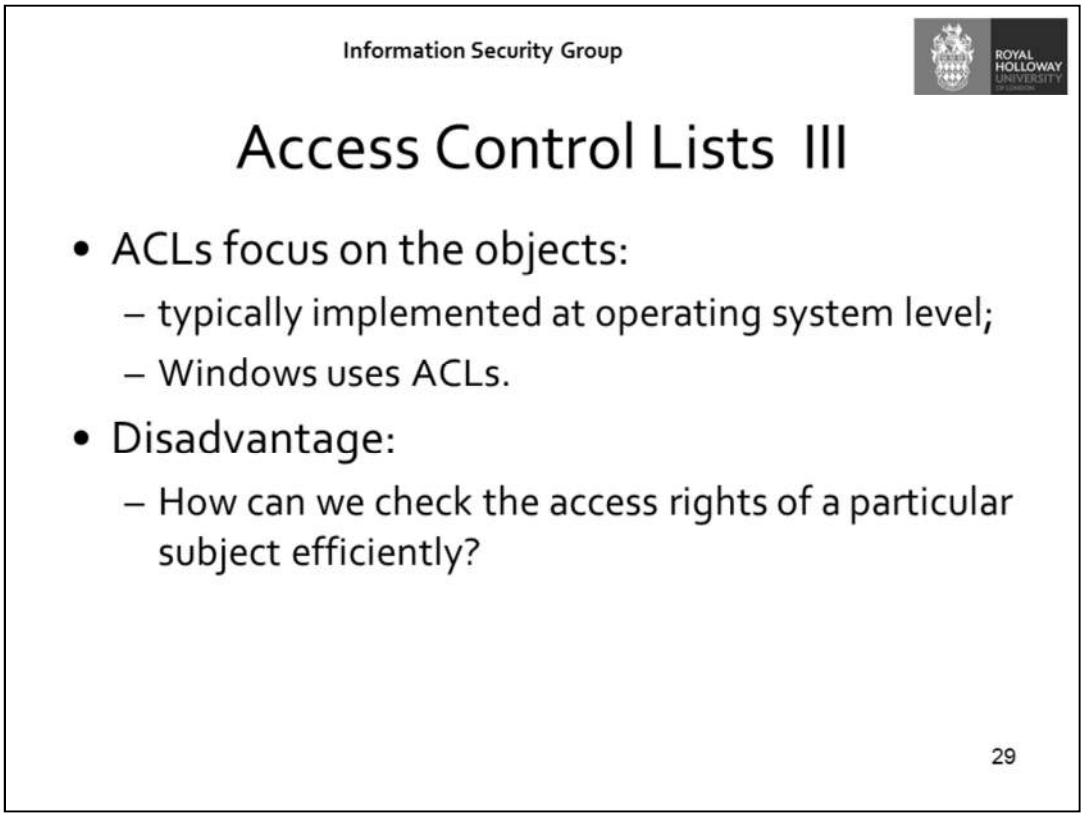
We can mathematically model ACLs using the access control matrix model, since the only difference between them is how access control information is stored. Of course, storing data in particular ways can have huge performance implications.



ACLs are typically represented internally as a (per-object) list of access control entries, where each entry includes a user account identifier and an access mask.

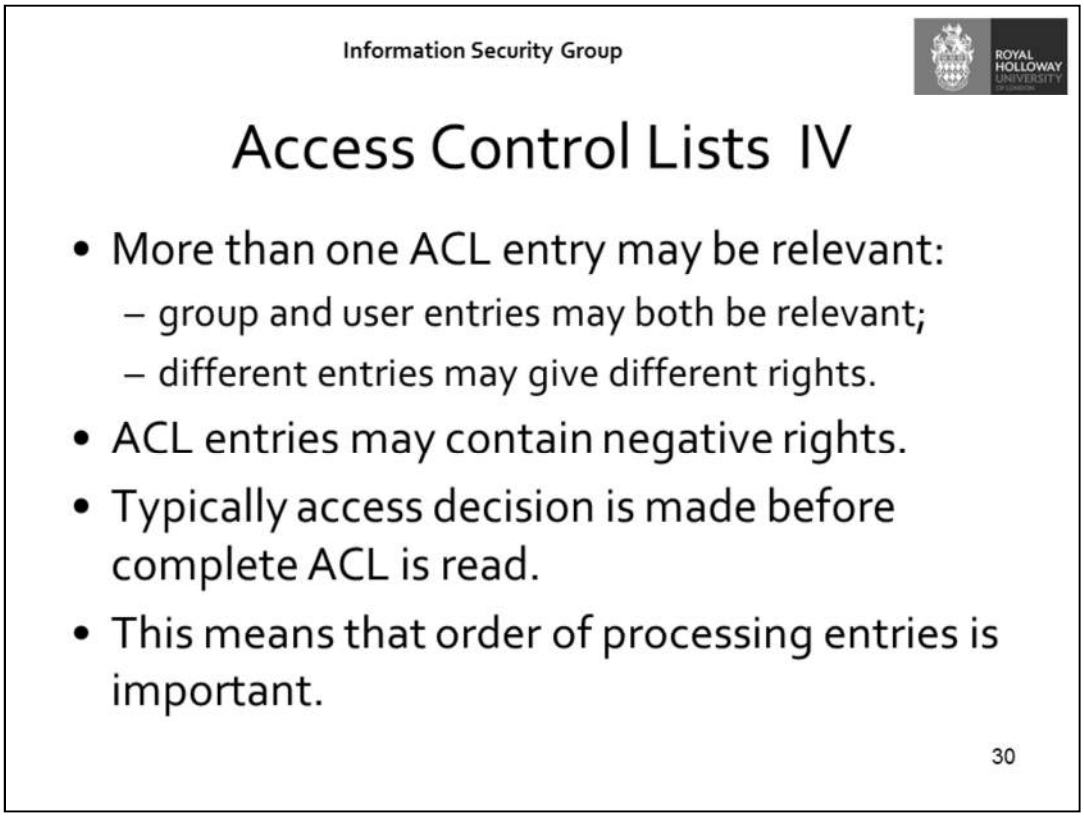
An access mask is a bit pattern in which each bit represents a particular access right:

- if the bit is set then access of the particular type is granted;
- if 111 represents {`r`, `w`, `x`} then 100 represents {`r`} etc.;
- if `jason`'s account identifier is 138 and `mick`'s is 533, then the ACL for `a.out` would be [(138, 111), (533, 101)].



Access Control Lists are associated with the objects. They are typically implemented at the operating system level (e.g. the ACL for an object will be stored with that object). Windows uses ACLs.

One major disadvantage of ACLs arises if we want to check the access rights of a particular subject efficiently, e.g. for auditing reasons. This will require looking at every object's ACL.

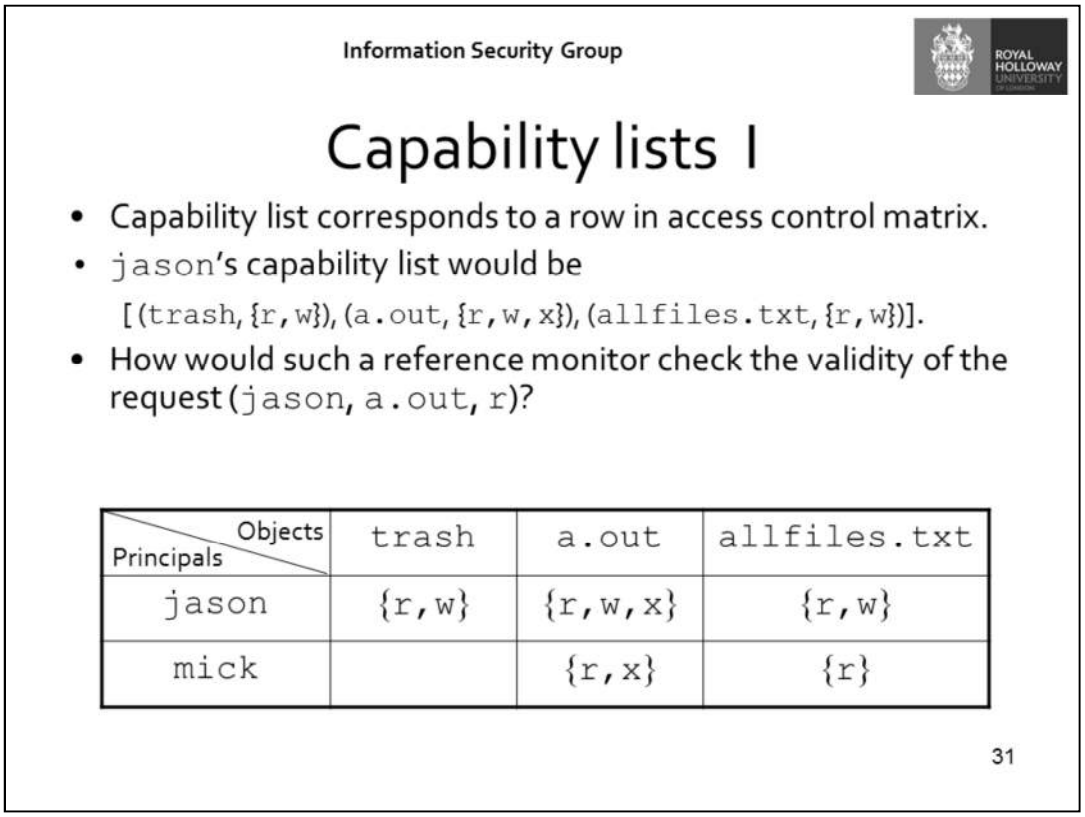


An ACL contains a list of entries. More than one entry in an ACL may be relevant to an access control decision. In particular:

- entries with user and group identifiers may both be relevant;
- distinct entries may give different rights (e.g. one may give read access and another may give write access).

Depending on the implementation, an ACL entry may contain negative access rights, i.e. specifying that a specific account identifier should be denied certain access rights to the object associated with the ACL.

Typically, an access decision is made as soon as enough entries from the ACL have been processed, i.e. before the complete ACL has been read. This means that the order in which entries are processed is important (if there are conflicts in the list).



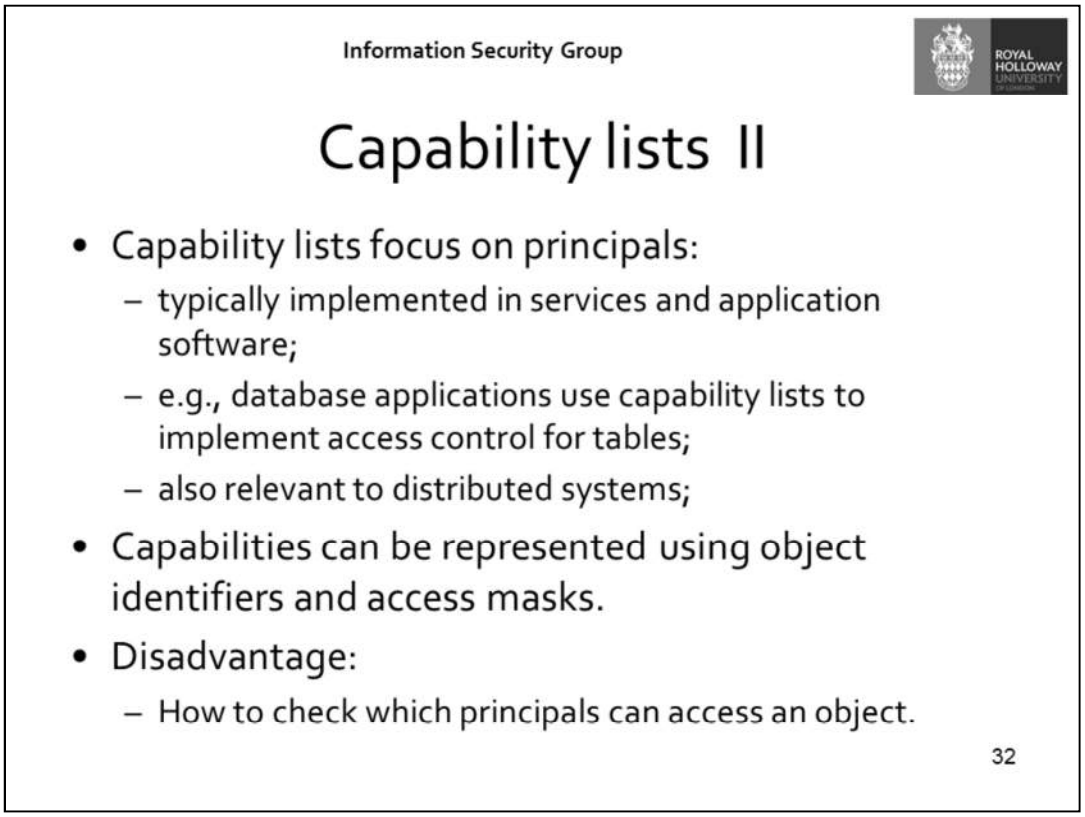
The notion of a capability is, in some sense, the 'dual' of an ACL. A capability list corresponds to a row in the access control matrix.

In the previous example, `jason`'s capability list would be

[(`trash`, {`r`, `w`}), (`a.out`, {`r`, `w`, `x`}), (`allfiles.txt`, {`r`, `w`})].

How would such a reference monitor check the validity of the request (`jason`, `a.out`, `r`)?

Note that, like ACLs, capabilities can be modelled using the access control matrix. Again the only difference is where the access control information is located, not in the fundamental way in which access decisions are made.

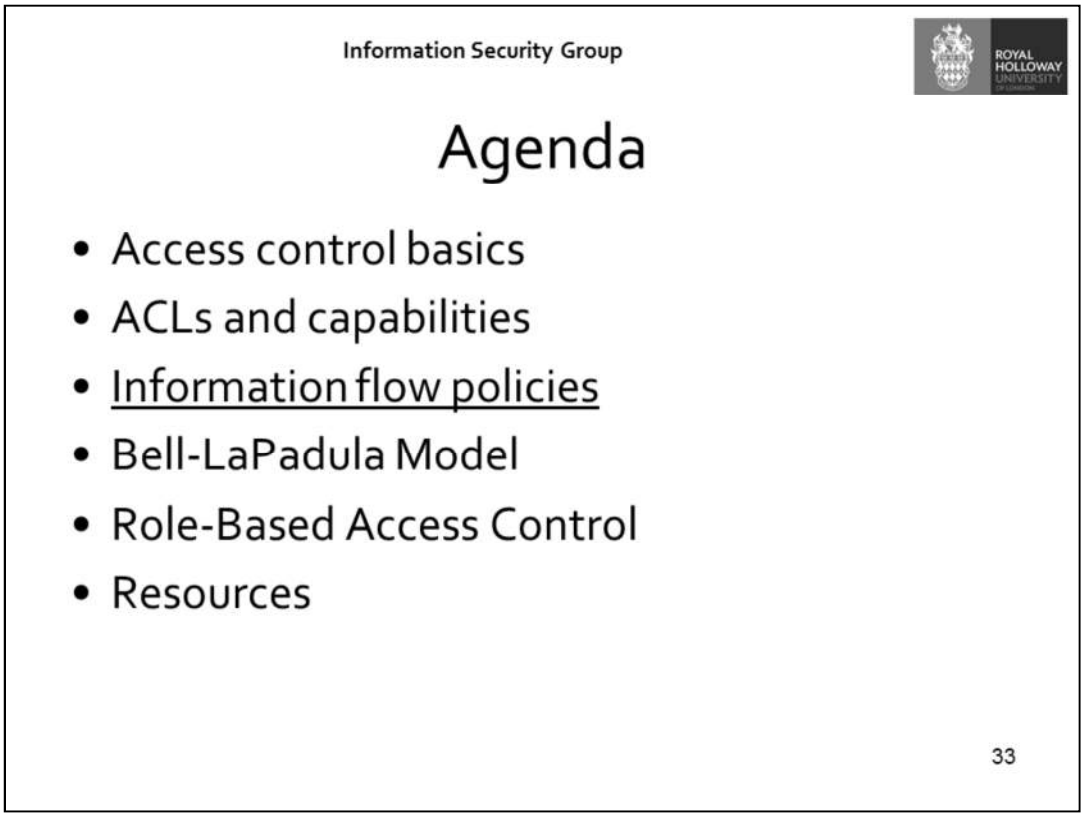


Capability lists are associated with the subjects. They are typically implemented in services and application software.

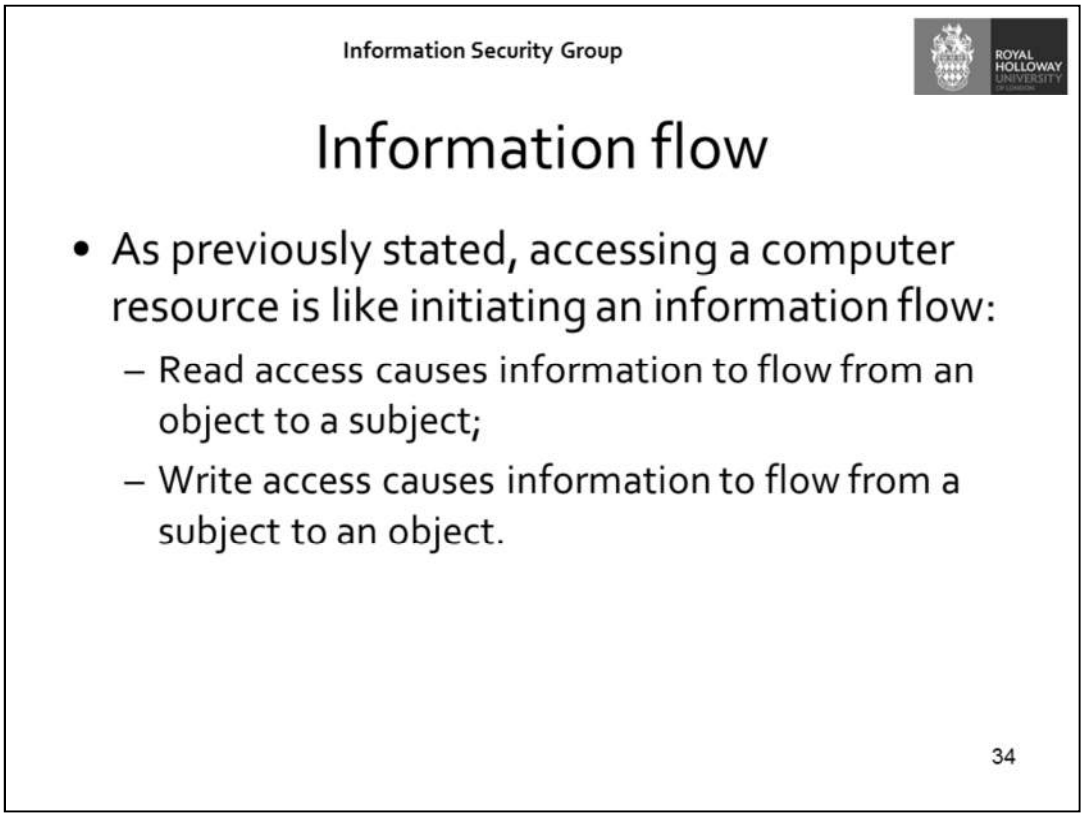
Database applications often use capability lists to implement fine-grained access control for queries to tables. Recently there has been renewed interest in capability-based access control for distributed systems.

Capabilities can be represented using object identifiers and access masks. However, in other circumstances a capability can be a 'stand alone' object digitally signed by the owner of the object, which can be presented to the reference monitor to gain access to an object.

One major disadvantage of capabilities arises if we need to check which principals can access a given object. This will require looking at every capability.

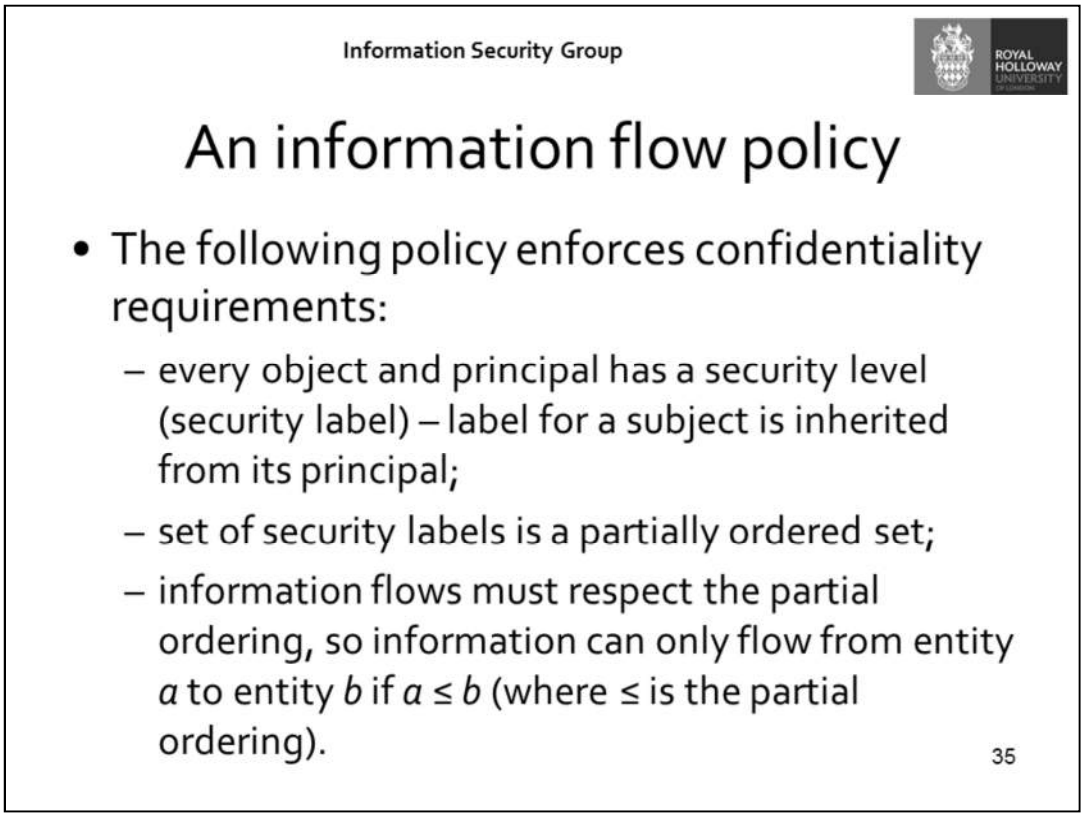


We next look at a somewhat different access control model, aimed primarily at preserving the confidentiality of sensitive data.



As previously stated, accessing a computer resource can be regarded as initiating an information flow. A **read** access causes information to flow from an object to a subject. A **write** access causes information to flow from a subject to an object.

That is, when a program has read access to file, information flows from the file to the program. Similarly, when a program has write access to a file, information flows from the program to the file.



The notion of information flows can be used to construct an access control model, on the basis that confidentiality constraints can be thought of as rules governing flows of information.

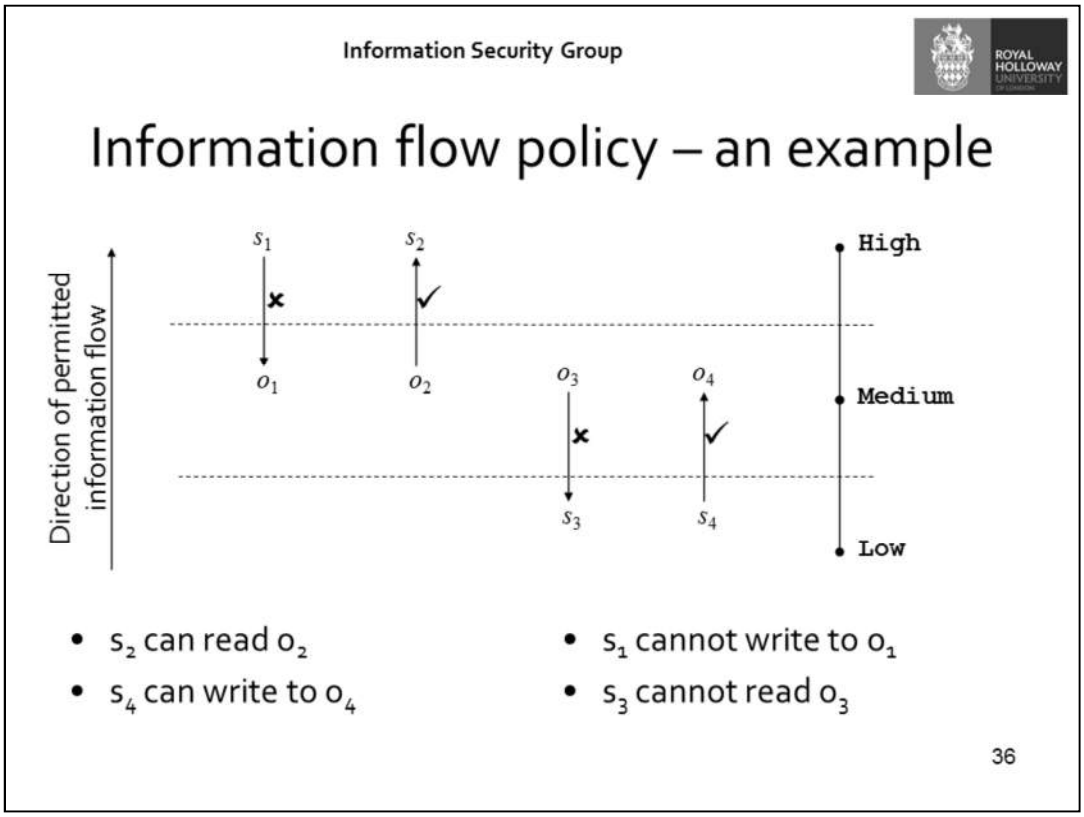
The following access control policy enforces confidentiality requirements for the case where information (and subjects) are subject to classification rules:

• every object and principal has a security level (security label) – the label on an object indicates the **sensitivity** of the object, and the label on a principal indicates the **clearance** of the principal (the label for a subject is inherited from its associated principal);

• the set of security labels is a (partially) ordered set;

• information flows must respect the partial ordering, in that information can only flow from entity *a* to entity *b* if $a \leq b$ (where ≤ is the partial ordering).

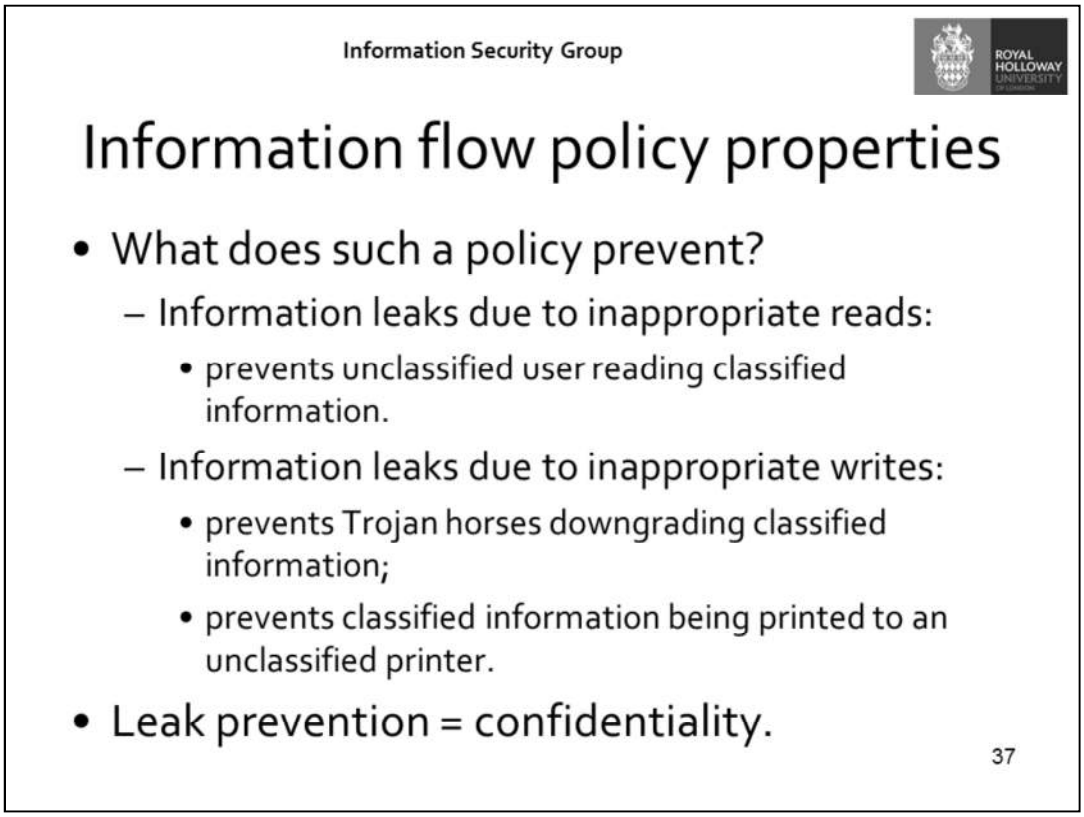
I.e. information is only allowed to flow from a lower classification to a higher classification (or to the same level of classification).



In this example, subjects s_1 and s_2 have high security clearance (inherited from the associated principal). Objects o_1, o_2, o_3 and o_4 all have medium security sensitivity. Subjects s_3 and s_4 have low security clearance. Then:

- s_2 can read o_2
- s_4 can write to o_4
- s_1 cannot write to o_1
- s_3 cannot read o_3

In other words, this policy enables ‘read down’ and ‘write up’.

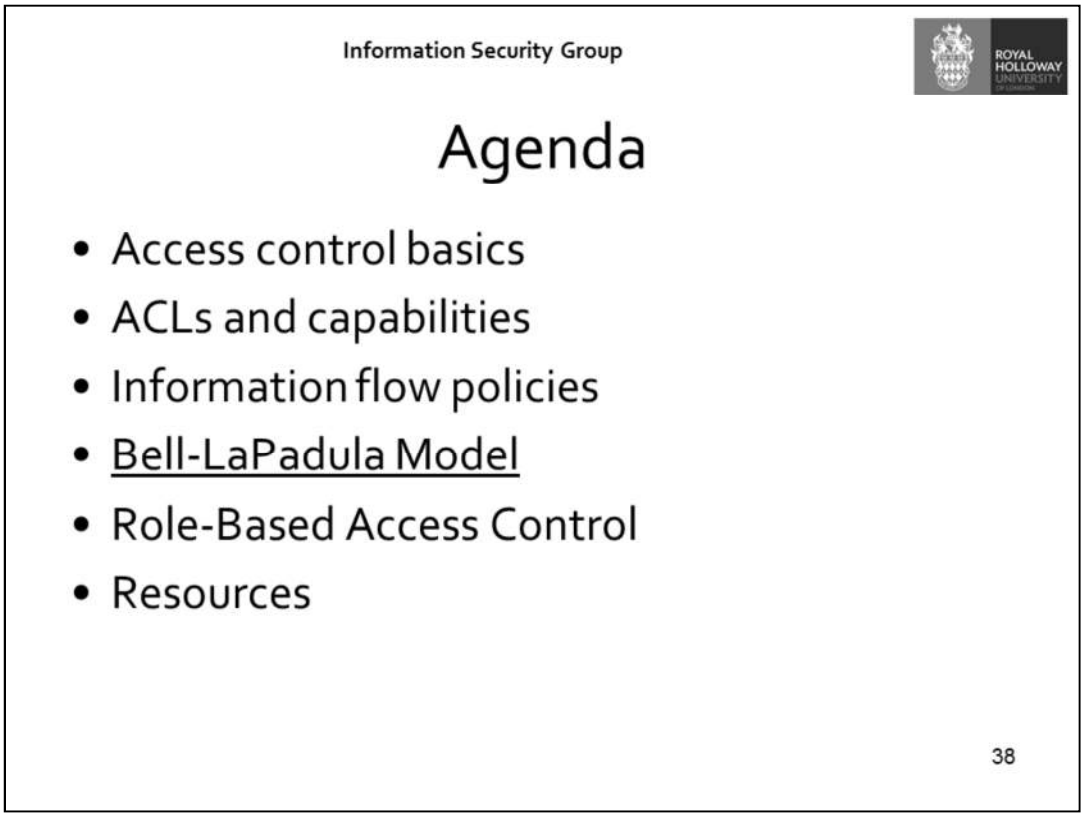


An information flow policy addresses the following types of security violation.

• Information leaks due to inappropriate read actions. For example, it prevents unclassified user reading classified information.

• Information leaks due to inappropriate write actions. For example, it prevents Trojan horses downgrading classified information, and prevents classified information being printed to an unclassified printer.

That is, it addresses confidentiality. However, it does not address integrity requirements.



The Bell-LaPadula access control model has been widely discussed in the literature.

The original reports by David Bell and Leonard LaPadula are available here:

http://www.albany.edu/acc/courses/ia/classics/belllapadula1.pdf

and here:

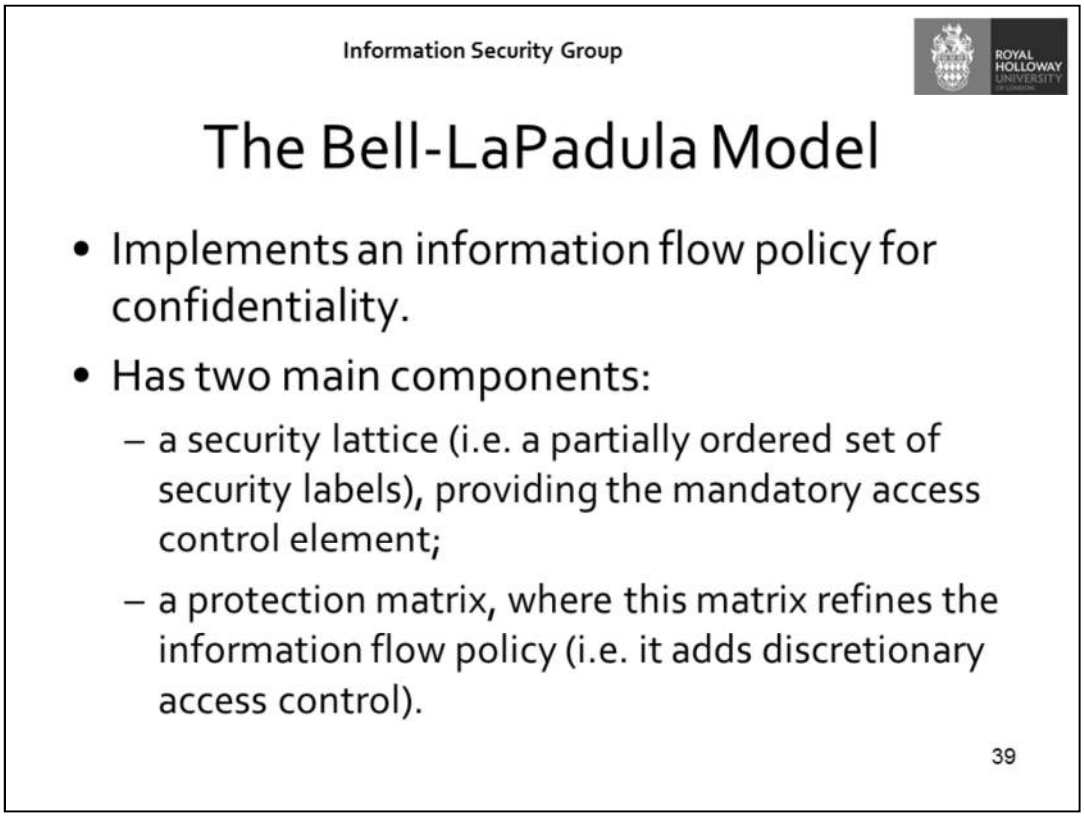
http://csrc.nist.gov/publications/history/bell76.pdf

The latter paper is part of the collection of classic computer security papers available here:

http://csrc.nist.gov/publications/history/

A useful retrospective on this ground-breaking work, written by David Bell, is available here:

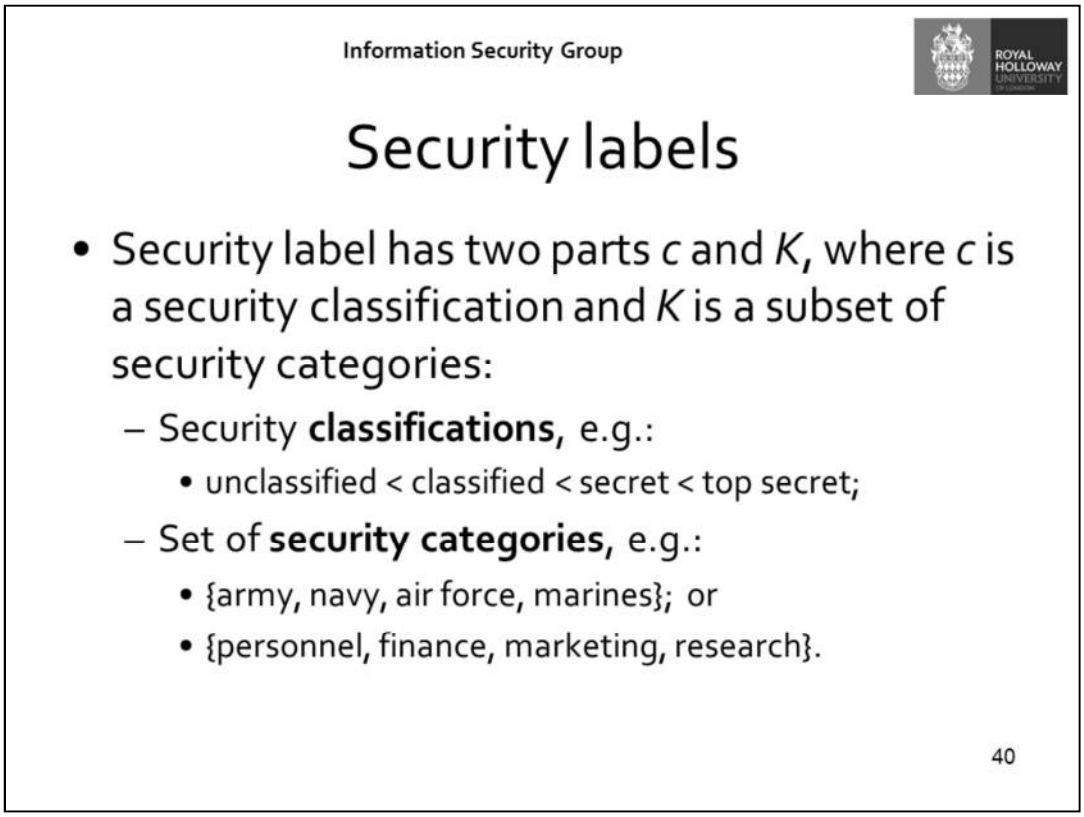
http://www.acsac.org/2005/papers/Bell.pdf



The Bell-LaPadula model, is one of the most famous access control models, designed for high security (e.g. government) systems. It implements an information flow policy for confidentiality (providing mandatory access control), and also integrates an access control matrix (to provide discretionary access control).

It requires subjects and objects to be assigned labels from a security lattice (i.e. a partially ordered set of security labels) – that is all subjects are objects are assigned classification categories.

It also uses a protection matrix (i.e. an access control matrix), where this matrix refines the information flow policy.



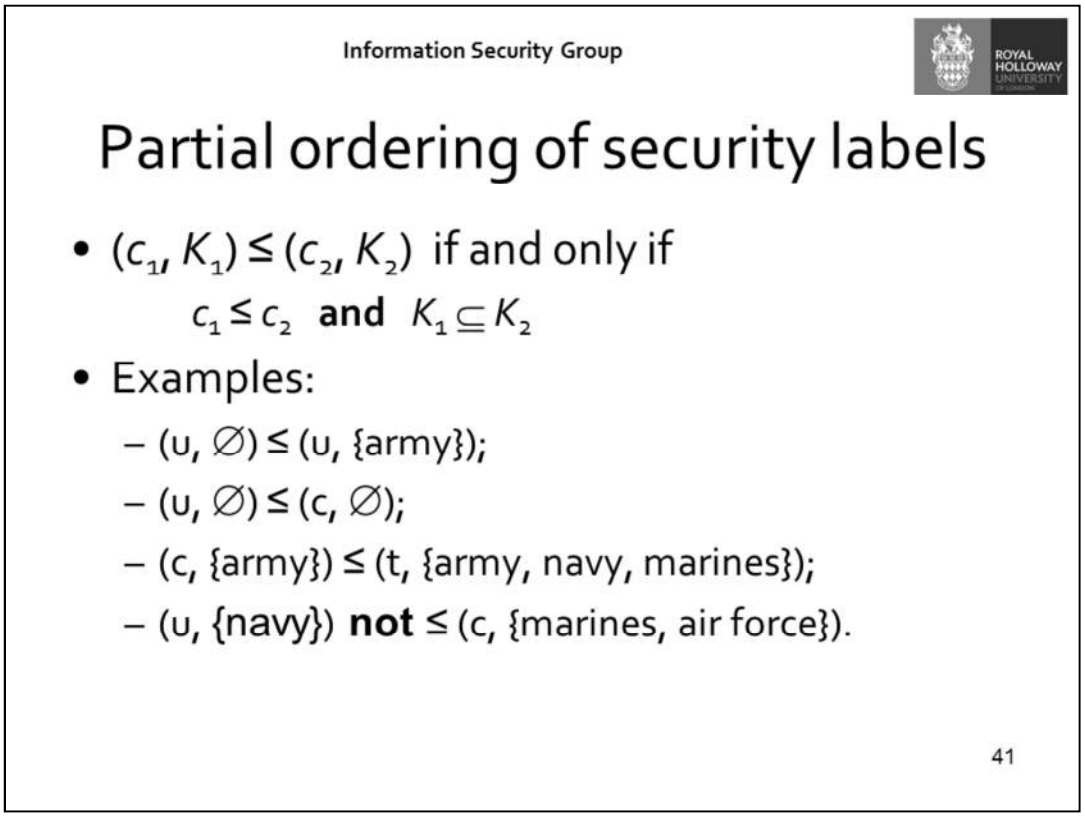
Bell-LaPadula (BLP) uses a special type of security label. A BLP security label has two parts: *c* and *K*, where *c* is a security classification, and *K* is a subset of security categories.

An example of the set of security classifications might be:

unclassified < classified < secret < top secret;

An example of security (need-to-know) categories, would be:

{army, navy, air force, marines} or {personnel, finance, marketing, research}.



A partial ordering ≤ is defined on the set of security labels, where:

$(c_1, K_1) \leq (c_2, K_2)$ if and only if $c_1 \leq c_2$ **and** $K_1 \subseteq K_2$

Using the previous examples of classifications and categories, examples include:

(u, ∅) ≤ (u, {army});

(u, ∅) ≤ (c, ∅);

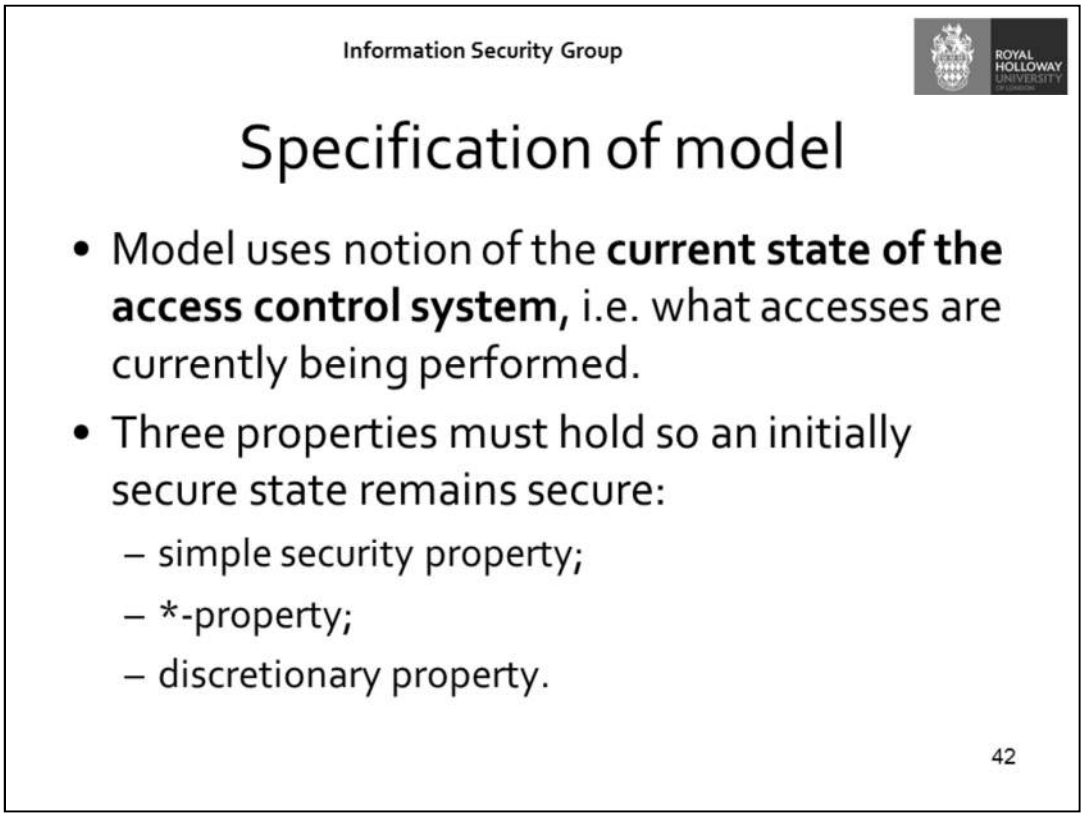
(c, {army}) ≤ (t, {army, navy, marines}).

where u stands for unclassified, c for classified, s for secret, and t for top secret.

However, note that:

(u, {army}) not≤ (c, {marines, air force})

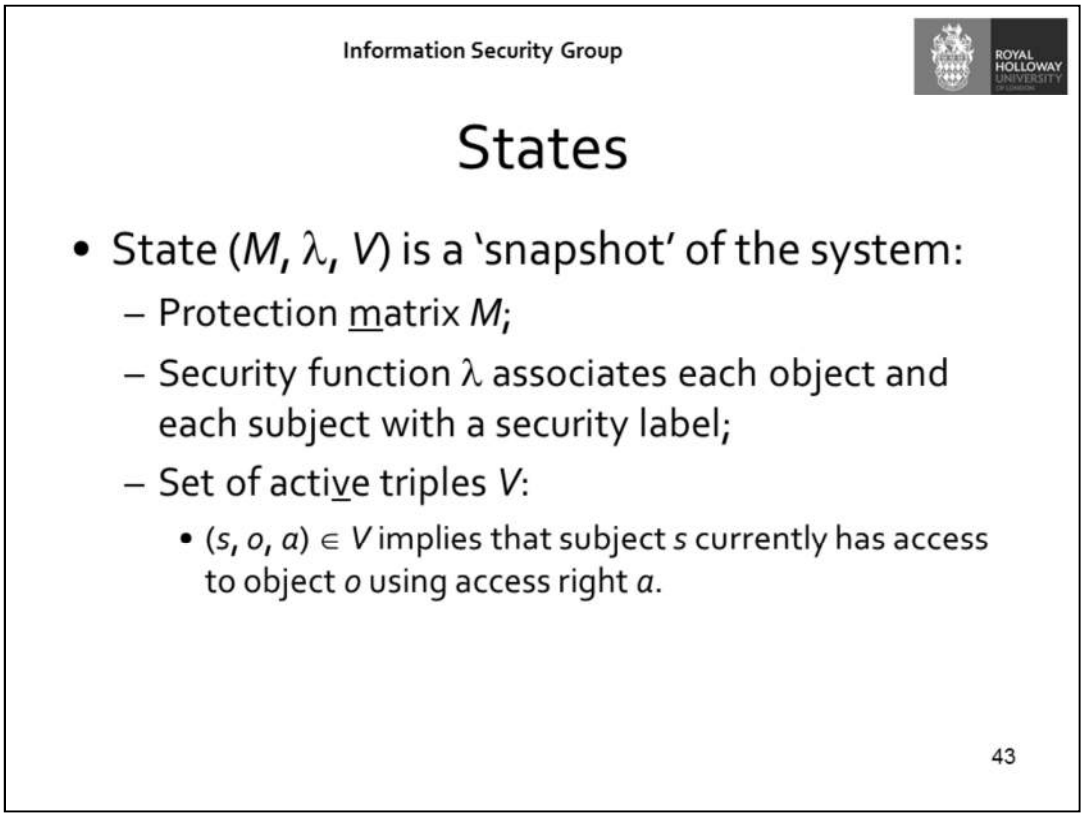
since although u≤c is true, {army} is not a subset of {marines, air force}.



The Bell-LaPadula (BLP) Model introduces the notion of the **current state of the access control system**, i.e. what accesses are currently being performed within the system.

The model also incorporates three properties which must hold so that an initially secure state remains secure:

- the simple security property;
- the *-property;
- the discretionary property.



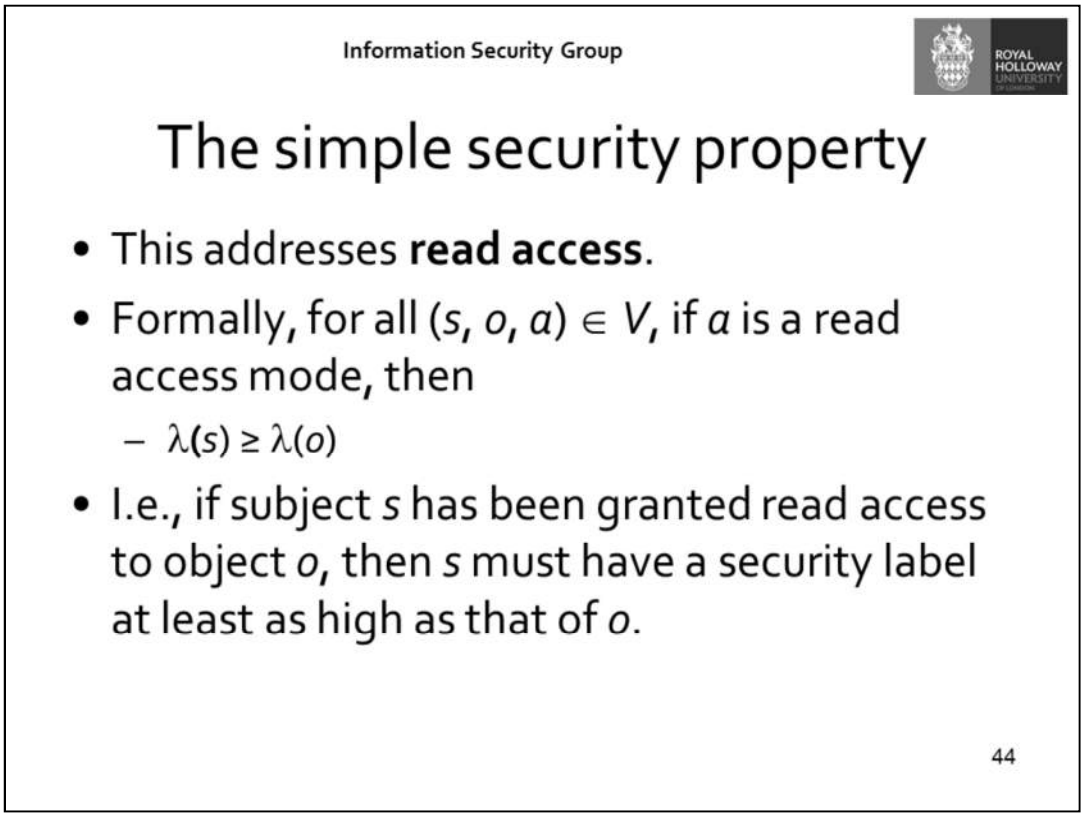
A state (M, λ, V) in the BLP model represents a "snapshot" of the system:

• M is the **protection matrix** (with rows corresponding to subjects, and columns corresponding to objects);

• The security function λ associates each object and each subject with a security label;

• V is the set of **active triples**, i.e. (s, o, a) $\in$ V implies that subject s currently has access to object o using access right a.

If a user (s) has requested read access (r) to a file f and it has been granted, that is, the file has been loaded into primary memory, then (s, f, r) $\in$ V.



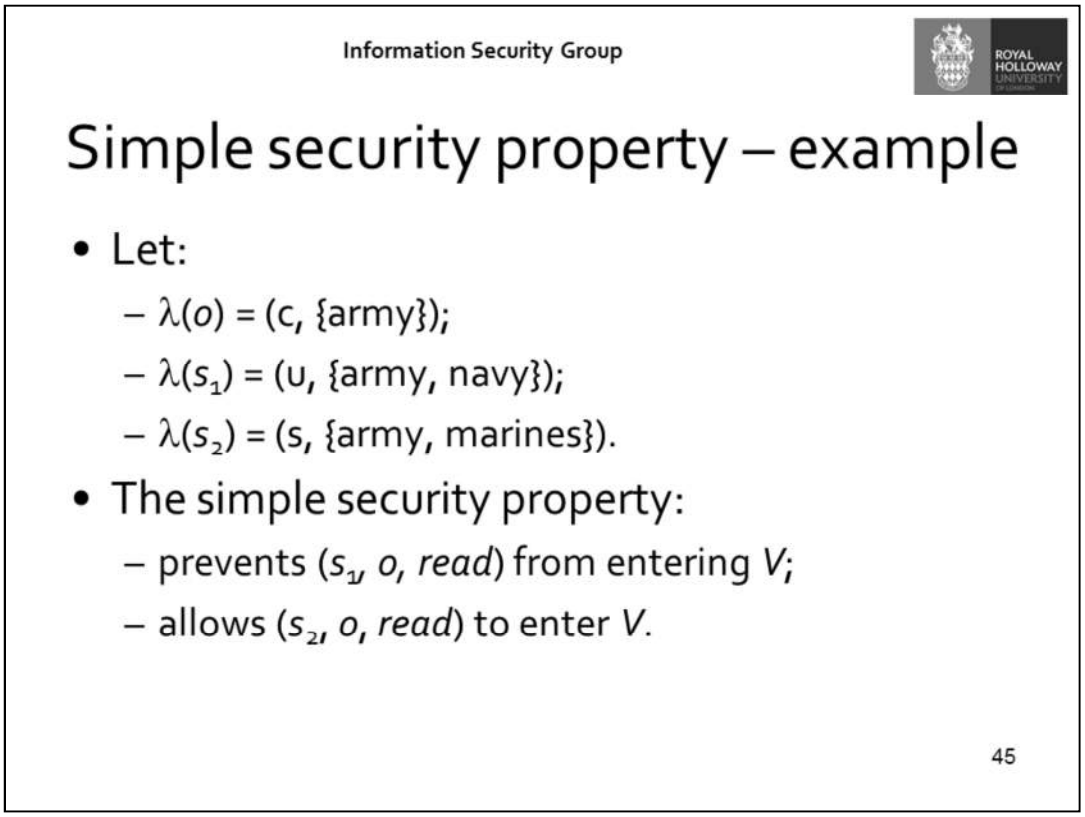
As stated previously, the BLP model requires three security properties to hold, the **simple security property**, the ***-property** and the **discretionary property**.

The simple security property, which covers read accesses, requires that, for all (*s*, *o*, *a*) ∈ *V*, if *a* is a read access mode, then

$$\lambda(s) \geq \lambda(o)$$

In other words, if subject *s* has been granted read-type access to object *o*, then *s* must have a security label that is at least as high as that of *o*.

This is implied by the information flow policy.



As an example, suppose that:

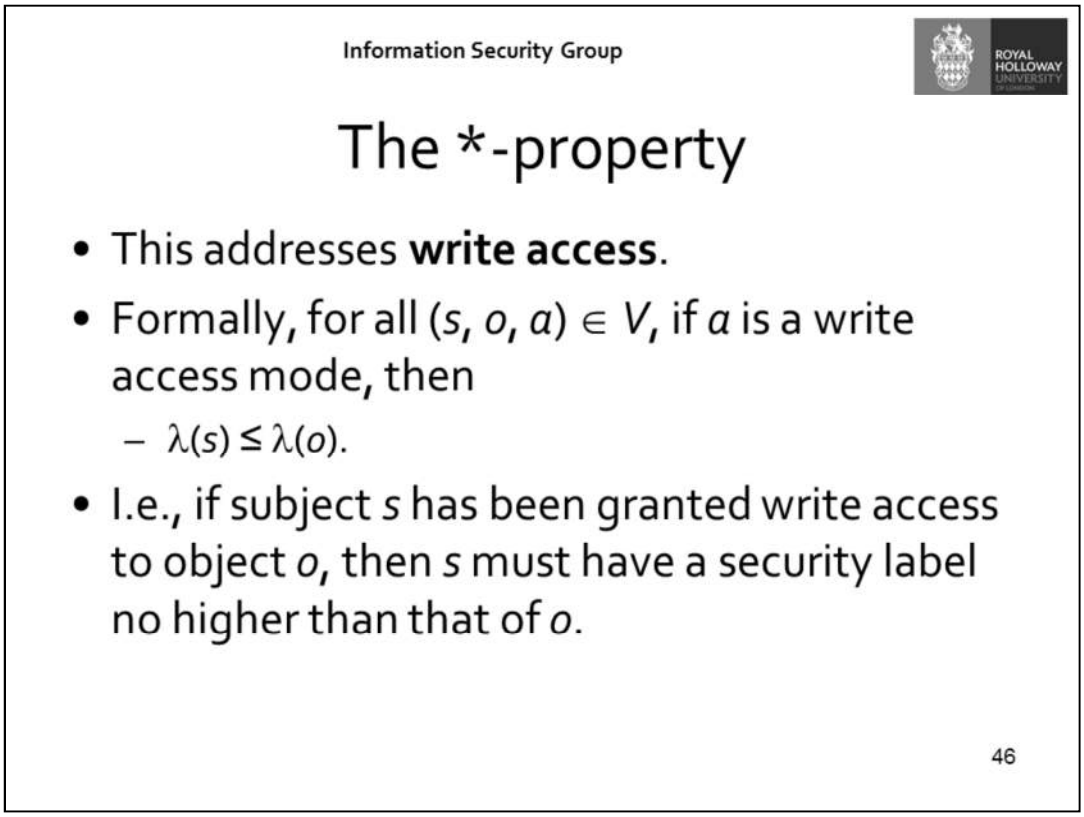
$\lambda(o)$ = (c, {army});

$\lambda(s_1)$ = (u, {army, navy});

$\lambda(s_2)$ = (s, {army, marines}).

The simple security property:

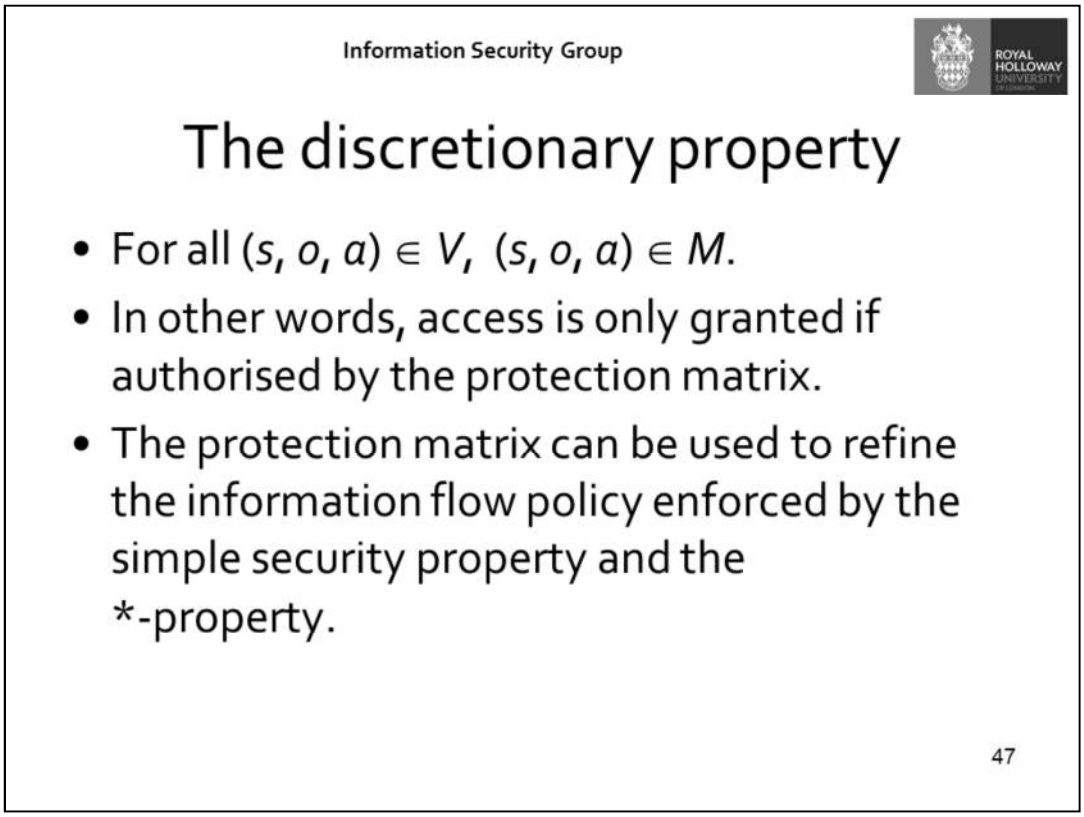
- prevents (s_1, *o, read*) from entering *V*;
- allows (s_2, *o*, *read*) to enter *V*.



The *-property, which covers **write accesses**, requires that, for all (s, o, a) $\in V$, if a is a write access mode, then

$$\lambda(s) \leq \lambda(o).$$

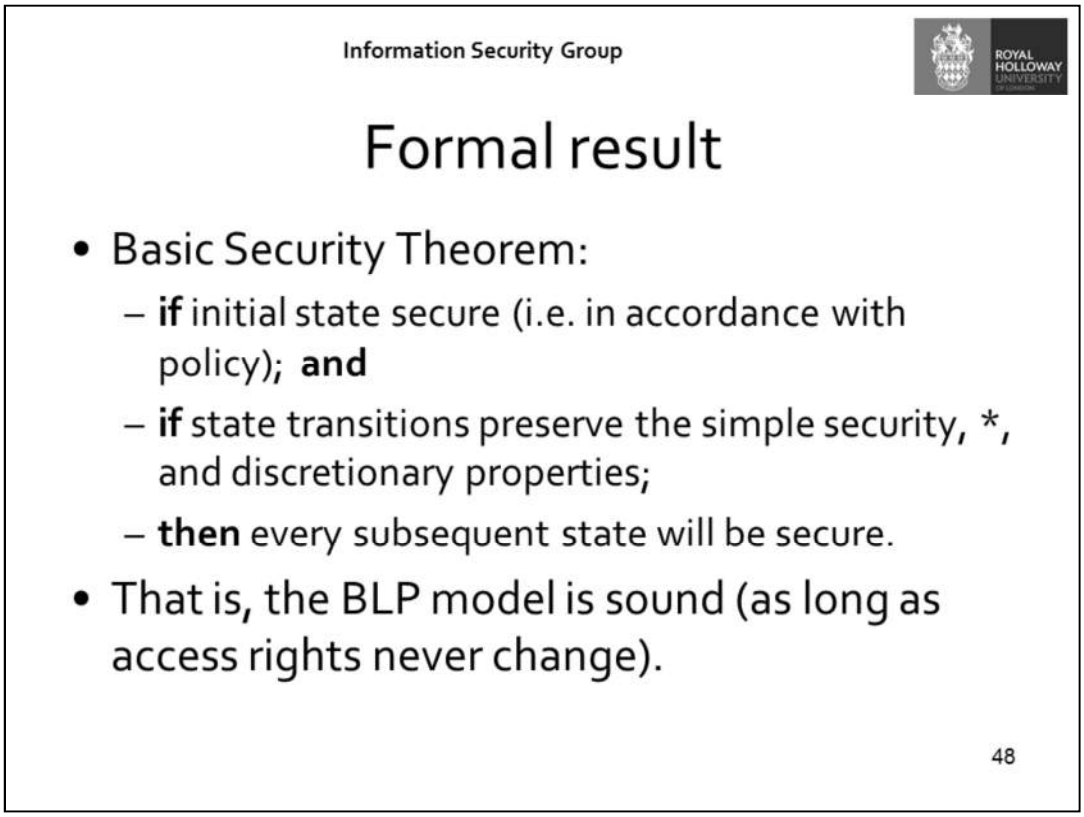
In other words, if subject s has been granted write-type access to object o, then s must have a security label that is no higher than that of o.



The discretionary property, which adds discretionary access control based on the access control matrix model, requires that, for all $(s, o, a) \in V$, $(s, o, a) \in M$.

In other words, access is only granted if authorised by the protection matrix.

The protection matrix can be used to refine the information flow policy that is enforced by the simple security property and the *-property.



Using the notions we have just introduced, it is possible to mathematically prove the following theorem.

Basic Security Theorem:

- **if** the initial state of the system is secure (i.e. in accordance with the security policy); **and**
- **if** state transitions preserve the simple security, *, and discretionary properties;
- **then** every subsequent state of the system will be secure.

That is, the BLP model is a sound model for access control. Note that the model does not capture the notion of changing access rights, i.e. it does not cover **administration** of access control.

Information Security Group

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Example I

- One subject s; three objects o_1, o_2 and o_3
  - (simplified) labels: $\lambda(s) = 2$, $\lambda(o_1) = 1$, $\lambda(o_2) = 2$, $\lambda(o_3) = 3$
- $V = \varnothing$.
- Three access rights: read (r), append (a) and write (w):
  - Append is write only access mode;
  - Write is a read and write access mode, i.e. for Write access to be granted we must have both $\lambda(s) \leq \lambda(o)$ and $\lambda(s) \geq \lambda(o)$, i.e. we must have $\lambda(s) = \lambda(o)$.
- M contains every access right in each entry:
  - Every request is authorised.

49

To illustrate the operation of the BLP model we give a simplified example.

Suppose there is one subject s, and three objects o_1, o_2 and o_3

We further suppose that the subject and objects have the following (simplified) labels:

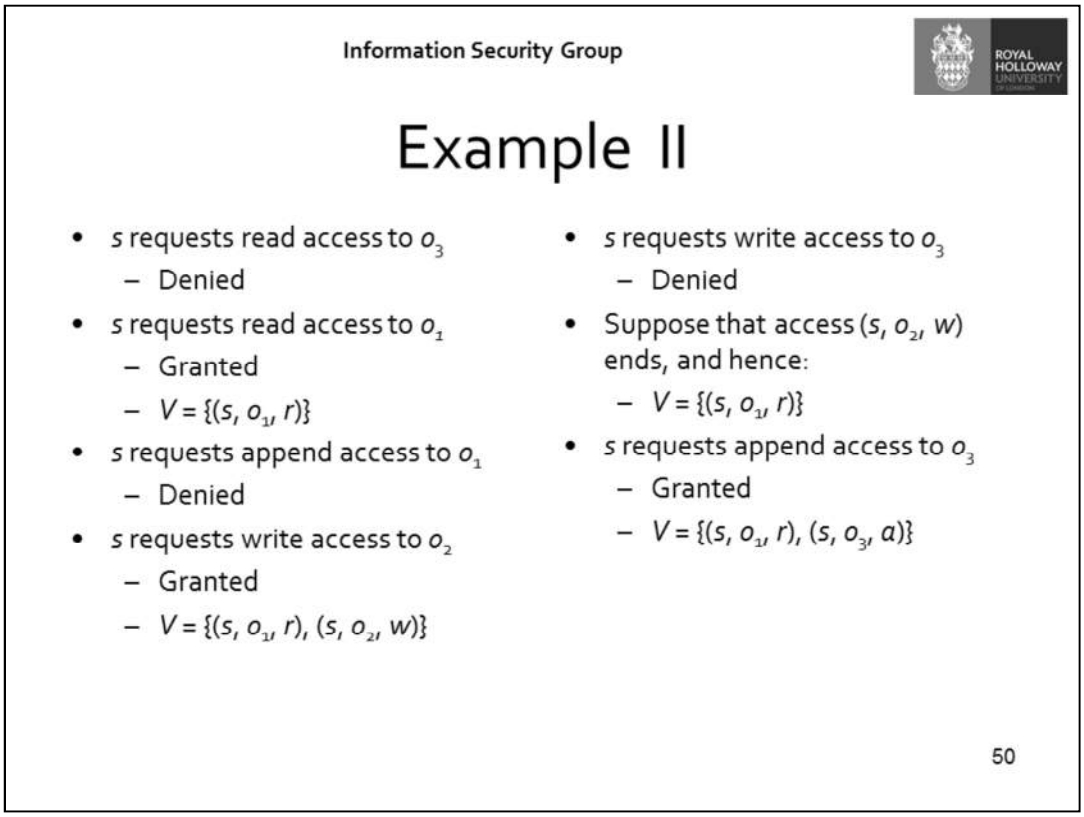
$$\lambda(s) = 2, \quad \lambda(o_1) = 1, \quad \lambda(o_2) = 2, \quad \lambda(o_3) = 3$$

We also suppose that the initial set of active triples, V, is empty, i.e. there are no current accesses – that is $V = \varnothing$.

We also suppose that there are three access rights: read (r), append (a) and write (w), where:

- Append is write only access mode;
- Write is a read and write access mode - this means that $\lambda(s) \leq \lambda(o)$ and $\lambda(s) \geq \lambda(o)$ must hold for a Write access to be permitted, i.e. we must have $\lambda(s) = \lambda(o)$

Finally we suppose that M contains every access right in each entry, i.e. every request is authorised as far as discretionary control is concerned.



Consider the following sequence of access requests.

s requests read access to o_3

Denied

s requests read access to o_1

Granted, and hence $V = \{(s, o_1, r)\}$.

s requests append access to o_1

Denied

s requests write access to o_2

Granted, and hence $V = \{(s, o_1, r), (s, o_2, w)\}$.

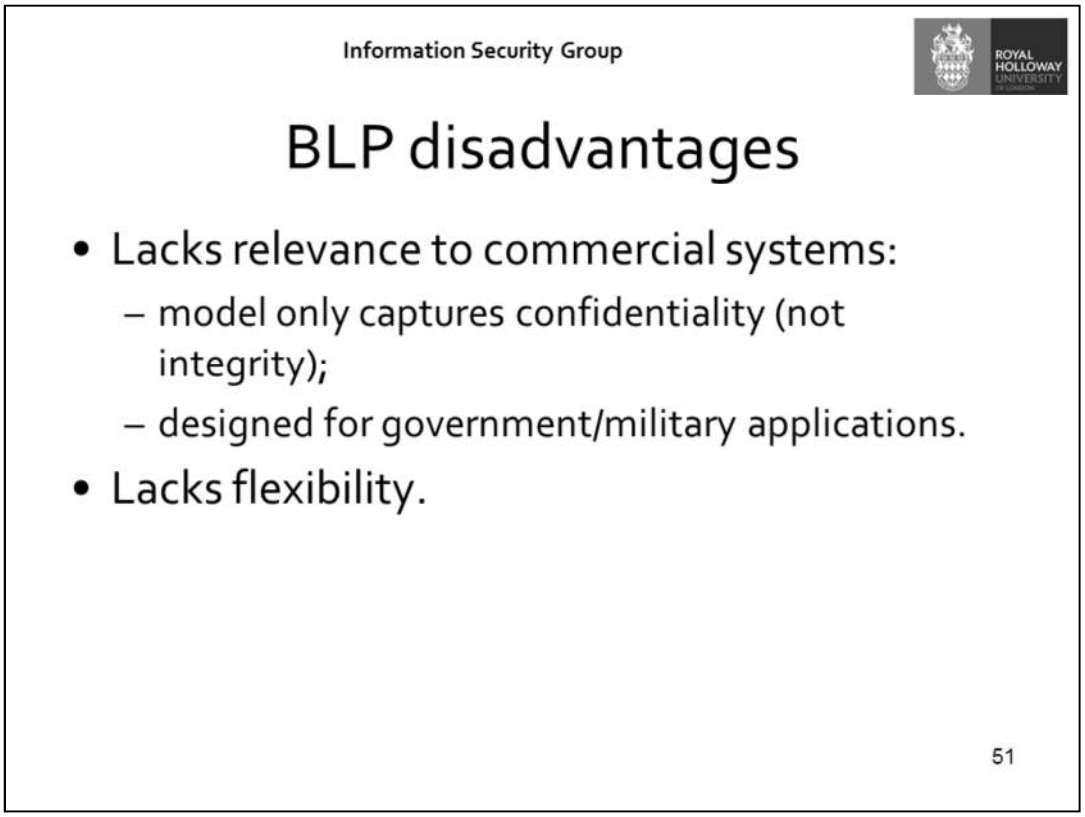
s requests write access to o_3

Denied.

Suppose that at this point the access (s, o_2, w) ends, and hence we have $V = \{(s, o_1, r)\}$.

s requests append access to o_3

Granted and hence $V = \{(s, o_1, r), (s, o_3, a)\}$.

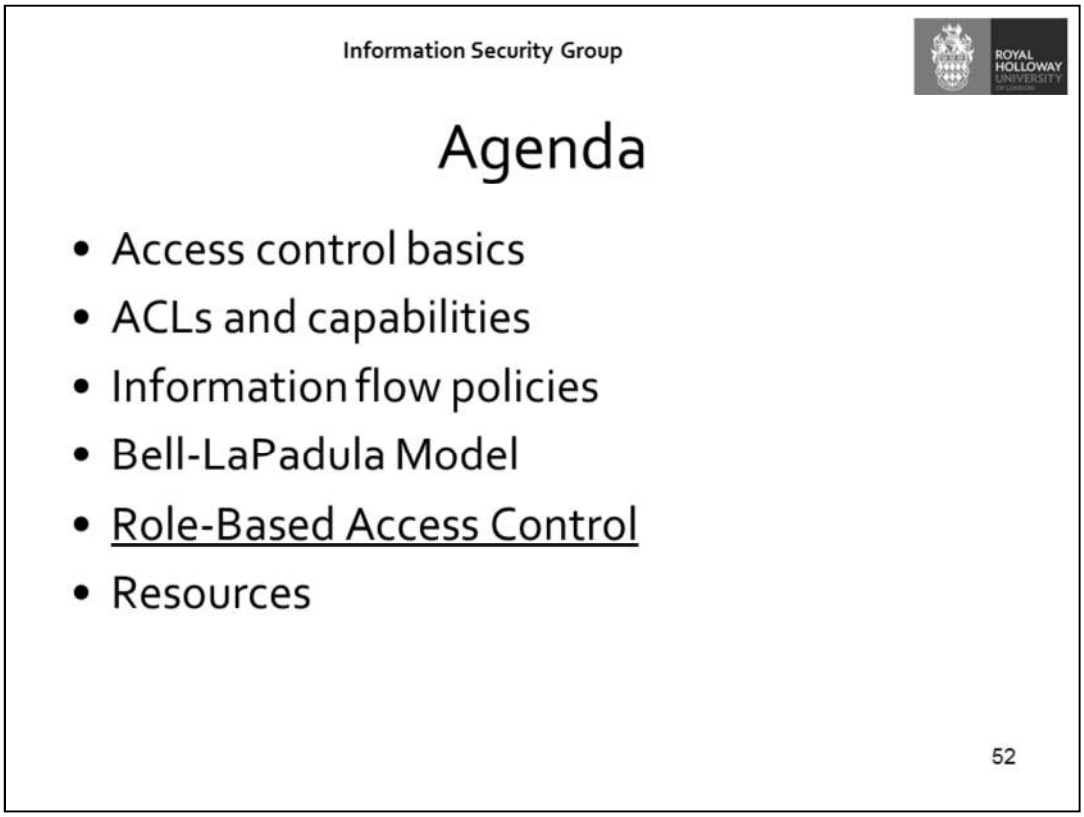


The Bell-LaPadula model has serious disadvantages. It lacks relevance to commercial systems.

Most seriously, the model only covers data confidentiality.

It was designed for government/military applications.

It lacks flexibility.



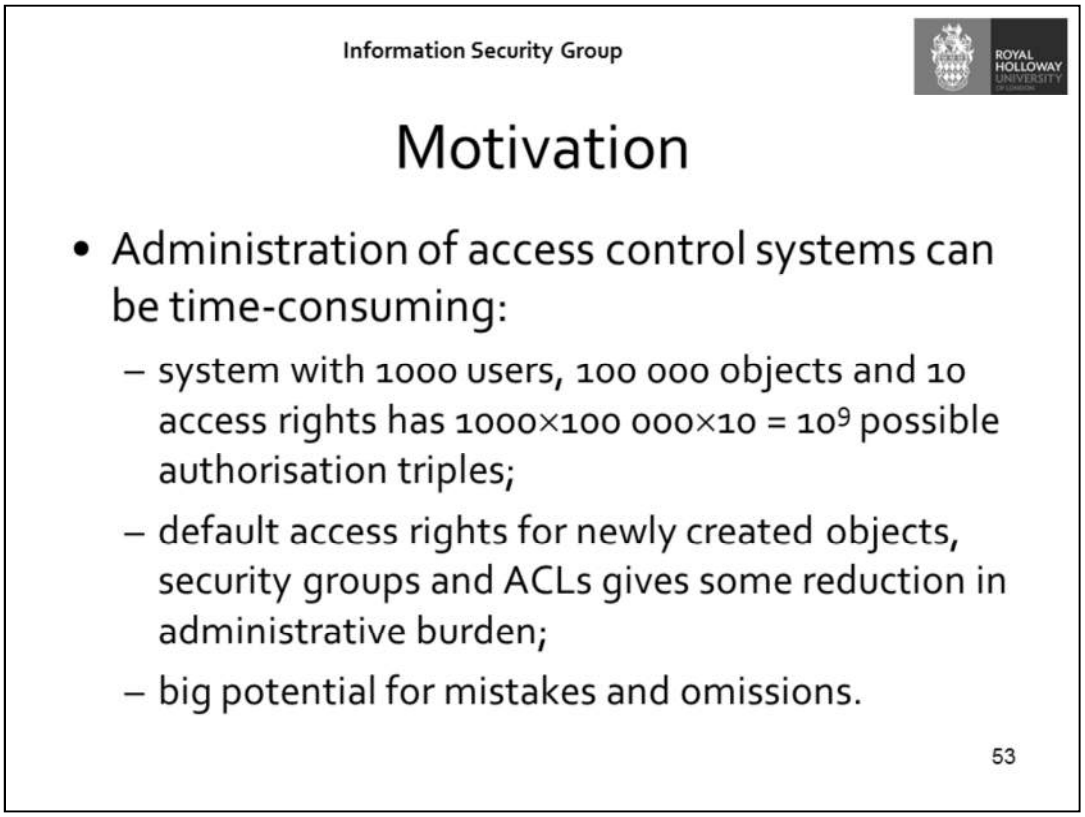
The ANSI standard for RBAC **ANSI INCITS 359-2004** is available here:

www.cs.purdue.edu/homes/ninghui/readings/AccessControl/ANSI+INCITS+359-2004.pdf

NIST provides useful information about RBAC, see e.g.:

http://csrc.nist.gov/groups/SNS/rbac/

This page also provides links to a range of useful background material on RBAC.



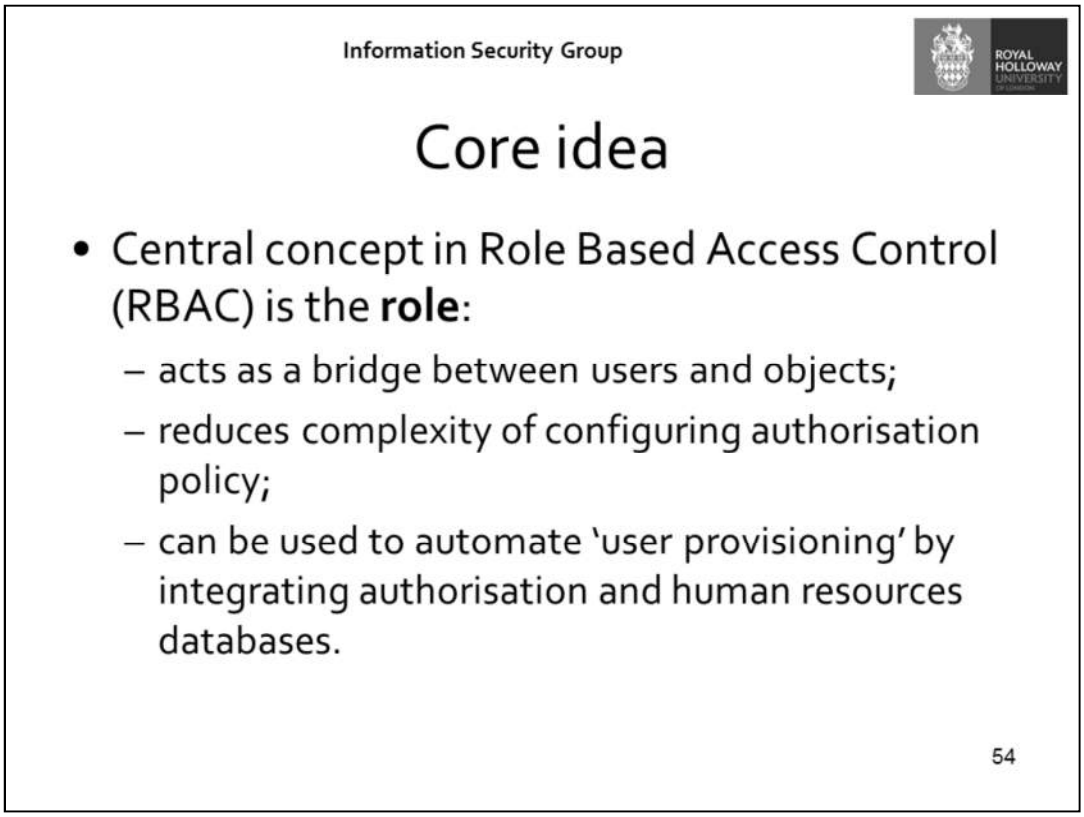
The administration of 'conventional' access control systems can be time-consuming and error-prone:

• a system with 1000 users, 100 000 objects & 10 access rights contains $1000 \times 100\,000 \times 10 = 10^9$ possible authorisation triples;

• the use of default access rights for newly created objects, security groups and ACLs leads to some reduction in the administrative burden;

• the potential for mistakes and omissions is large.

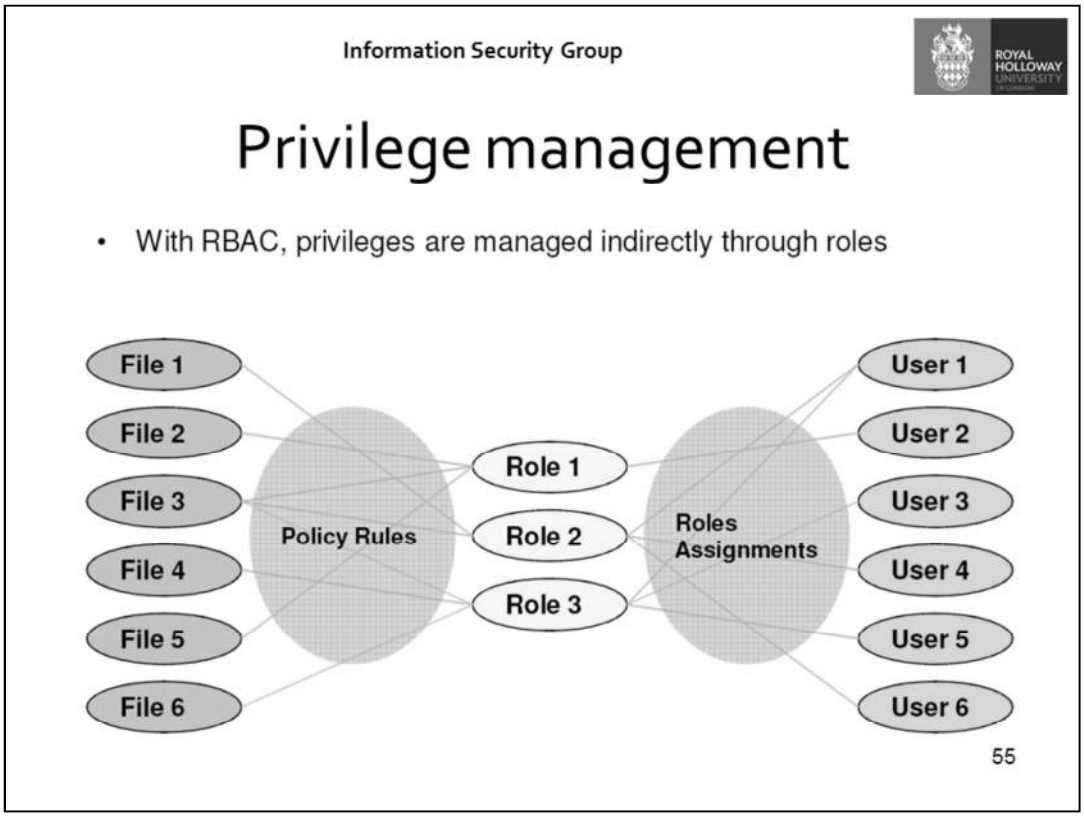
This problem has led to the development of the notion of Role-Based Access Control (RBAC).



The central concept in Role Based Access Control (RBAC) is that of a **role**:

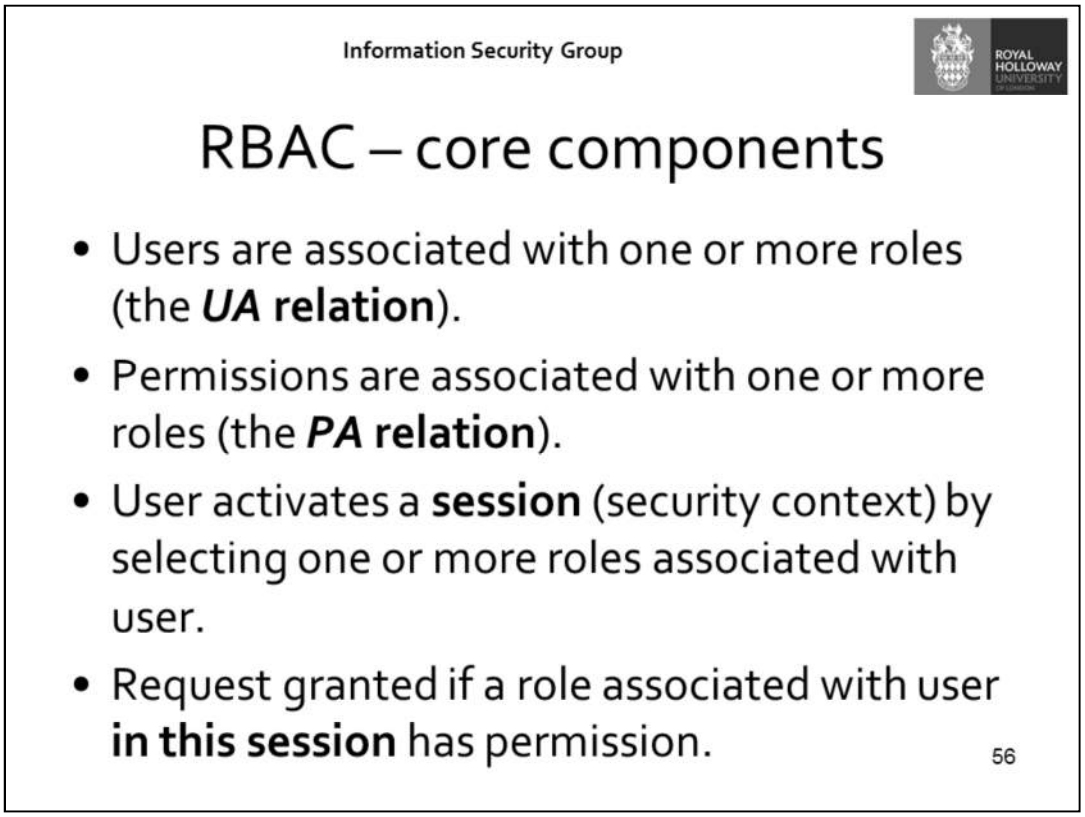
- the notion of role acts as a bridge between users and objects;
- it reduces the complexity of configuring the authorisation policy;
- it can be used to automate 'user provisioning' by integrating authorisation and human resources databases.

The motivating idea is that, in a typical organisation, there are many fewer roles than there are users. As a result it can save a lot of time to set up access rights for each role, and then assign users to roles as appropriate. A user might have more than one role, and the assignment of users to roles will typically change over time.



This diagram is taken from a presentation by Ed Coyne, available on the NIST website.

It shows in diagrammatic form how access rights (privileges or policy rules) are assigned to roles, and users (principals) are also assigned to roles. Users (principals) are not directly assigned access rights.



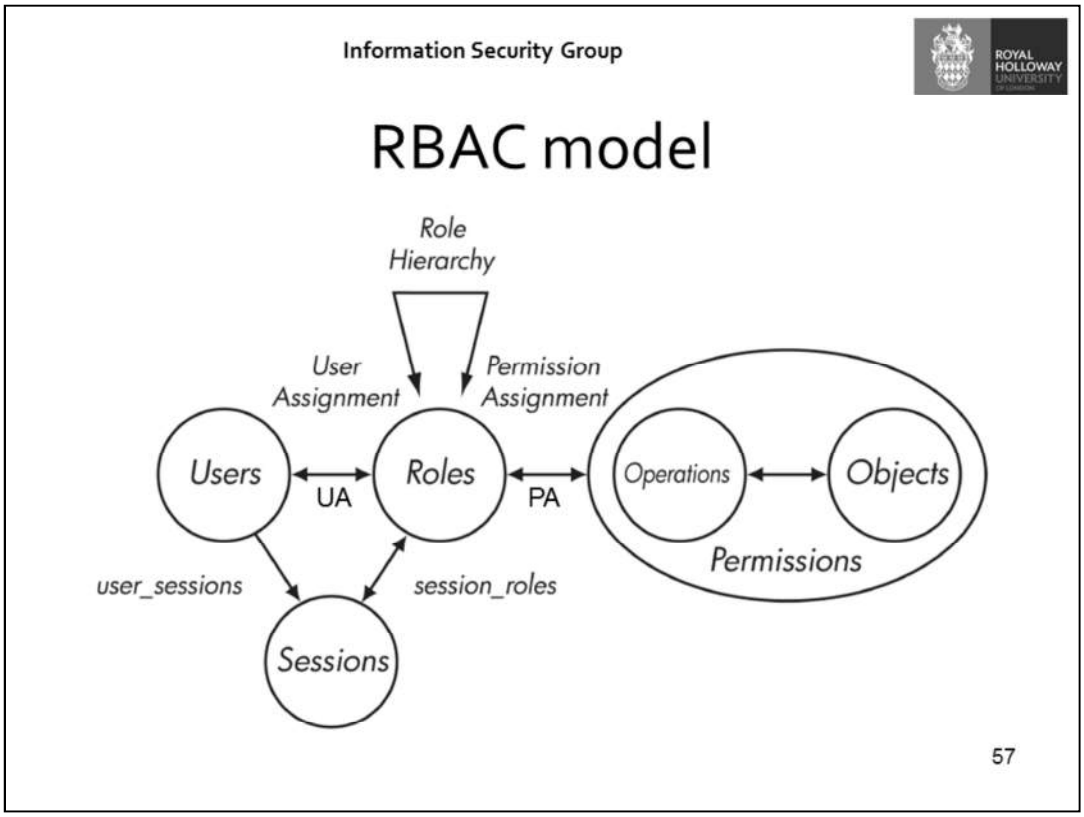
Formally:

• Principals (Users) are associated with one or more roles (the ***UA* relation**, where UA stands for 'User Assignment').

• Permissions are associated with one or more roles (the ***PA* relation**, where PA stands for 'Permission Assignment').

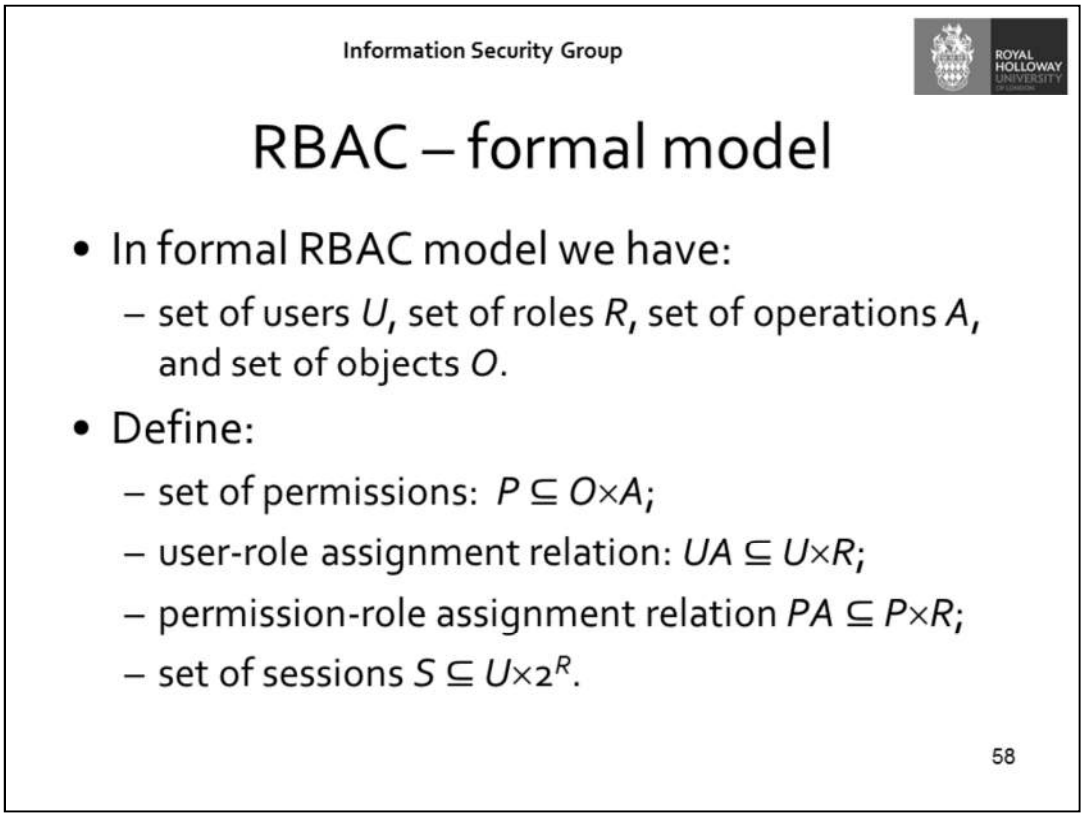
• A user activates a **session** (security context) by selecting one or more of the roles with which the user is associated.

• An access request is granted if one or more of the roles associated with the user **in this session** has the necessary permission.



This diagram is taken from a presentation by Ed Coyne, available on the NIST website.

Note that the 'role hierarchy' shown in this diagram only applies for hierarchical RBAC (see later slides).



In a formal RBAC model we have:

• a set of users U, a set of roles R, a set of operations A, and a set of objects O.

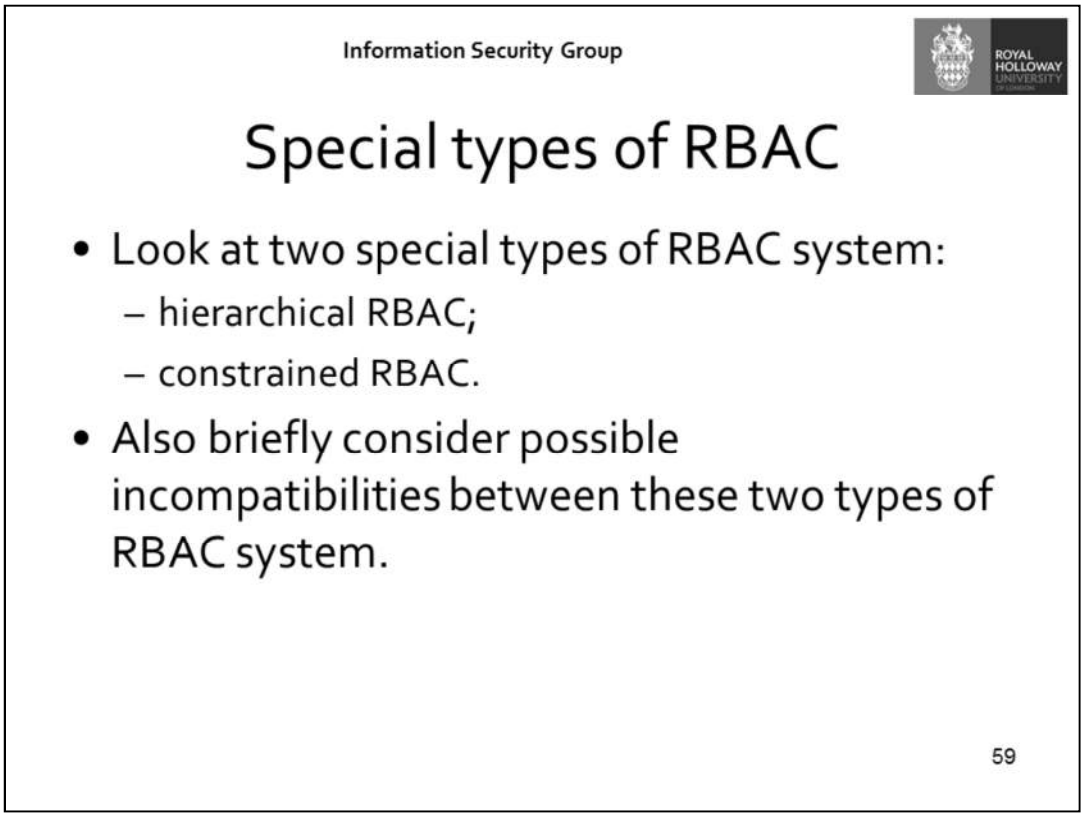
We define:

• the set of permissions: $P \subseteq O \times A$ (where $O \times A$ represents the set consisting of ordered pairs of elements from O and A), i.e. the set of permissions is a collection of pairs of objects and operations;

• the user-role assignment relation: $UA \subseteq U \times R$;

• the permission-role assignment relation $PA \subseteq P \times R$;

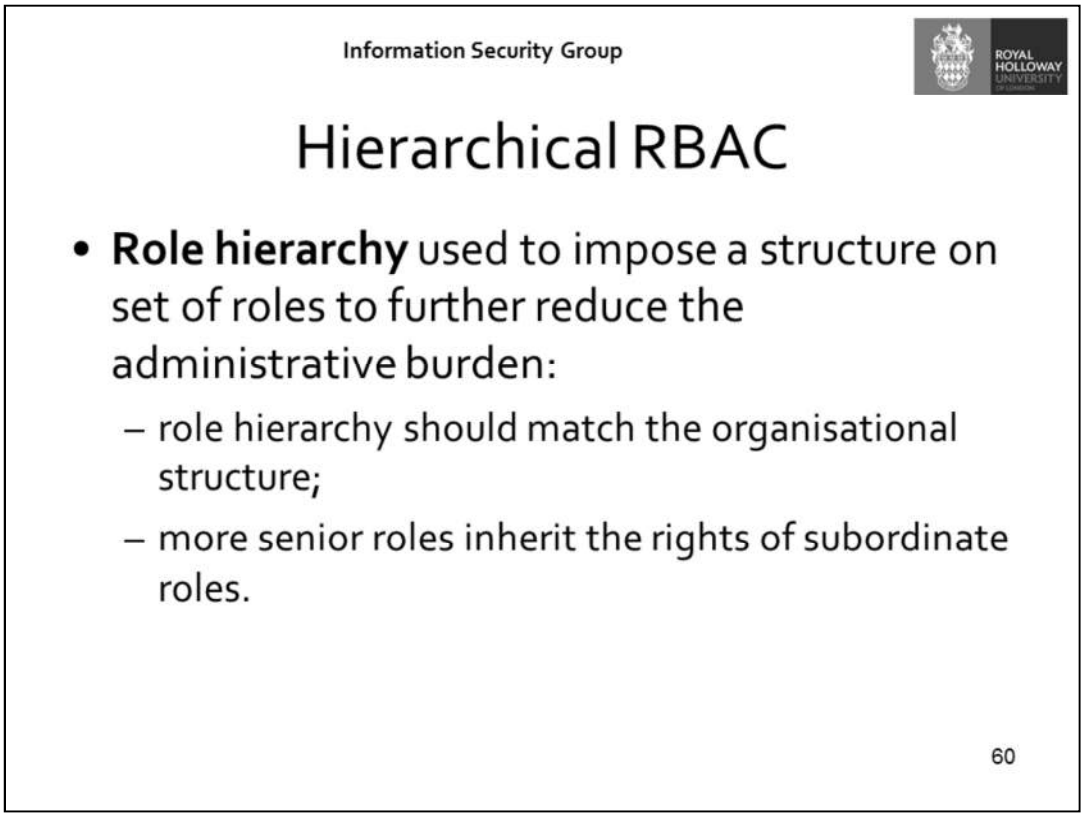
• the set of sessions $S \subseteq U \times 2^R$ (where 2^R represents the set of all subsets of R).



We look briefly at two special types of RBAC system:

- hierarchical RBAC;
- constrained RBAC.

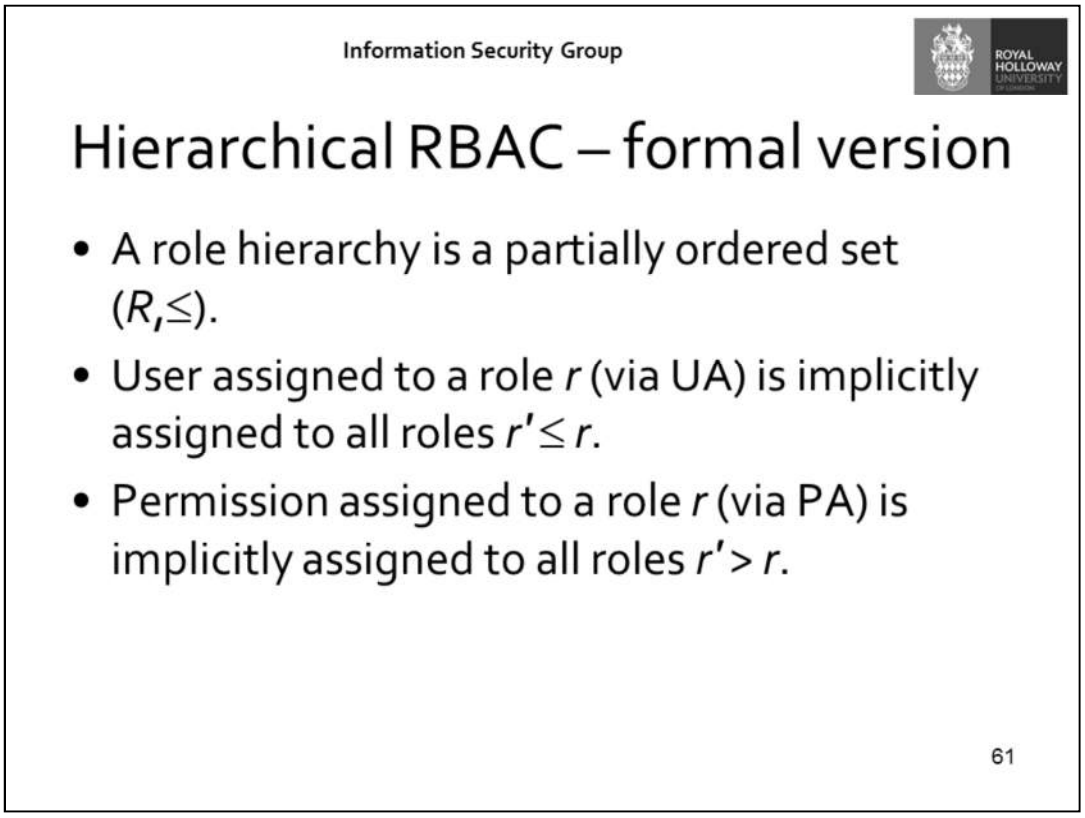
We also briefly consider possible incompatibilities between these two types of RBAC system.



In hierarchical RBAC, a **role hierarchy** is used to impose a structure on the set of roles in order to further reduce the administrative burden:

- the role hierarchy is intended to map on to the organisational structure.

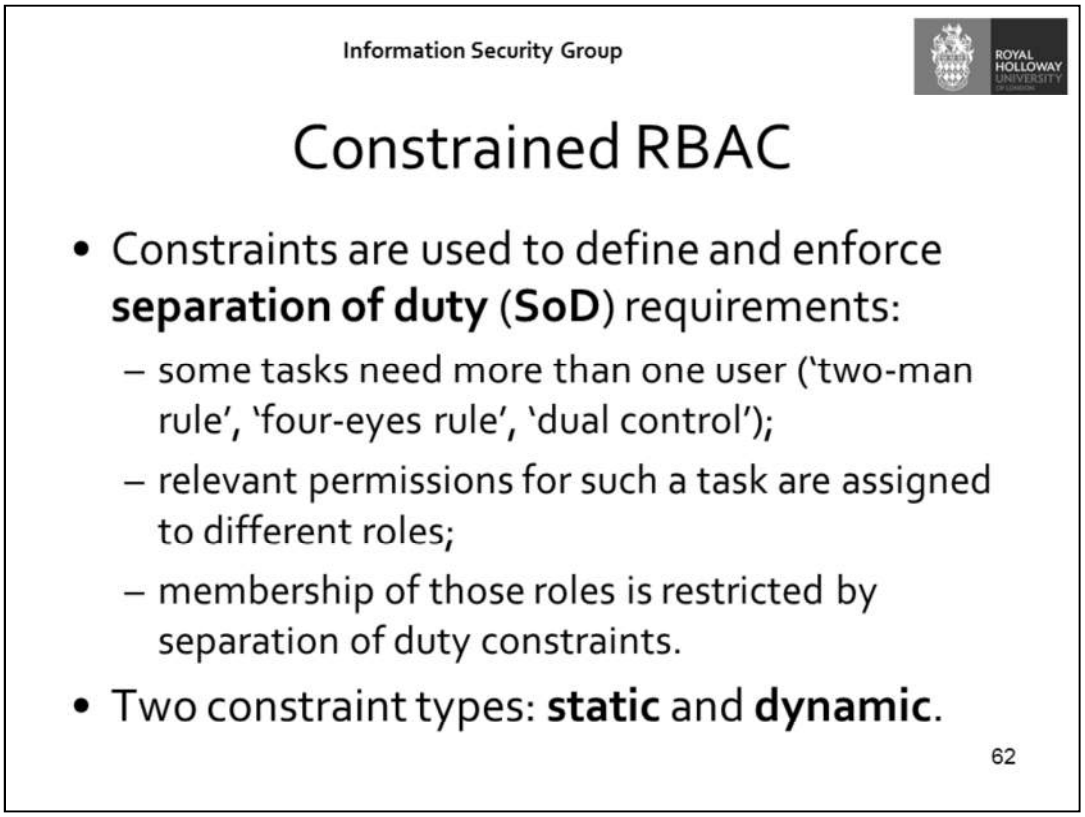
This is achieved by the fact that the more senior roles automatically inherit the rights of all roles that are subordinate in the hierarchy.



A role hierarchy is a partially ordered set (R,$\leq$), i.e. we define a partial ordering on the set of possible roles.

A user that is assigned to a role r (via the UA) is implicitly assigned to all subordinate roles $r' \leq r$.

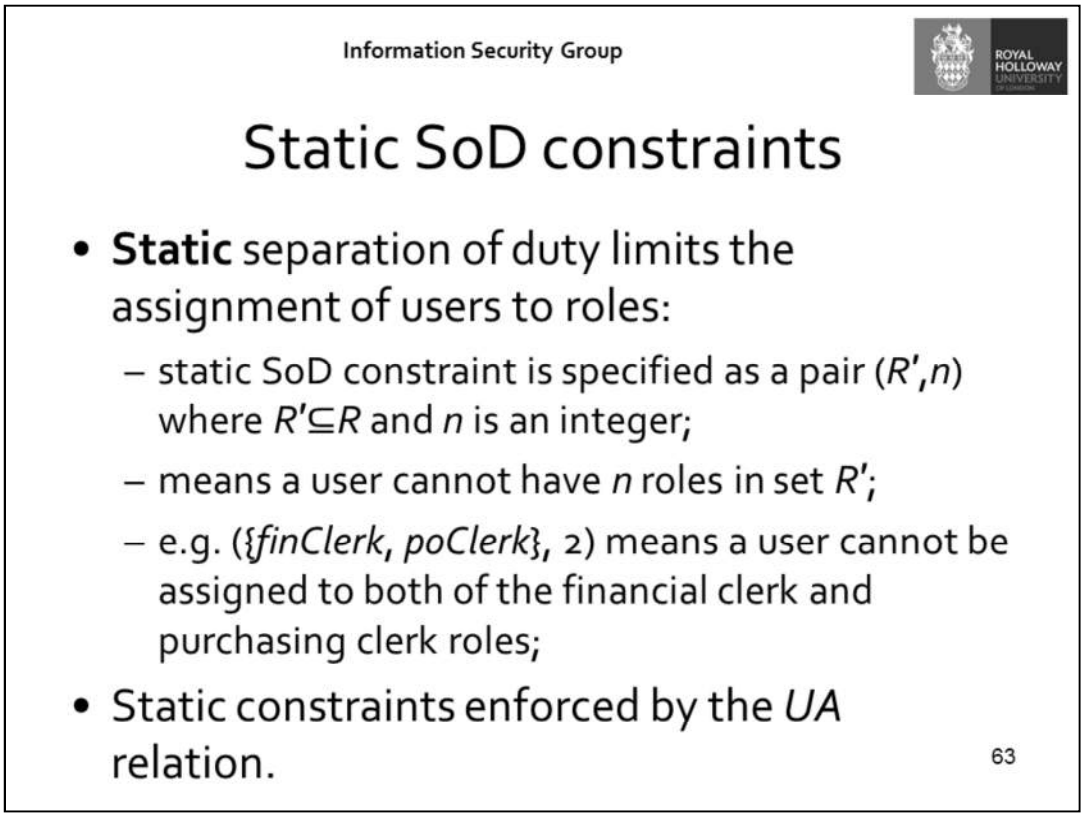
As a result, a permission that is assigned to a role r (via the PA) is implicitly assigned to all superior roles $r' > r$.



Constraints are used to define and enforce separation of duty (SoD) requirements:

- certain tasks may require more than one user to complete ('two-man rule', 'four-eyes rule', 'dual control');
- the relevant permissions for such a task are assigned to different roles;
- membership of those roles is restricted by separation of duty constraints.

There are two types of SoD constraint: **static** and **dynamic**, as we next discuss.



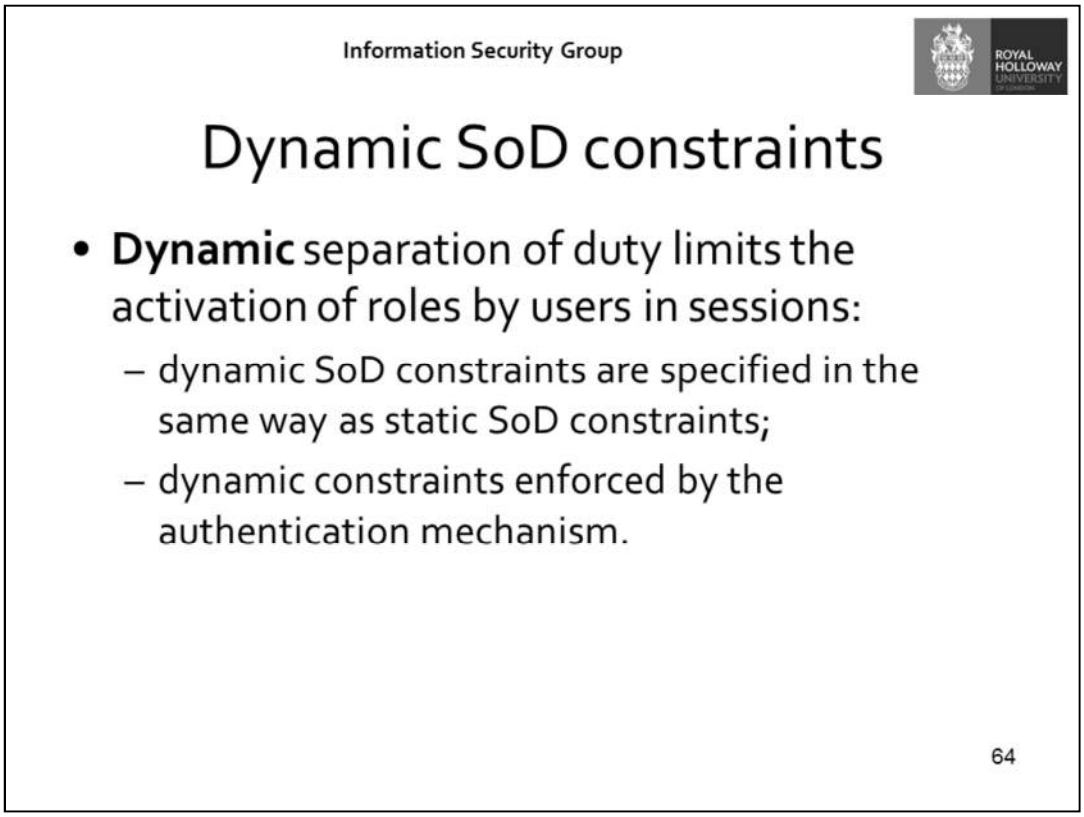
Static separation of duty limits the assignment of users to roles:

• a static SoD constraint is specified as a pair (R',n) where $R' \subseteq R$ and n is an integer;

• this means that it is not permitted to assign a user to as many as n roles in the set R';

• e.g. ({*finClerk*, *poClerk*}, 2) specifies that a user cannot be assigned to both the financial clerk and purchasing clerk roles;

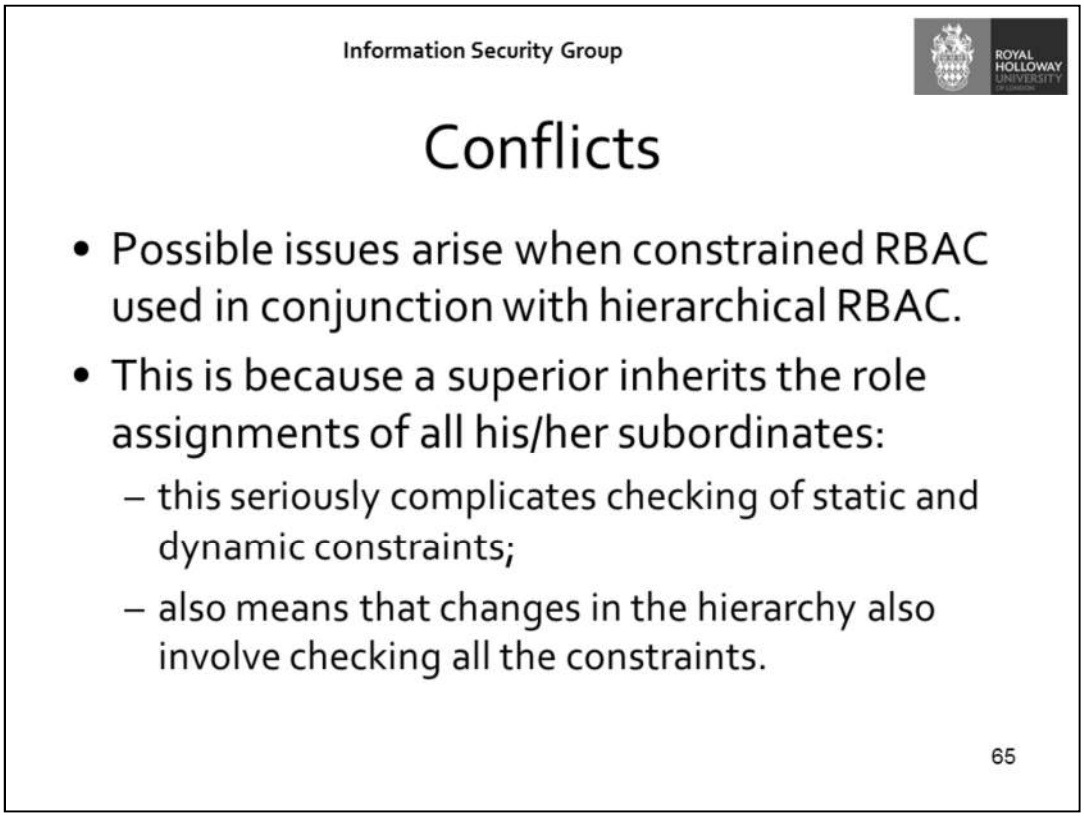
Static constraints are enforced by the *UA* relation. This is because the static constraints put restrictions on which roles can be assigned to users.



Dynamic separation of duty limits the activation of roles by users in sessions:

• dynamic SoD constraints are specified in the same way as static SoD constraints;

• dynamic constraints are enforced by the authentication mechanism; that is, when a user logs in and asks to be assigned certain roles, the constraint checking ensures that the user does not acquire a conflicting set of roles.

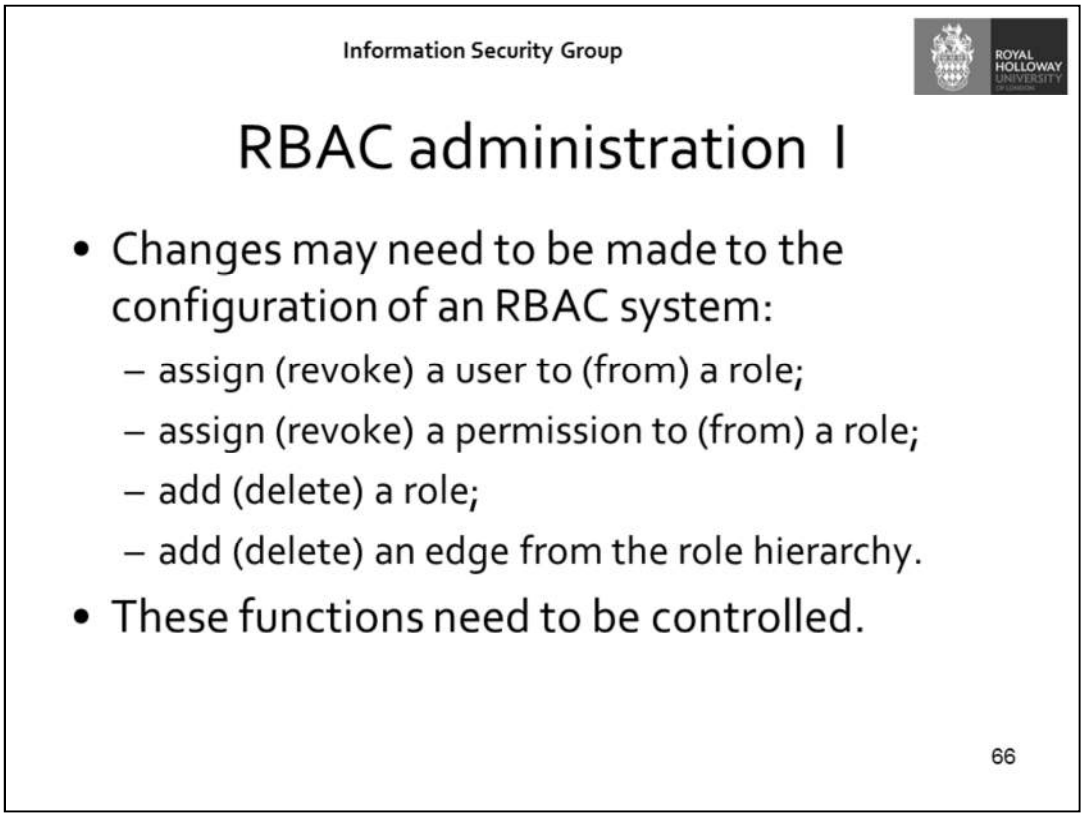


Possible issues arise when constrained RBAC is used in conjunction with hierarchical RBAC.

This is because a superior inherits the role assignments of all his/her subordinates:

- this seriously complicates checking of static and dynamic constraints;
- also means that changes in the hierarchy also involve checking all the constraints.

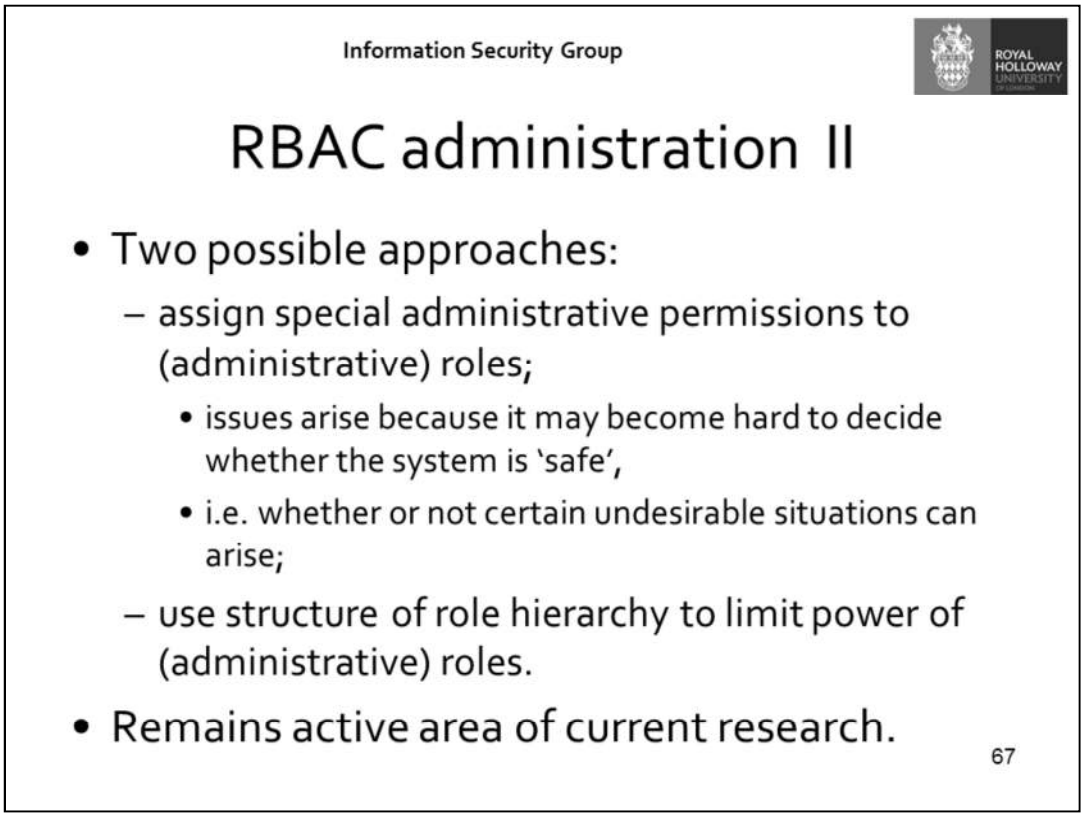
This could become very time-consuming in large systems.



Changes may need to be made to the configuration of an RBAC system. For example, an administrator may need to:

- assign (revoke) a user to (from) a role;
- assign (revoke) a permission to (from) a role;
- add (delete) a role;
- add (delete) an edge from the role hierarchy.

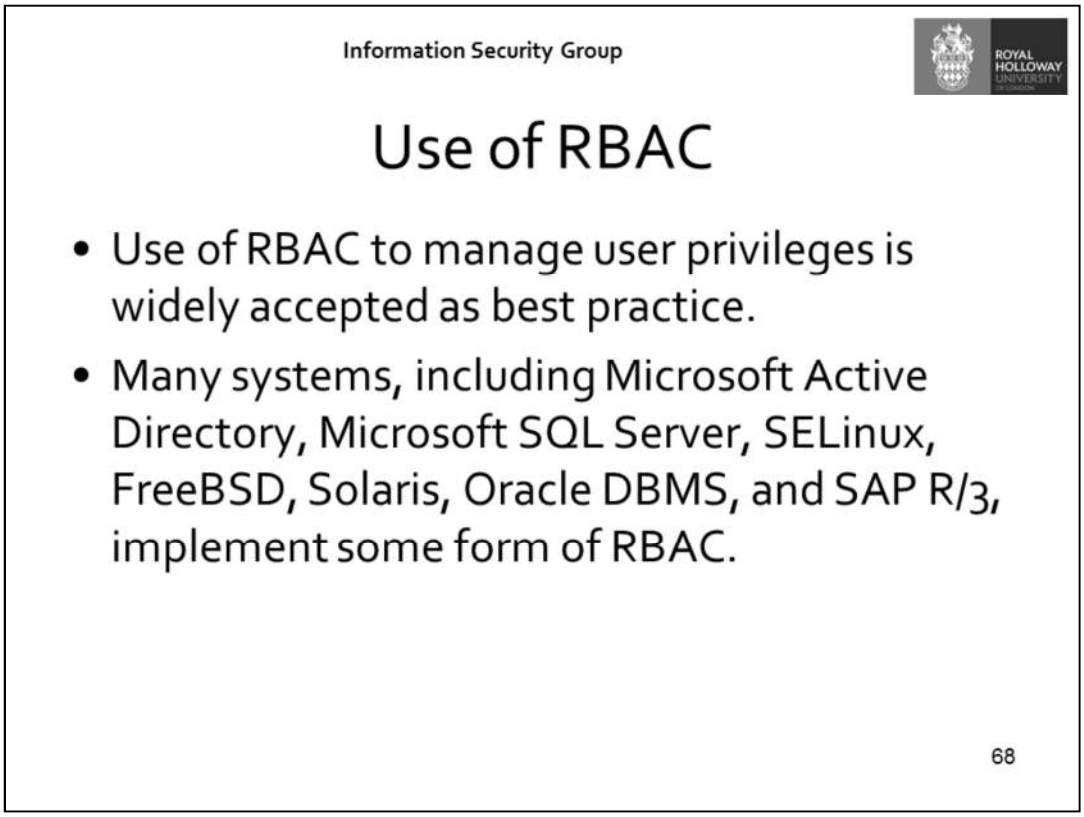
Obviously, these functions need to be very carefully controlled, or else the objectives of the system can be compromised.



There are two possible approaches to supporting and controlling administrative functions are as follows:

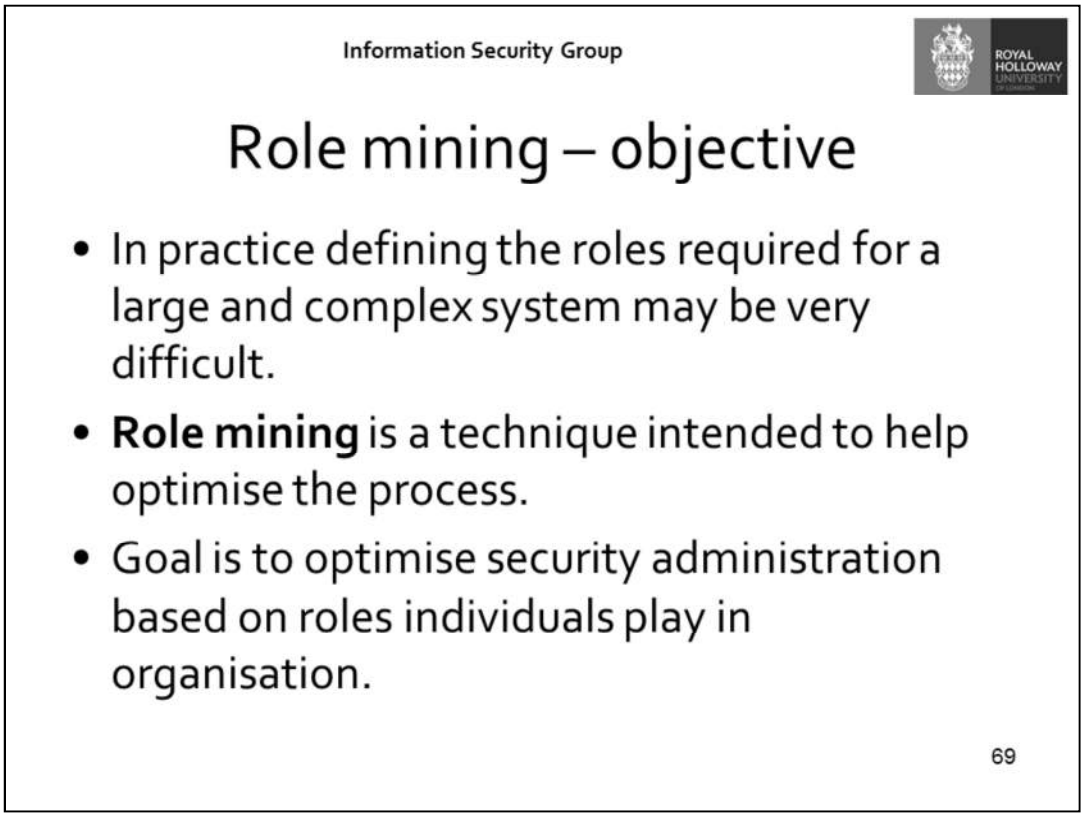
- assign special administrative permissions to the (administrative) roles;
    - issues arise because it may become hard to decide whether the system is 'safe',
    - i.e. whether or not certain undesirable situations can arise;
- use the structure of role hierarchy to limit the power of (administrative) roles.

This remains an active area for current research.



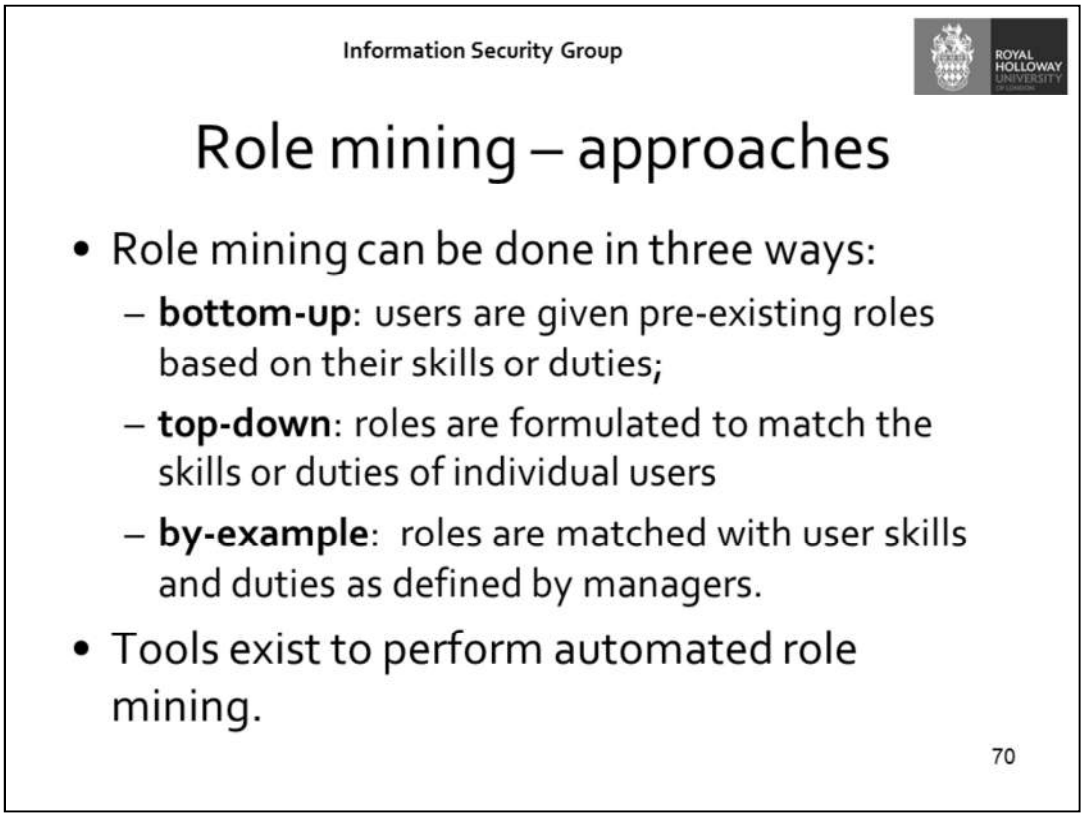
The use of RBAC to manage user privileges within a system or application is widely accepted as best practice.

Many systems, including Microsoft Active Directory, Microsoft SQL Server, SELinux, FreeBSD, Solaris, Oracle DBMS, and SAP R/3, effectively implement some form of RBAC.



In practice defining the roles required for a large and complex system may be very difficult. *Role mining* is a technique intended to try to optimise the process; that is , the iit is hoped that by following a defined procedure for defining roles, the end result will work better than by simply defining roles in an ad hoc way.

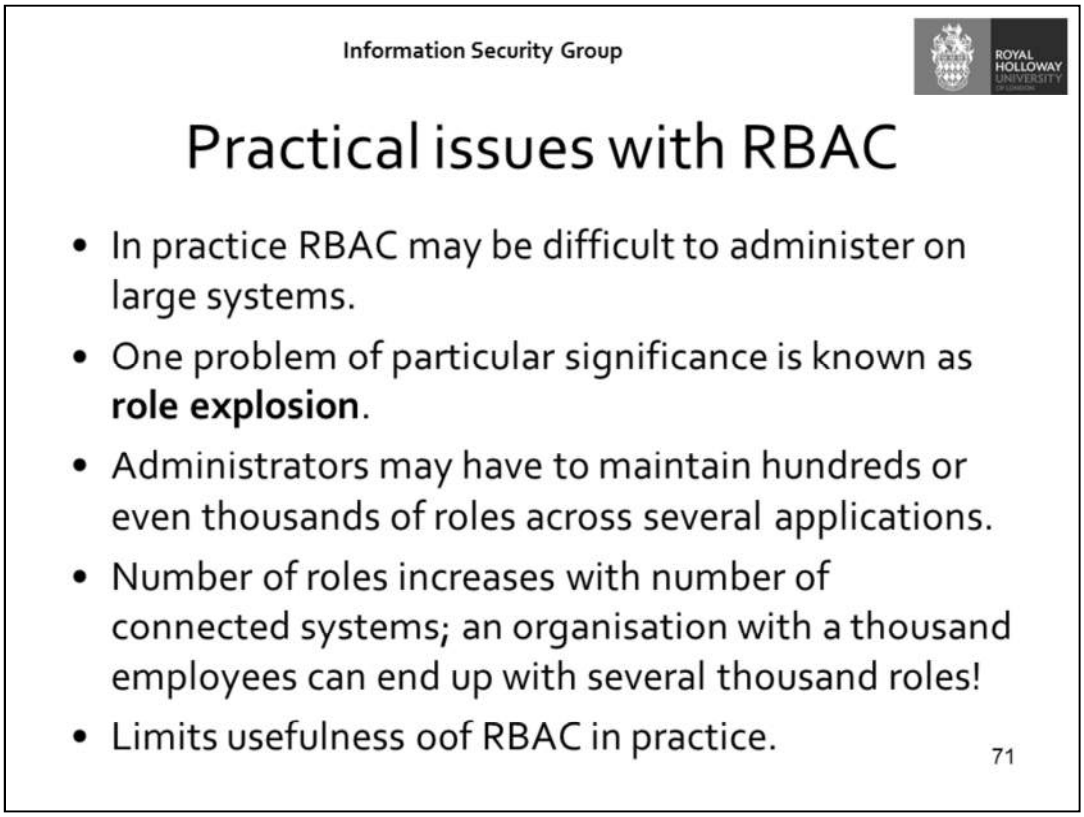
The main goal is to try to optimise security administration based on the roles individuals actually play in the organisation.



Role mining can be done in three ways:

- *bottom-up*: users are given pre-existing roles based on their skills or duties;
- *top-down*: roles are formulated to match the skills or duties of individual users
- *by-example*: roles are matched with user skills and duties as defined by managers.

Tools exist to perform automated role mining.



In practice RBAC may be difficult to administer on large systems.

One problem of particular significance is known as **role explosion**.

Administrators may have to maintain hundreds or even thousands of roles across several applications.

Number of roles increases with the number of connected systems, and an organisation with only a thousand employees can easily end up with several thousand roles.

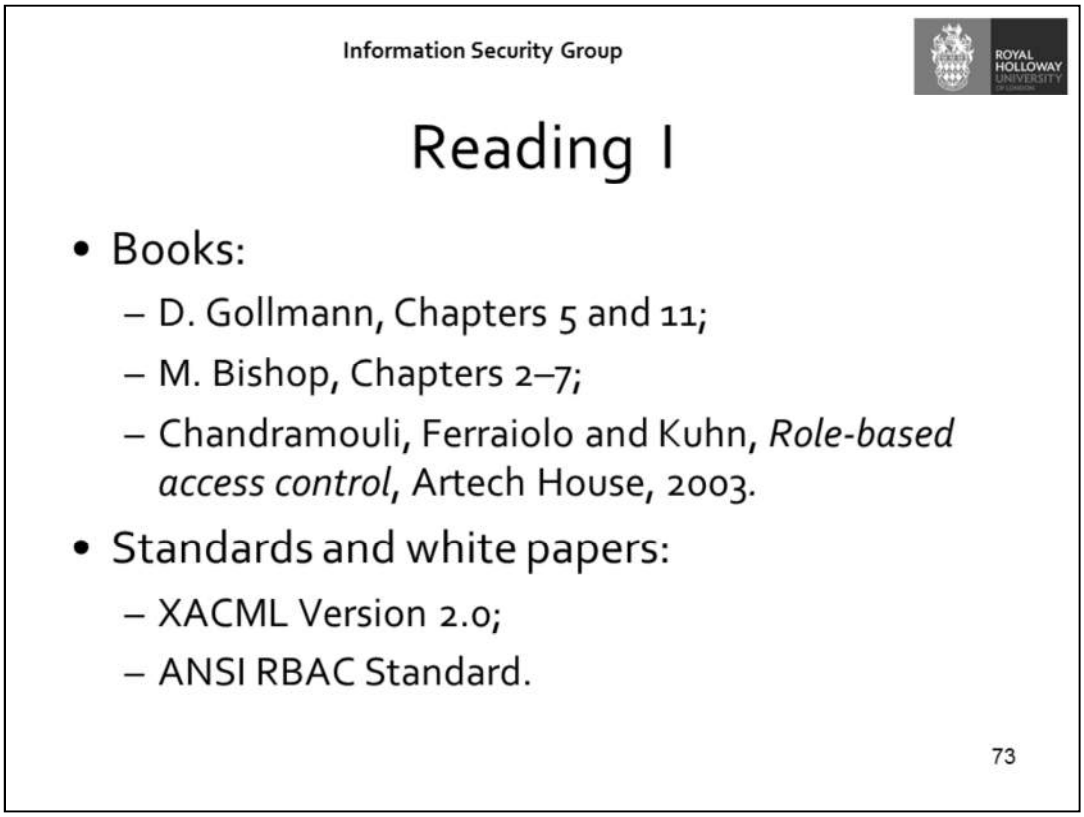
This problem seriously limits the usefulness of RBAC in practice.

Information Security Group

Agenda

- Access control basics
- ACLs and capabilities
- Information flow policies
- Bell-LaPadula Model
- Role-Based Access Control
- Resources

72

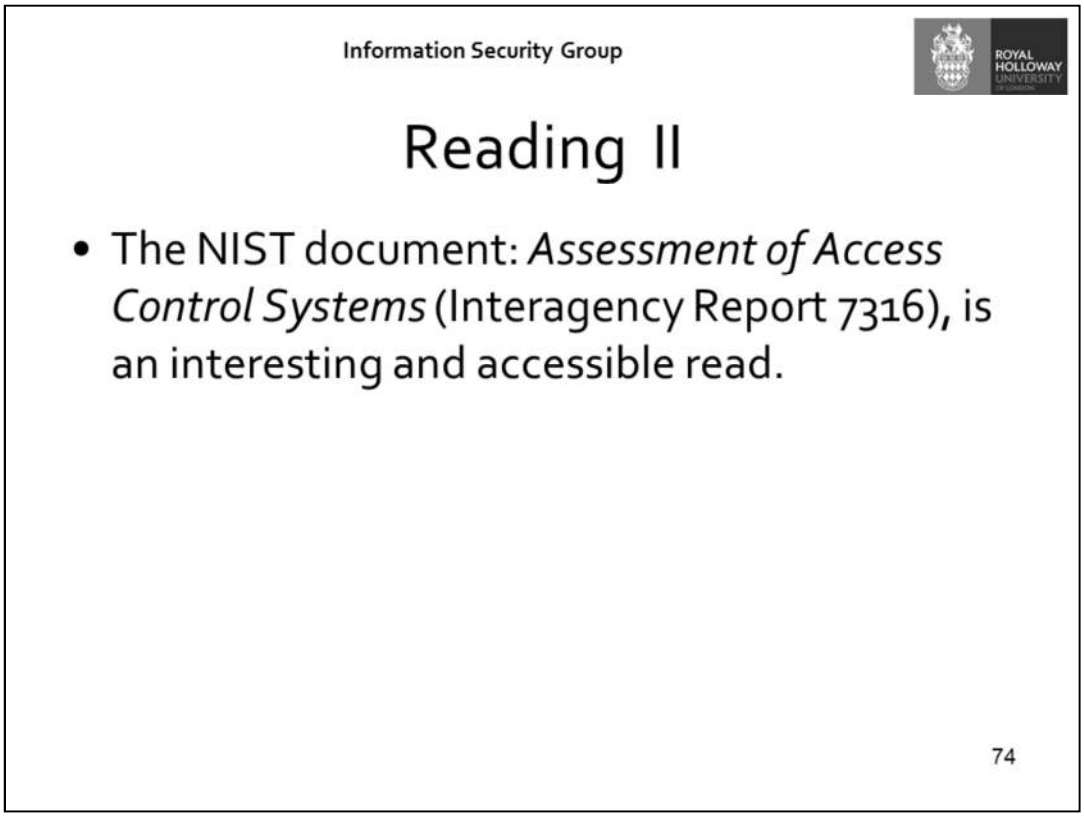


Books:

- D. Gollmann (3rd edition), Chapters 5 and 11;
- M. Bishop, Chapters 2–7;
- Chandramouli, Ferraiolo and Kuhn, *Role-based access control*, Artech House, 2003.

Standards and white papers:

- XACML Version 2.0 – see:
  http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf
- ANSI RBAC Standard – see:
  www.cs.purdue.edu/homes/ninghui/readings/AccessControl/ANSI+INCITS+359-2004.pdf



The NIST document: *Assessment of Access Control Systems* (Interagency Report 7316), is an interesting and accessible read. It is available here:

http://csrc.nist.gov/publications/nistir/7316/NISTIR-7316.pdf