Information Security Group

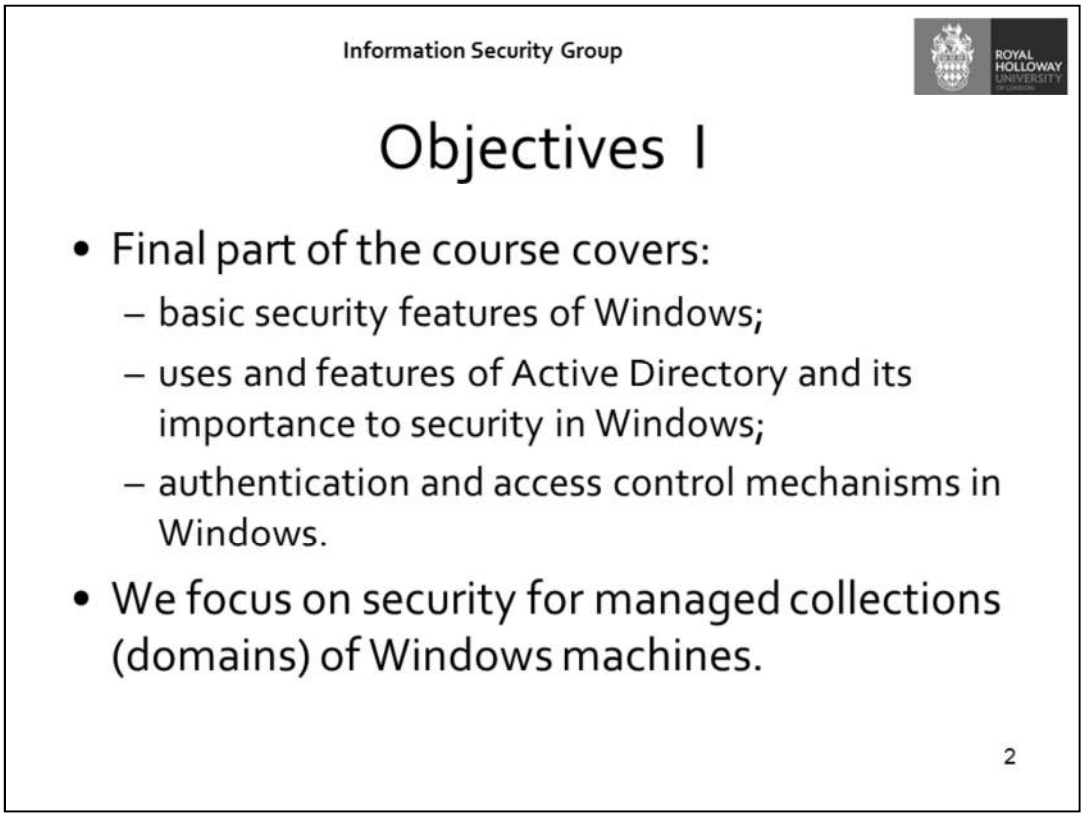

# IY5512 Computer Security
# Part 7b: Windows security

Chris Mitchell

me@chrismitchell.net

http://www.chrismitchell.net

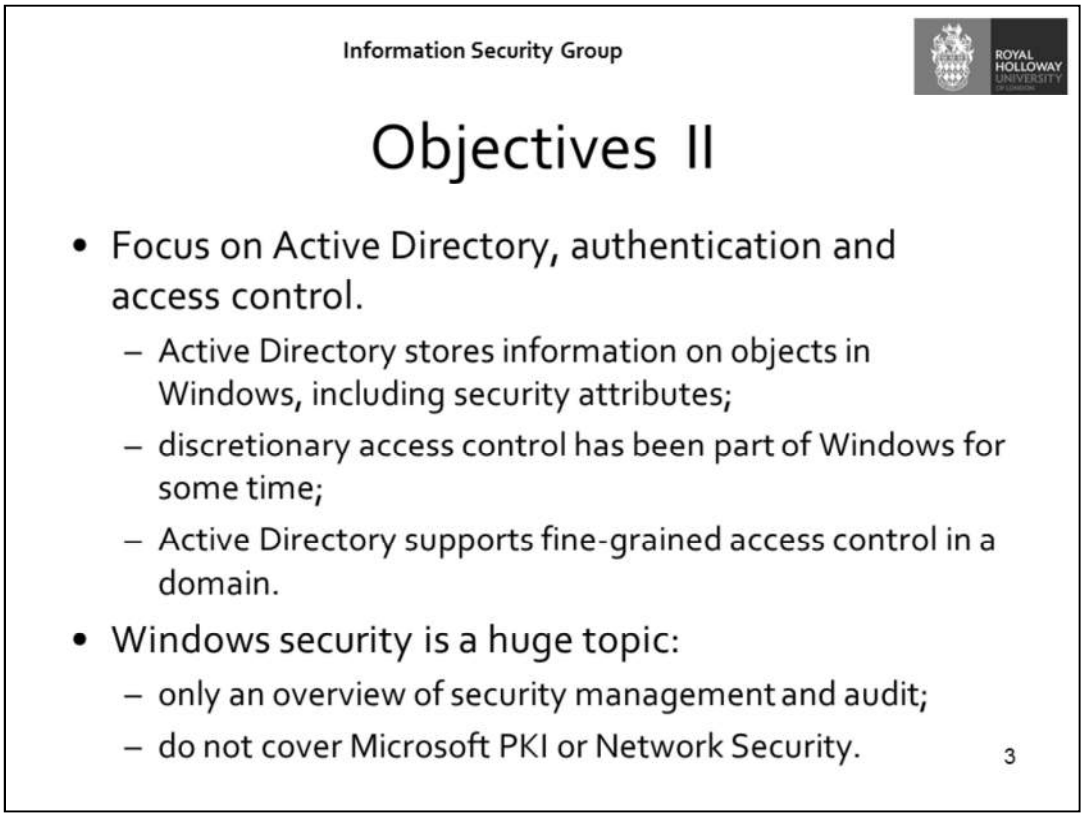
1




This (final) part of the course covers:

- the basic security features of Windows;
- the uses and features of Active Directory, and its importance to how security is provided in managed Windows environments;
- the authentication and access control mechanisms provided in Windows.

We look at security from the perspective of a managed set of Windows machines, known as a **domain**, i.e. we focus primarily on the security issues that apply within an organisation that wishes to manage a networked collection of PCs using the Windows features designed to support this.




We focus on Active Directory, authentication and access control, key elements for managing security in an organisation.

• Active Directory is a repository for information on (managed) objects in Windows, including their security attributes;

• a full-featured discretionary access control system has been a feature of Windows for a number of years;

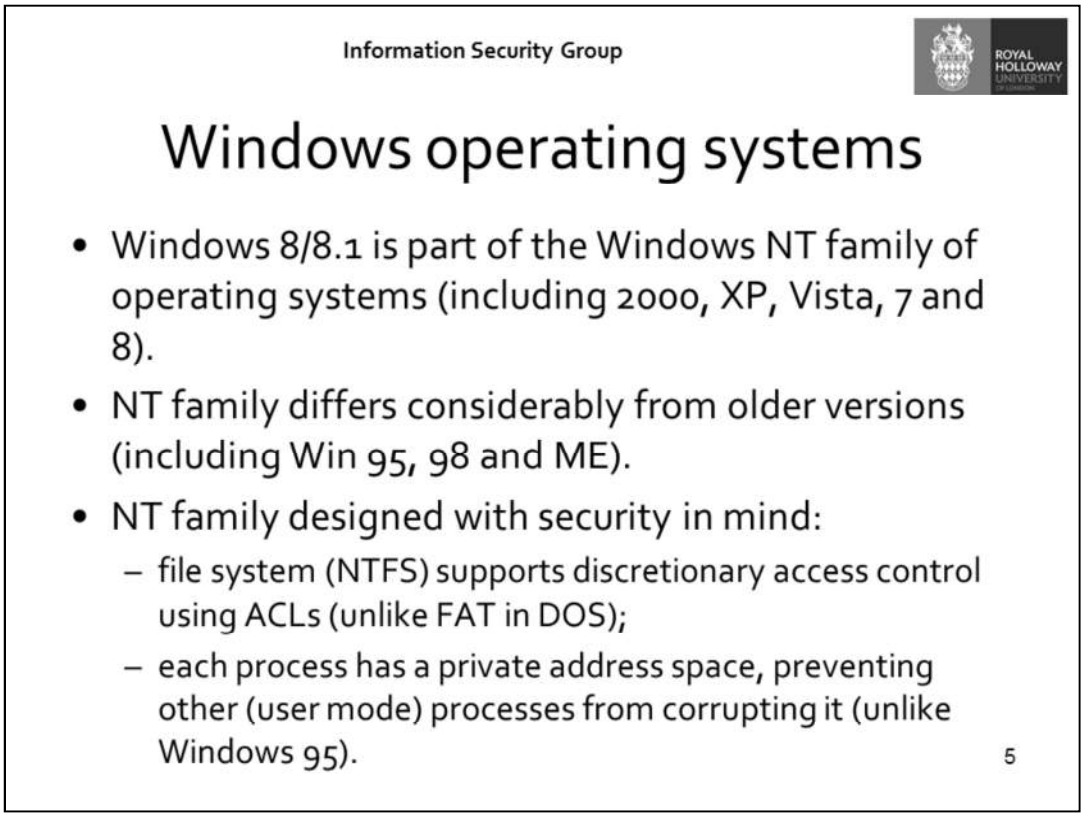
• Active Directory provides support for fine-grained access control within a domain.

Windows security is a huge topic, and hence some features receive little or no attention here. For example:

• we only give a brief overview of security management and audit;

• we will not cover the Microsoft PKI, the Microsoft cryptographic API, or Network Security.


We start by reviewing the key components supporting security in Windows.

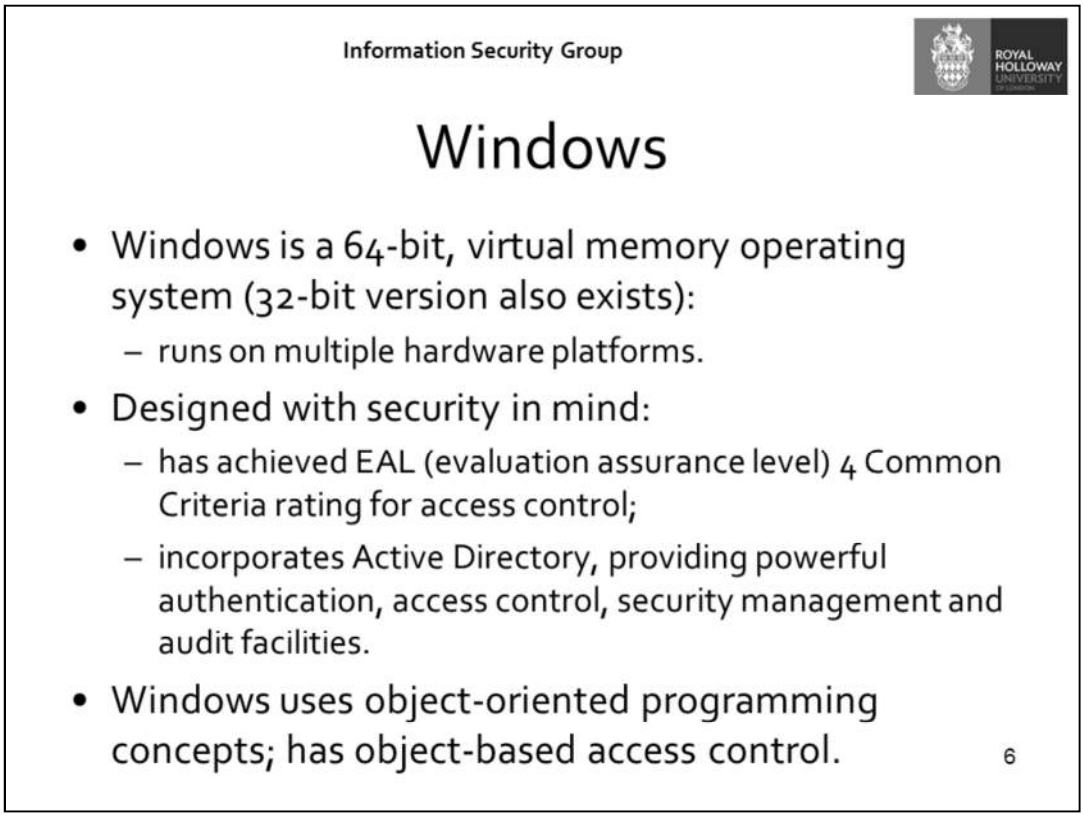



Windows 8 (and 8.1) and Windows Server 2012 are part of the Windows NT family of operating systems (including 2000, XP, Vista, 7 and 8, as well as the Windows Server family). The Windows NT family differs considerably from older versions of Windows (including Windows 95, 98 and ME).

Of particular importance for this course is the fact that the Windows NT family has been designed with security in mind:

• its file system (NTFS = NT File System) supports discretionary access control in the form of ACLs (unlike the FAT (File Allocation Table) file system used in DOS);

• each process has its own private address space, so preventing other (user mode) processes from corrupting it (unlike in Windows 95 and previous versions of Windows).



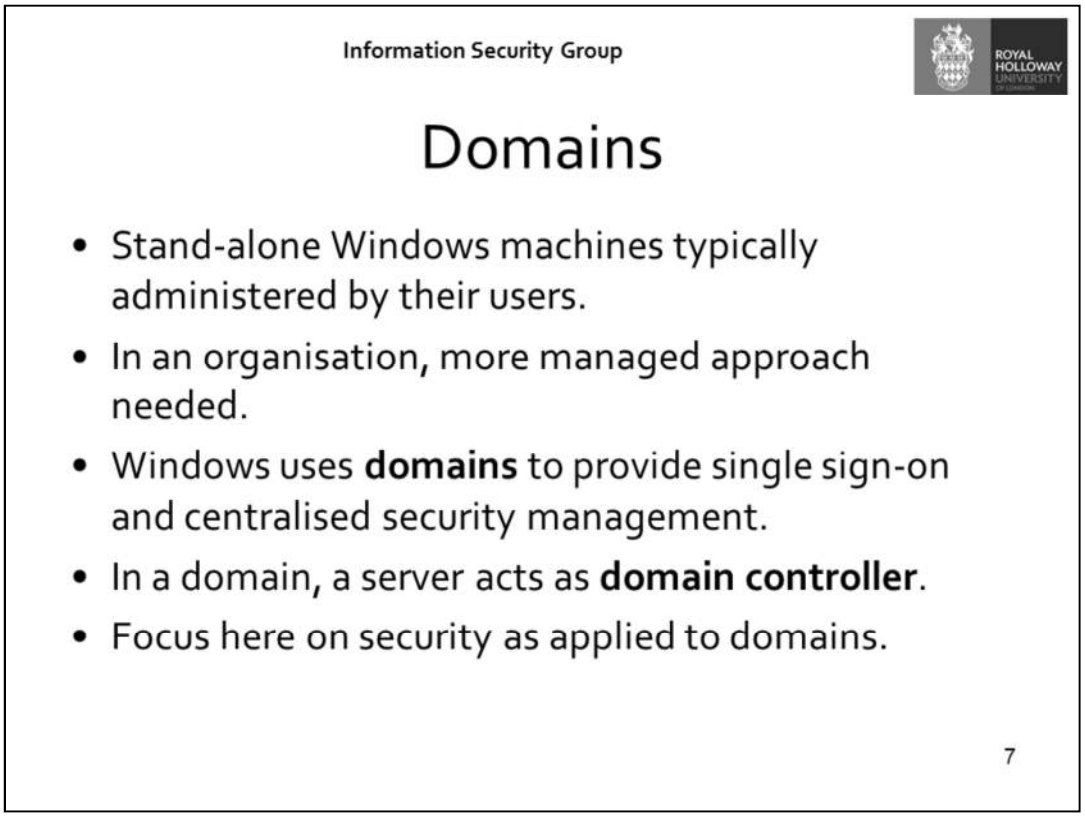

Windows is a 64-bit, virtual memory operating system that runs on multiple hardware platforms. A 32-bit version also exists, but the differences are not discussed in this course.

It has been designed with security in mind:

• it has achieved an EAL (evaluation assurance level) 4 Common Criteria rating for access control;

• it incorporates Active Directory, which provides powerful authentication, access control, security management and audit facilities.

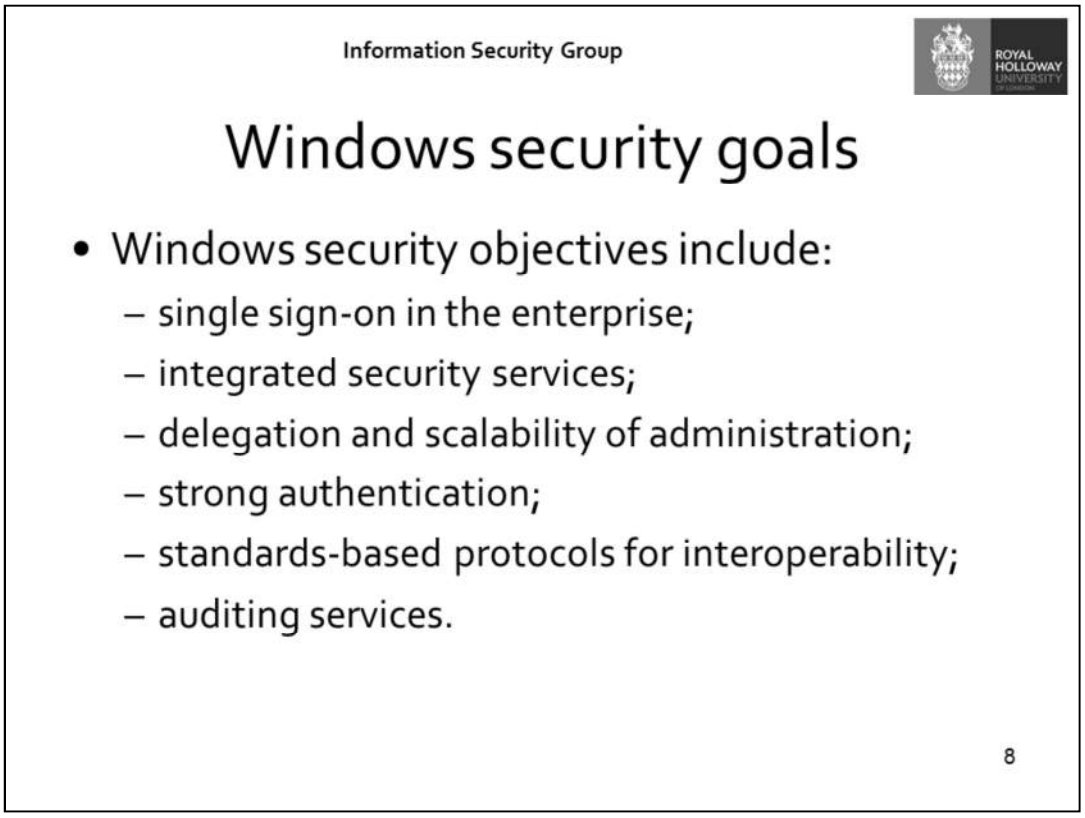
Windows uses object-oriented programming concepts, and has an object-based access control model.




Stand-alone Windows machines are typically administered by their users. However, in an organisation, a more managed approach is usually needed.

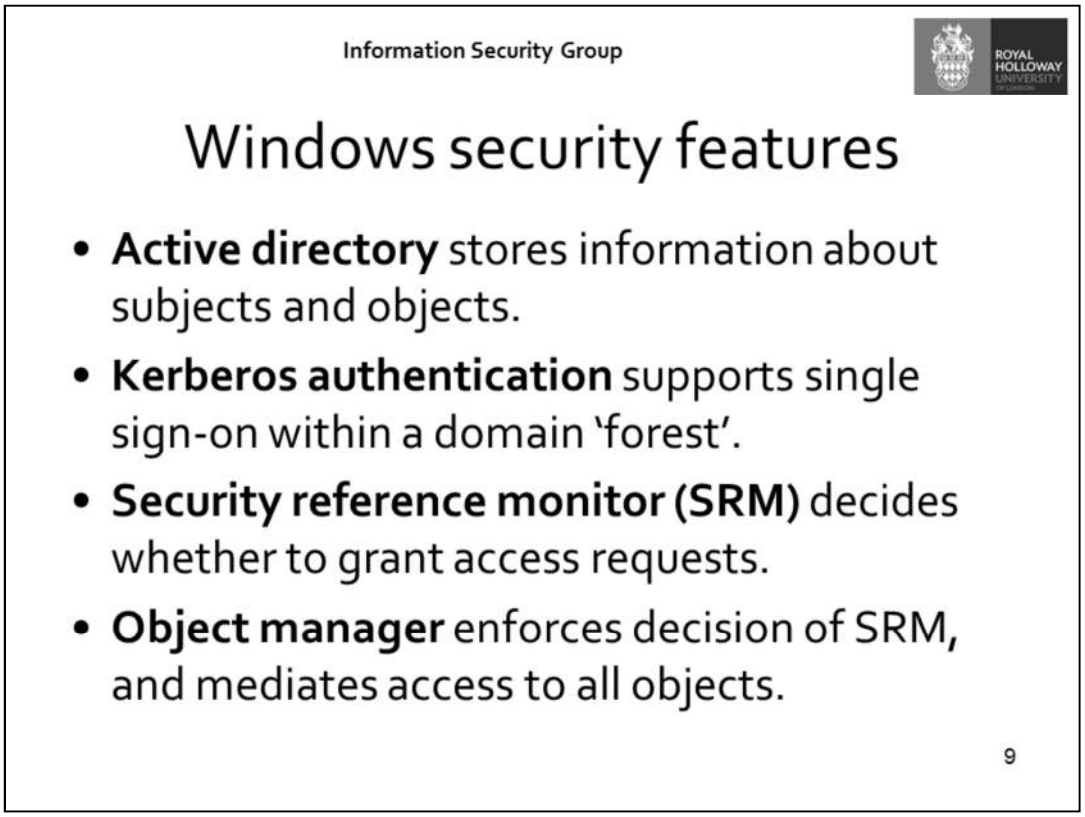
Windows uses the notion of **domains** to provide single sign-on and centralised security management functions across an interconnected set of machines. Within a domain, at least one server acts as **domain controller**.

As previously mentioned, the focus in this course is on Windows security as applied to domains.




The security objectives of Windows include supporting the following functions:

- single sign-on in the enterprise;
- integrated security services;
- delegation and scalability of administration;
- strong authentication;
- standards-based protocols for interoperability;
- auditing services.

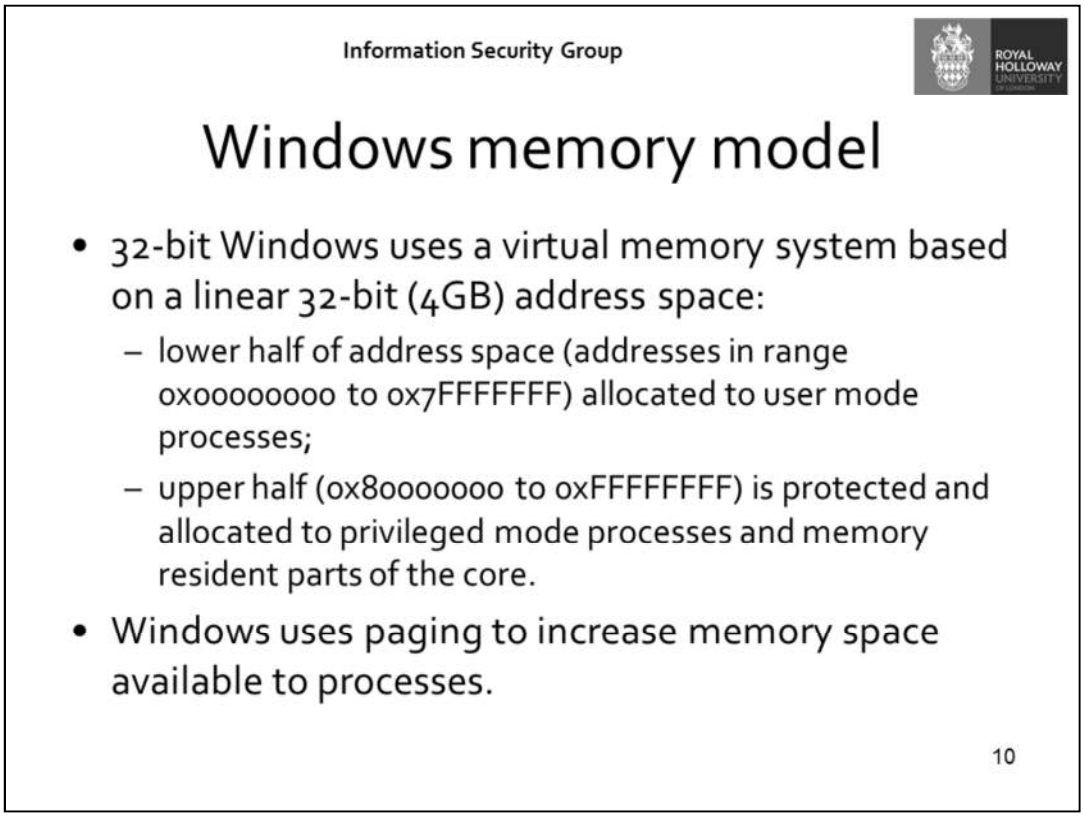



**Active directory** is used to store information about all subjects and objects, using object-based technology.

The **Kerberos authentication protocol** supports single sign-on within a domain 'forest' (a collection of closely related domains).

The **Security reference monitor (SRM)** is used to decide whether or not to grant an access request.

The **Object manager** enforces the decisions of the SRM, and mediates access to all objects.



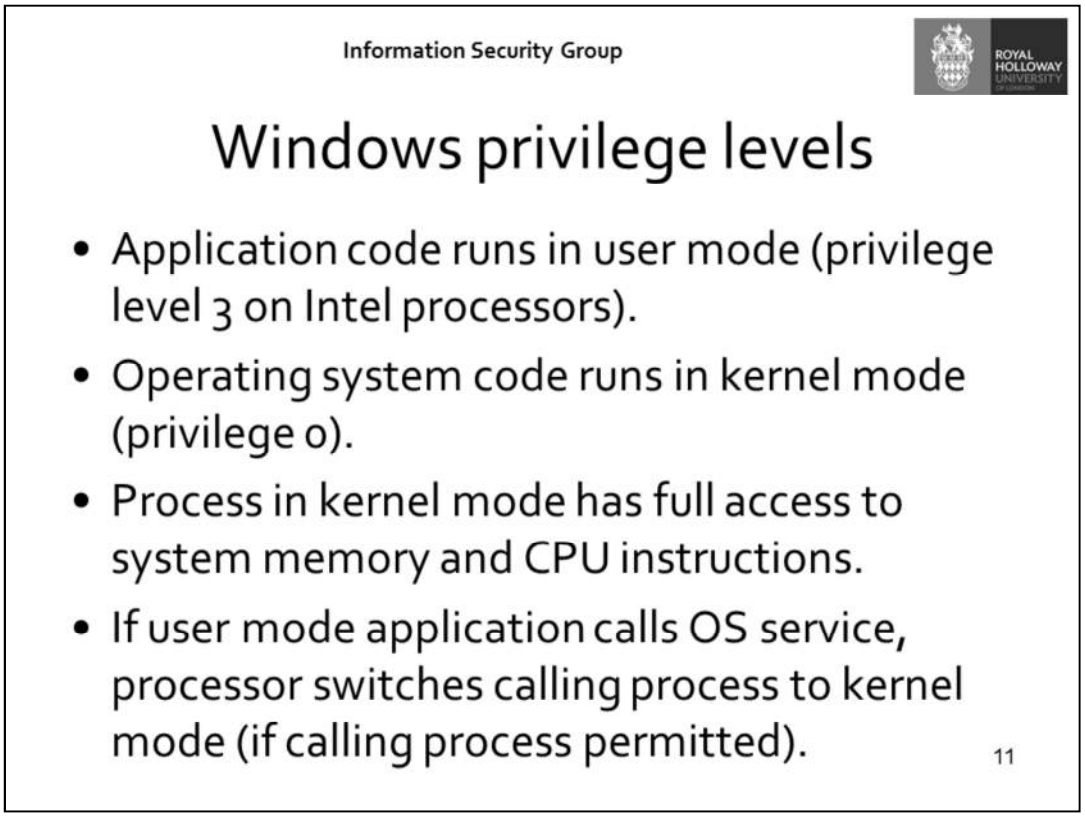

32-bit Windows implements a virtual memory system based on a linear 32-bit (4GB) address space:

• the lower half of this address space (addresses in the range 0x00000000 to 0x7FFFFFFF) is allocated to user mode processes;

• the upper half of the address space (0x80000000 to 0xFFFFFFFF) is protected, and is allocated to privileged mode processes and memory resident parts of the core.

Windows uses paging in order to increase the size of the memory space that is available to processes.

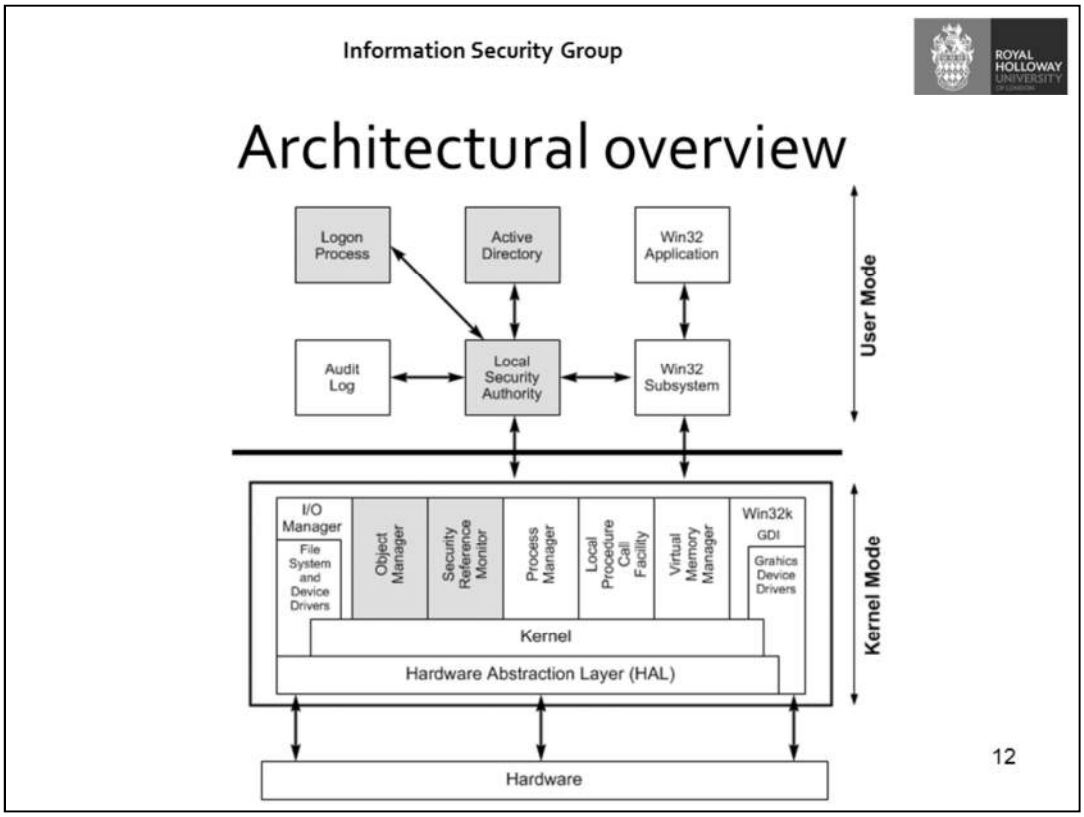
However, the 4GB address space means that 32-bit Windows is unable to make use of more than 4GB of physical memory. This limit is removed in 64-bit Windows.



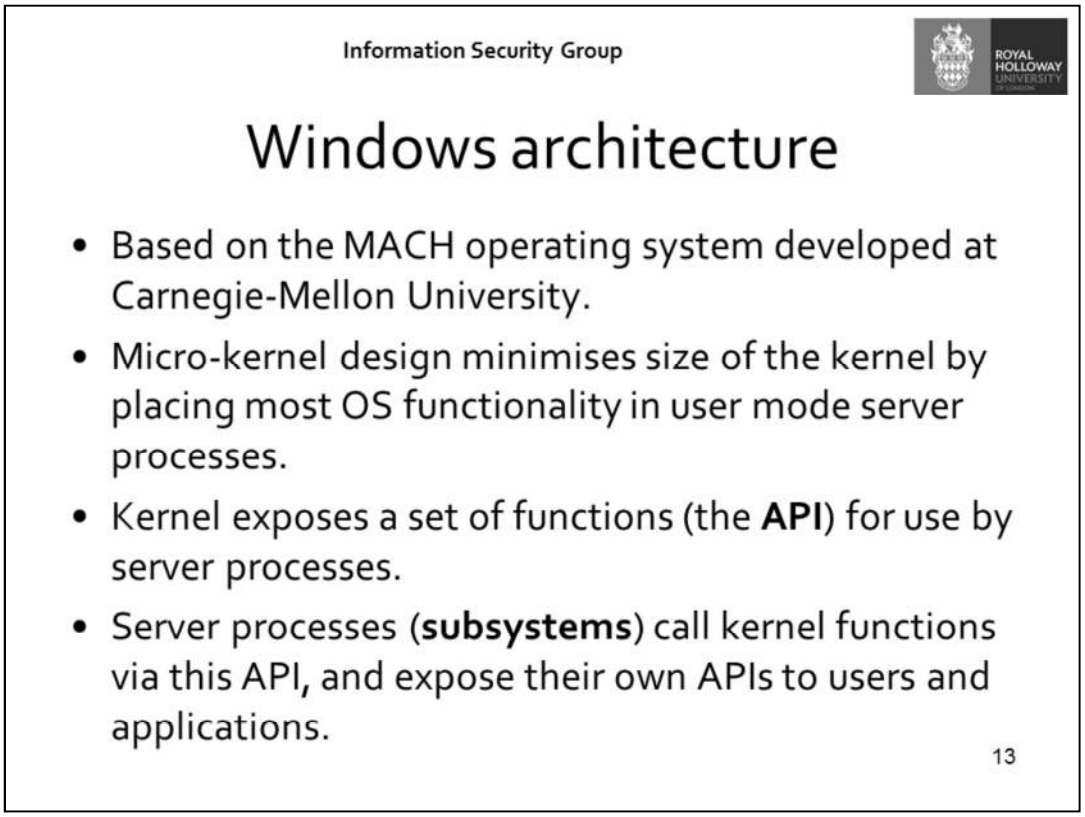

Application code runs in user mode, i.e. privilege level 3 (the least privileged level) on Intel processors. The operating system code runs in kernel mode, i.e. privilege level 0 (the most privileged level).

When a process runs in kernel mode it has full access to system memory and to the full set of CPU instructions.

When a user mode application calls an OS service, the processor traps the call and switches the calling process to kernel mode (if the calling process is permitted to do so).




The diagram provides a simplified overview of the architecture of 32-bit Windows, showing which tasks run in system (kernel) mode and which run in user mode.

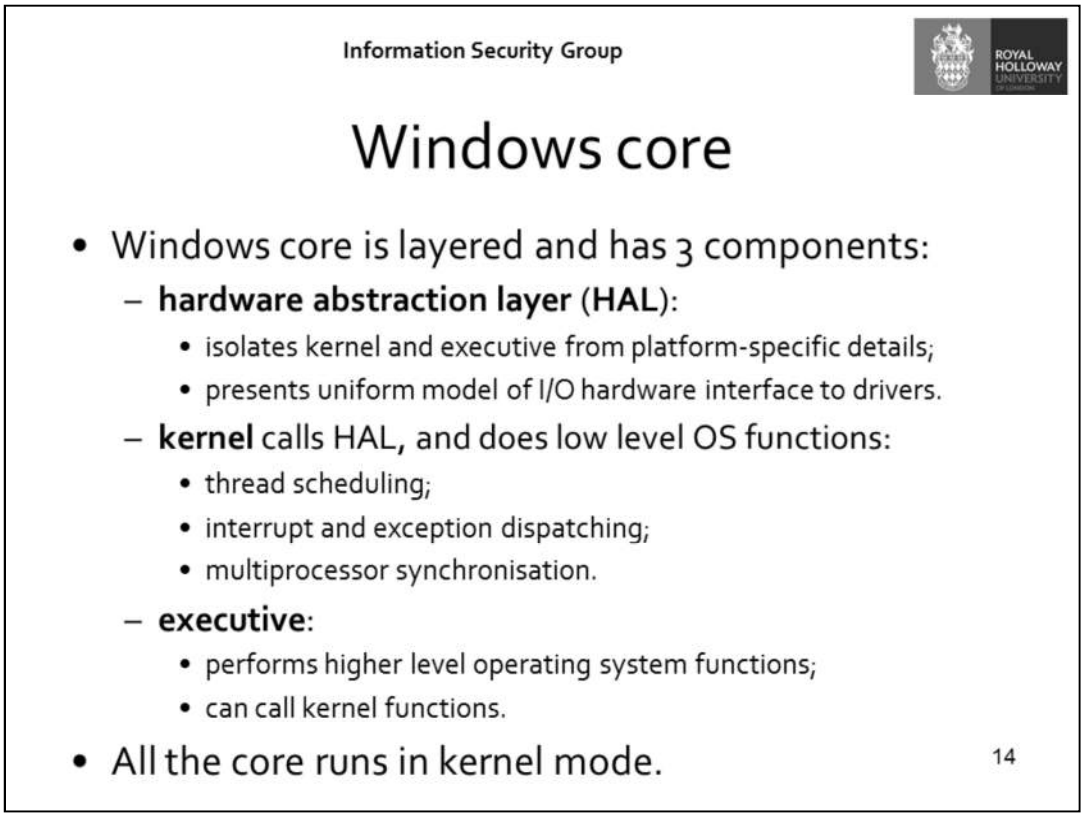



Windows is based on the MACH operating system developed at Carnegie-Mellon University (in Pittsburgh, in the US).

The micro-kernel design minimises the size of the kernel by placing the majority of OS functionality in user mode server processes;

The kernel exposes a set of functions that can be used by the server processes. This set of functions is called an API (**application programming interface**)

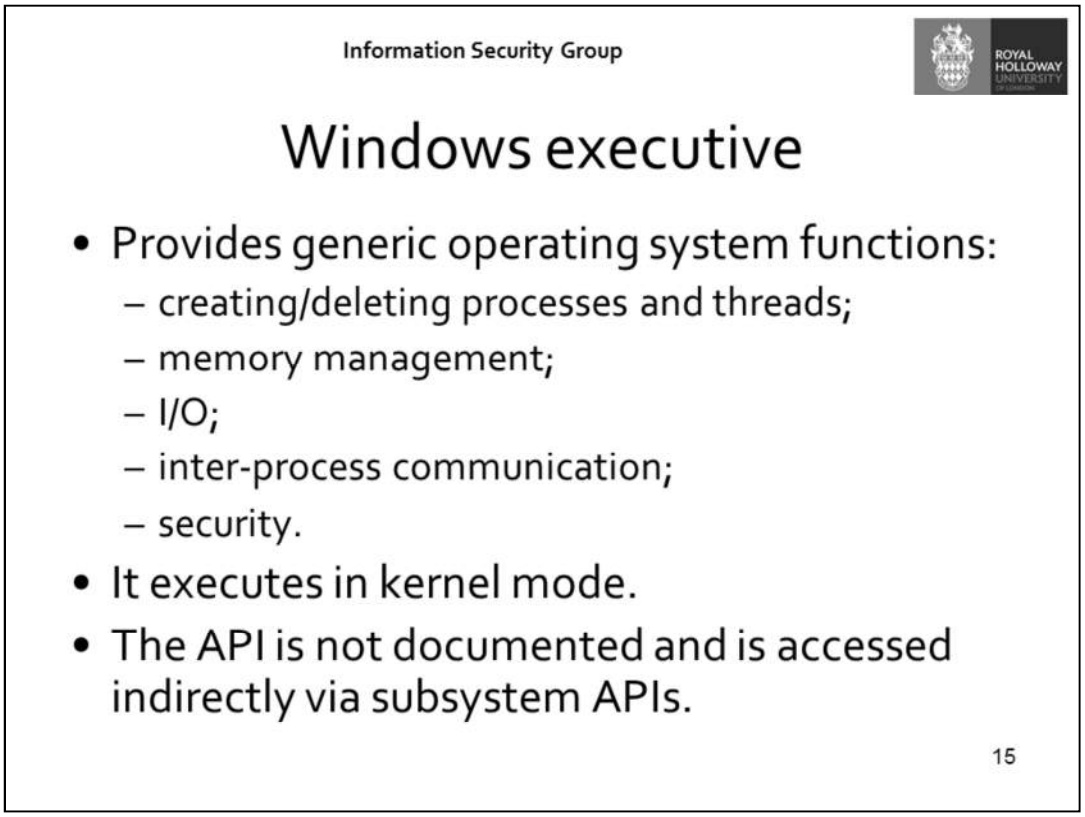
The server processes (called **subsystems** in Windows) call kernel functions via this API, and expose their own APIs to users and applications.




The Windows core has a layered architecture, and contains three different components:

- the **hardware abstraction layer** (**HAL**):
  - isolates the kernel and the executive from platform-specific details; and
  - presents a uniform model of the I/O hardware interface to drivers.
- the **kernel** can call HAL functions, and performs low level operating system functions, including:
  - thread scheduling;
  - interrupt and exception dispatching; and
  - multiprocessor synchronisation.
- the **executive**:
  - performs higher level operating system functions; and
  - can call kernel functions.

All parts of the core run in kernel mode.

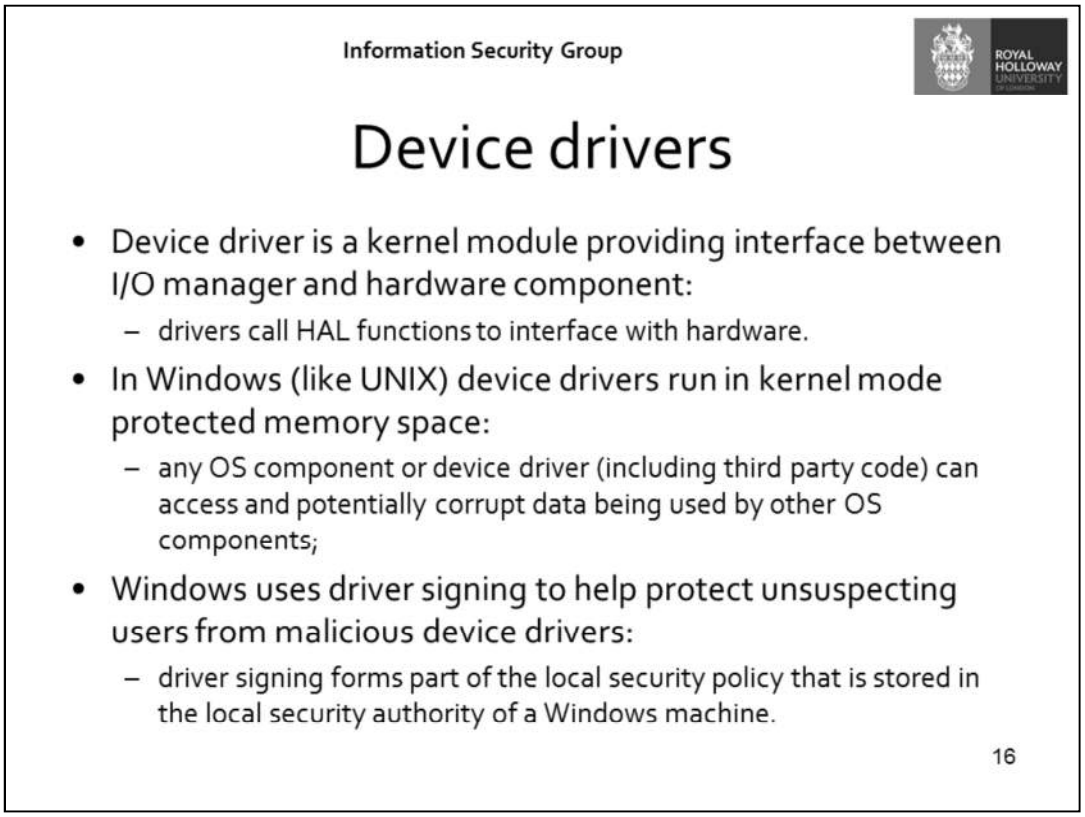



The Windows executive provides generic operating system functions, including:

- creation and deletion of processes and threads;
- memory management;
- input/output (I/O);
- interprocess communication; and
- security.

It executes in kernel mode.

The API it provides is not documented and is not accessed directly by applications – instead applications call subsystem APIs (running in user mode) which themselves interact with the executive.

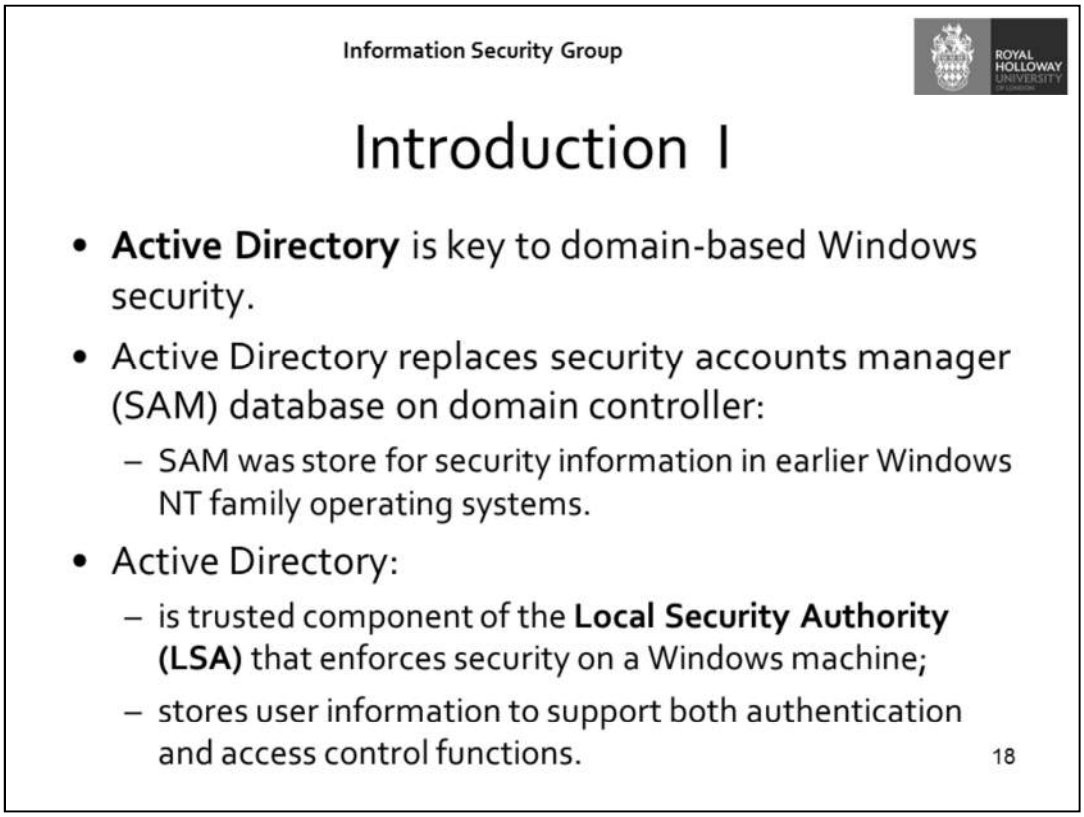



A device driver is a kernel module that act as an interface between the I/O manager and a specific hardware component. Drivers call HAL functions to interface with the hardware.

In Windows (as in UNIX), device driver code uses the kernel mode protected memory space. As a result any OS component or device driver can potentially corrupt data being used by other OS components. In particular third party device drivers run in kernel mode and have access to all operating system data.

Windows uses driver signing to help protect unsuspecting users from malicious device drivers; driver signing forms part of the local security policy that is stored in the local security authority of a Windows machine.


We next consider the core repository of information that enables a Windows domain to be managed.



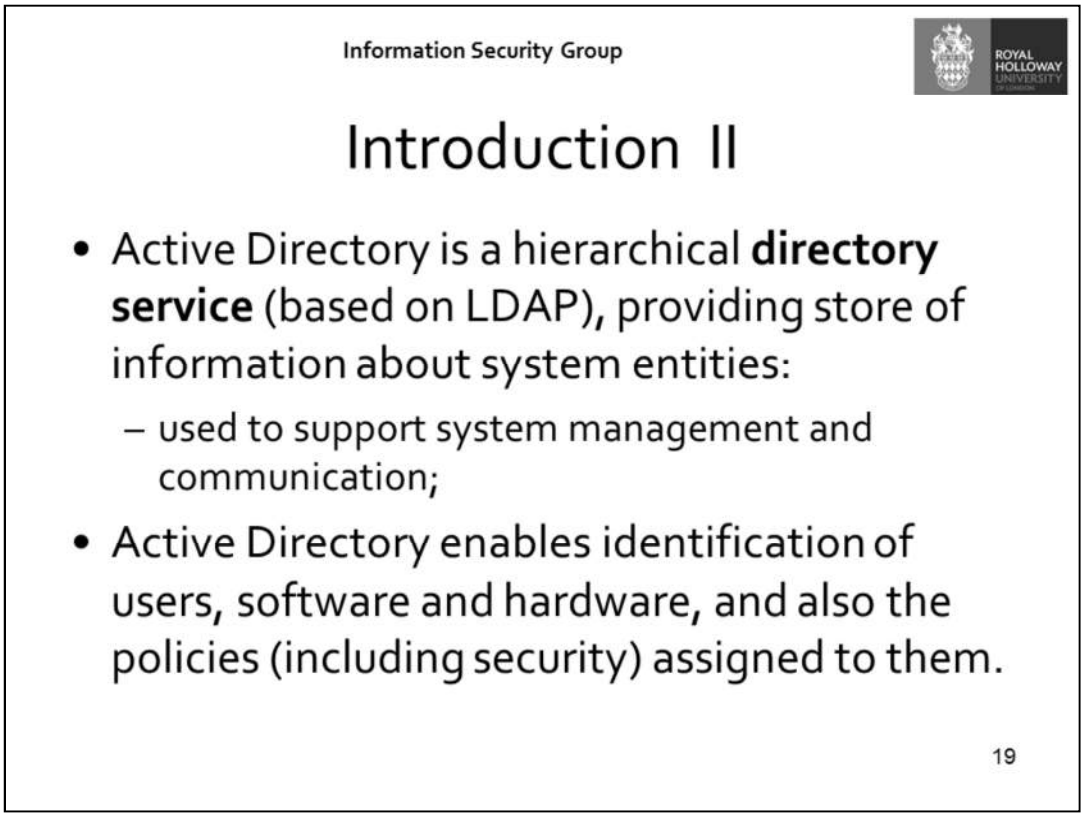

**Active Directory** is a key component in supporting Windows security functions in a managed domain.

Active Directory replaces the security accounts manager (SAM) database on a domain controller (where every domain has at least one domain controller machine). SAM was the repository for security information in earlier Windows NT family operating systems.

Active Directory is a trusted component of the Local Security Authority (LSA), and stores user information that supports both authentication and access control functions.

The **Local Security Authority Subsystem Service** (LSASS), is a Windows process that is responsible for enforcing the security policy on an individual Windows system. It verifies users logging on to a Windows computer, handles password changes, and creates access tokens (part of the access control system – see later slides). It also writes to the Windows Security Log. Forcible termination of lsass.exe will result in the Welcome screen losing its accounts, prompting a restart of the machine.

Depending on whether the Windows system is part of a domain or not, either the LSA on the domain controller or the local machine will verify user logins.

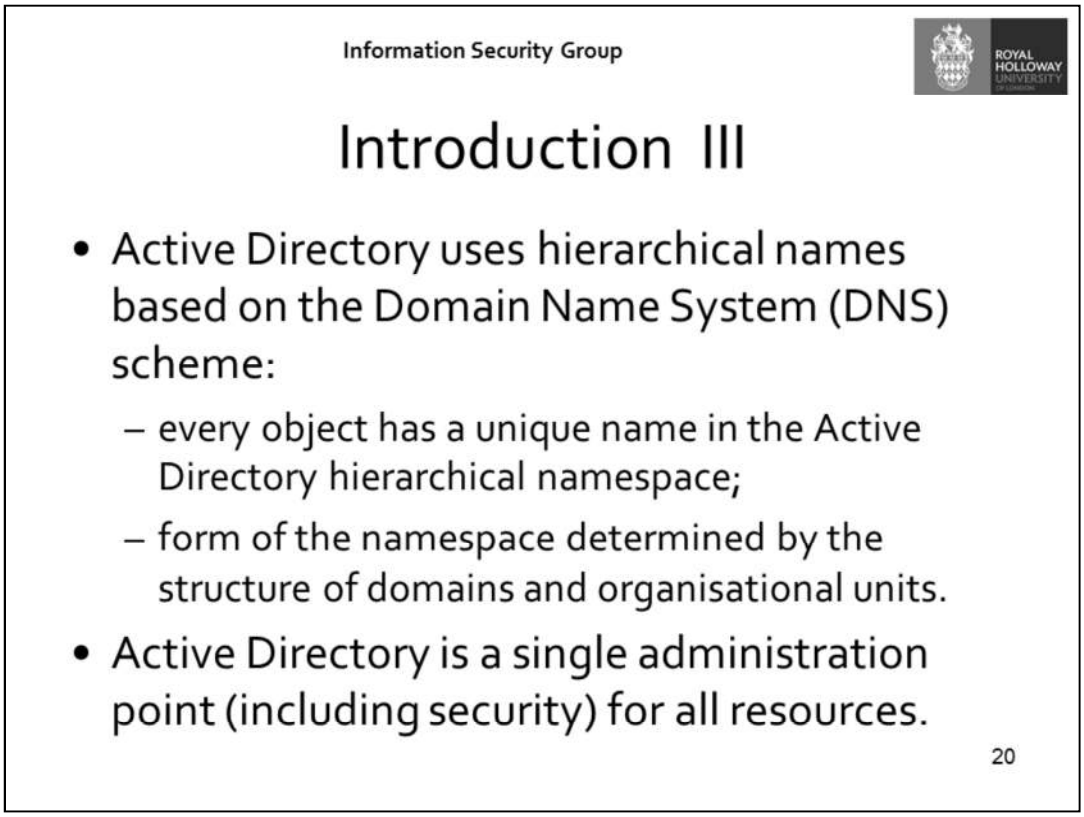



Active Directory is a hierarchical **directory service**. That is, it is a structured repository of information about entities within an information system.

- it is used to support Windows system management and communication;

Active Directory, as a networked directory service, is used to support identification of users, software and hardware, as well as the policies (including security) assigned to these entities.

The X.500 directory service, as standardised by the ITU (and by ISO), allows heterogeneous networks to share information. The LDAP (lightweight directory access protocol) is based on X.500, and was developed by the IETF. Active Directory is based on LDAP versions 2 and 3.




Active Directory uses the Internet Domain Name System (DNS) concept of a namespace within its directory service:

• every object in a Windows environment has a unique name in the Active Directory hierarchical namespace;

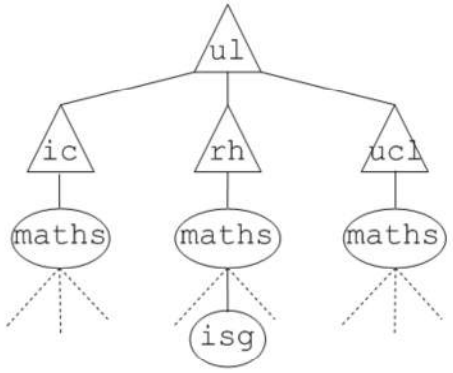
• the form of the hierarchical namespace is determined by the structure of domains and organisational units (where an organisational unit is a sort of sub-domain).

Active Directory provides a single administration point (including security administration) for all resources.

Information Security Group

# Active Directory hierarchy

- Domains connected in tree structure.
- Domain can contain organisational units (OUs), also arranged hierarchically.
- Diagram shows possible structure for a University of London (`ul`) domain:
  - sub-domains include Imperial College, Royal Holloway and University College London;
  - OUs include Maths and the ISG.




21

Multiple domains can be connected to form a tree structure.

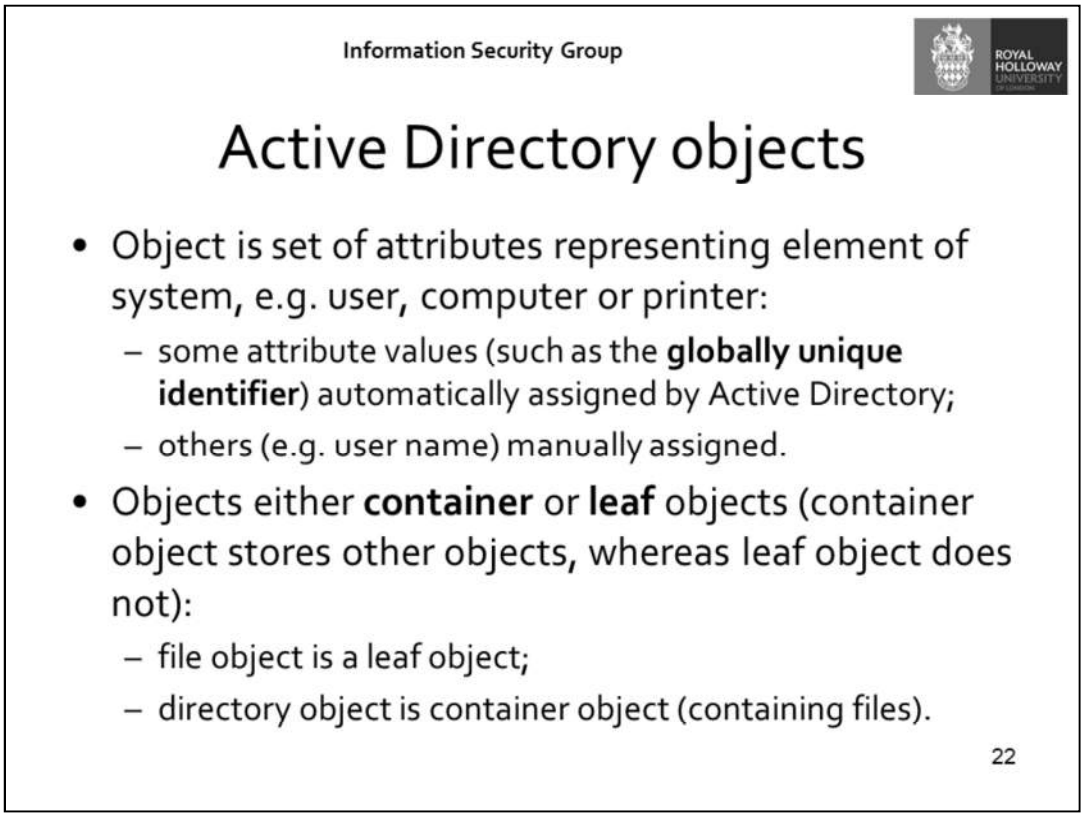
Each domain can contain organisational units (OUs), also arranged in a hierarchical structure .

The diagram illustrates part of a possible Active Directory structure for a hypothetical University of London (`ul`) domain:

• the sub-domains include Imperial College, Royal Holloway and University College London;

• the organisational units include Maths and the ISG.

[Note that this is now an out of date example, since Imperial College is no longer part of the University of London! It also incorrectly shows the ISG as part of Mathematics, whereas they are both component parts of a single school]

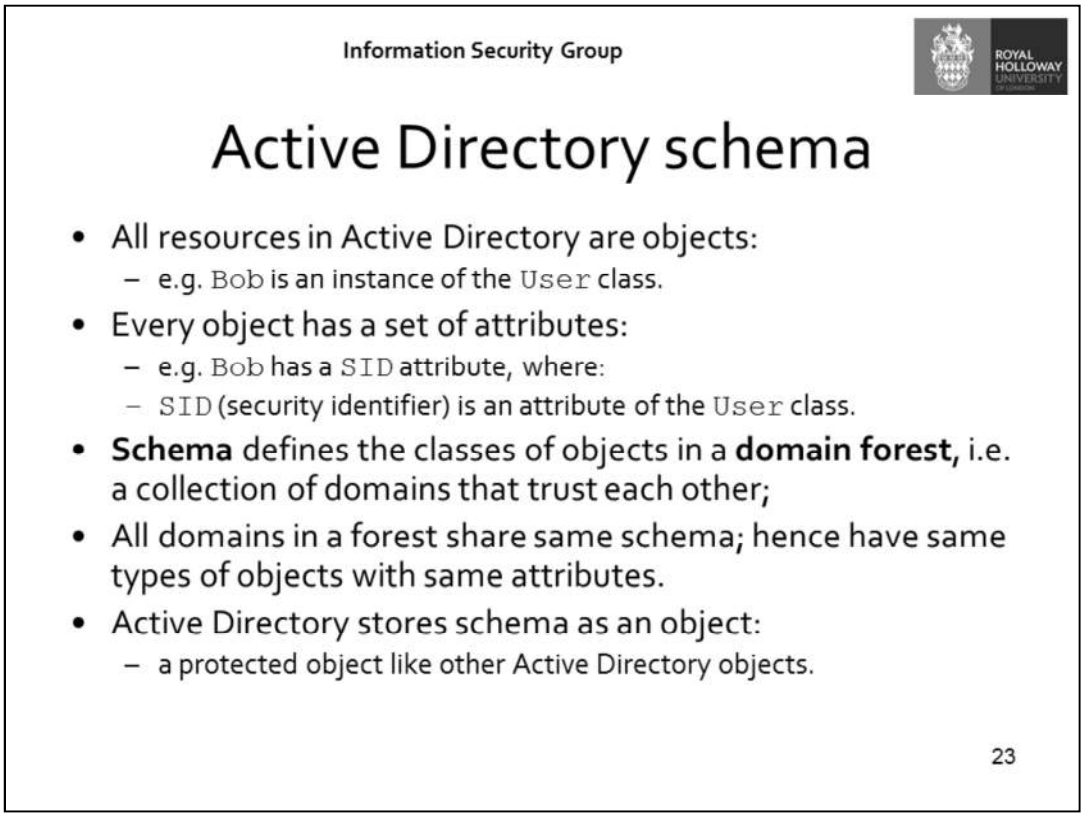



An object is a set of attributes (properties) representing something tangible within a system, e.g. a user, computer or printer:

- some attribute values (such as the **globally unique identifier**) are automatically assigned by Active Directory;
- others (such as user name) are manually assigned by an administrator.

Objects are either **container** or **leaf** objects, where container objects can store other objects, but a leaf object cannot. For example:

- a file object is a leaf object;
- a directory object is a container object (that contains files).




All resources in Active Directory are handled as objects, and each object has a **class** (indicating the kind (or type) of the object). Specifically, each object is said to be an **instance** of a particular **class**, simply meaning the object is of a particular type. This is all 'standard' object-oriented language, designed to make computer scientists sound clever by using fancy language for simple ideas (cf. the language associated with Java).

• For example, `Bob` is an instance of the `User` class.

Every object has a set of **attributes** (i.e. well-defined properties, where the available attributes depend on the class):

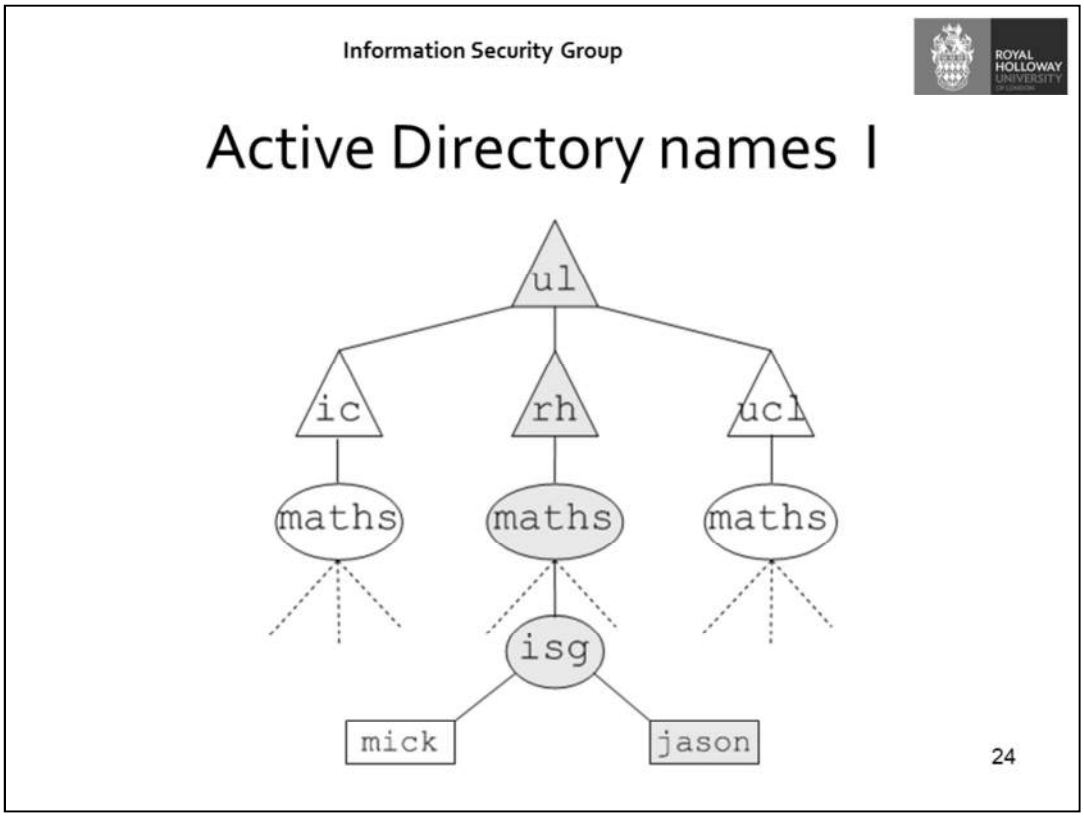
• for example, `Bob` has a `SID` attribute, where `a SID` (security identifier) is an attribute of the `User` class;

A **schema** is a description of the defined classes of objects within a **domain forest**, i.e. a collection of domains that trust each other.

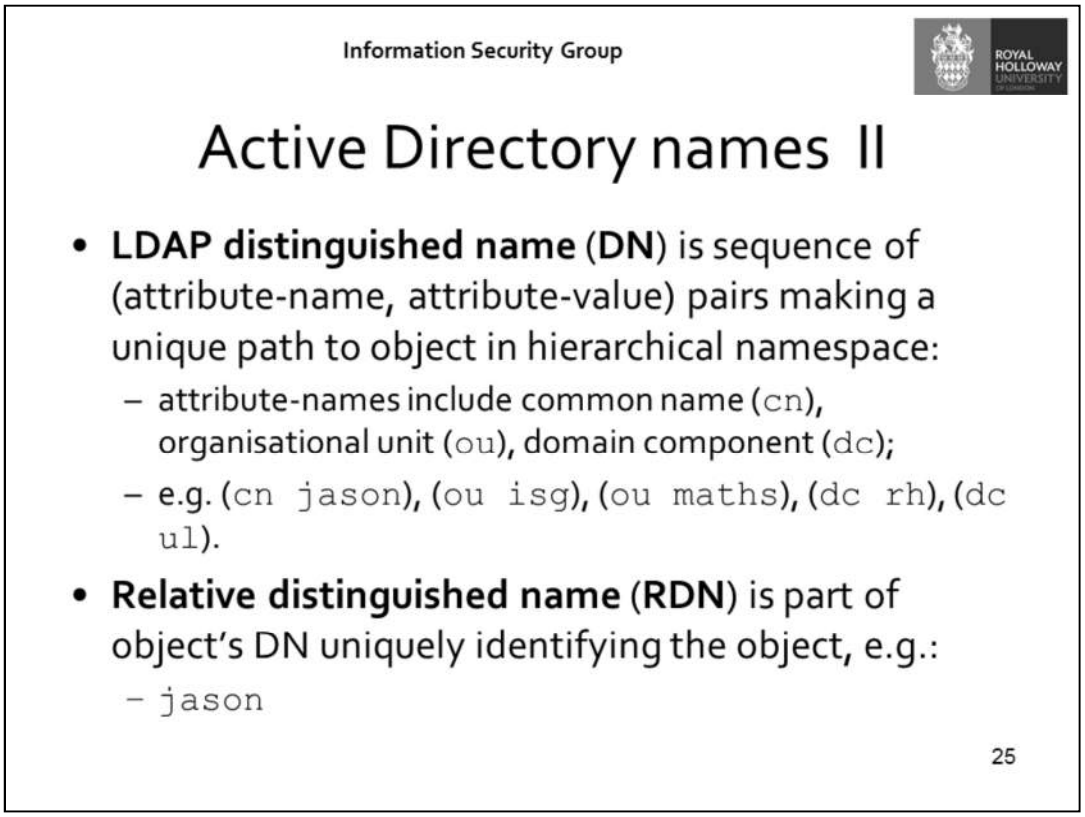
Put simply, a scheme simply lists the available classes (types of object), and the available attributes for each class.

All domains within a forest share the same schema and hence have the same types of objects with the same categories of attributes.

Active Directory stores the schema as an object. It is a protected object like other Active Directory objects.




An example of an Active Directory name is highlighted in the figure.




An **LDAP distinguished name** (**DN**) is a sequence of (attribute-name, attribute-value) pairs, separated by commas, identifying a unique path to an object in a hierarchical namespace.

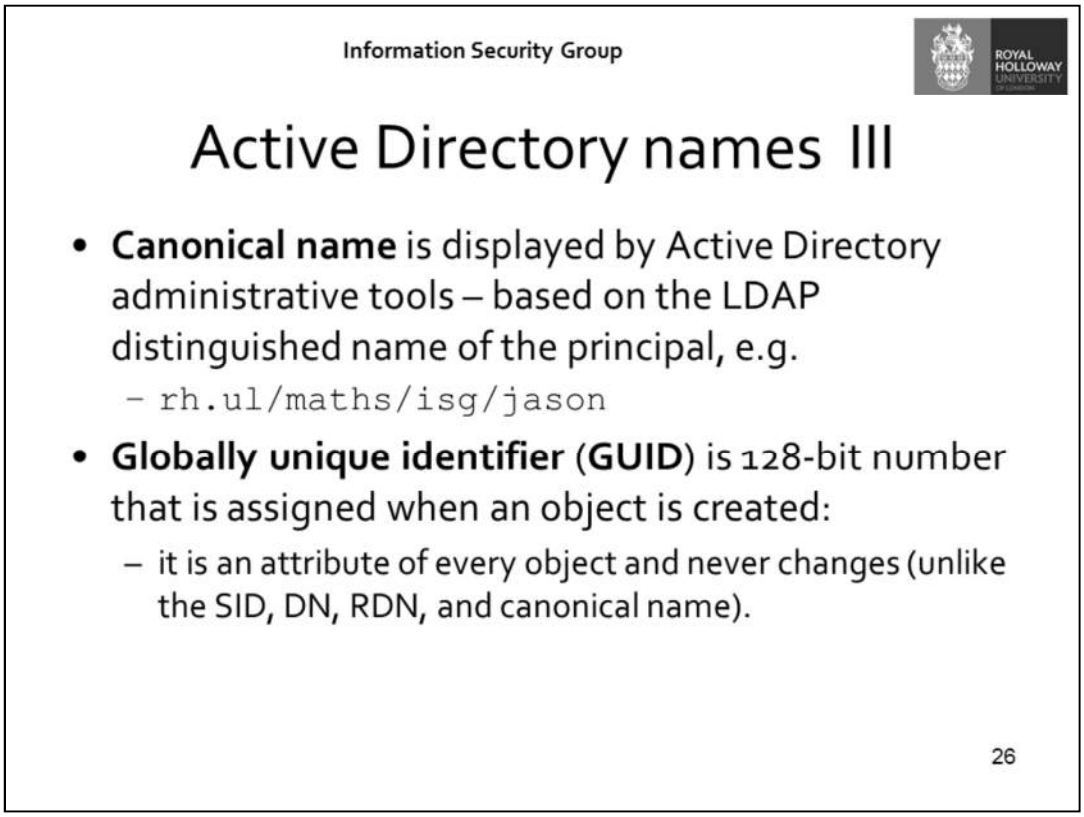
Common attribute-names include common name (`cn`), organisational unit (`ou`) and domain component (`dc`).

An example of an DN would be:

- (`cn jason`), (`ou isg`), (`ou maths`), (`dc rh`), (`dc ul`).

A **Relative distinguished name** (**RDN**) is the part of an object's DN that uniquely identifies the object, e.g.:

- `jason`




A **Canonical name** is the version of an object name that is displayed by Active Directory administrative tools. It is based on the LDAP distinguished name of the principal, e.g.

- `rh.ul/maths/isg/jason`

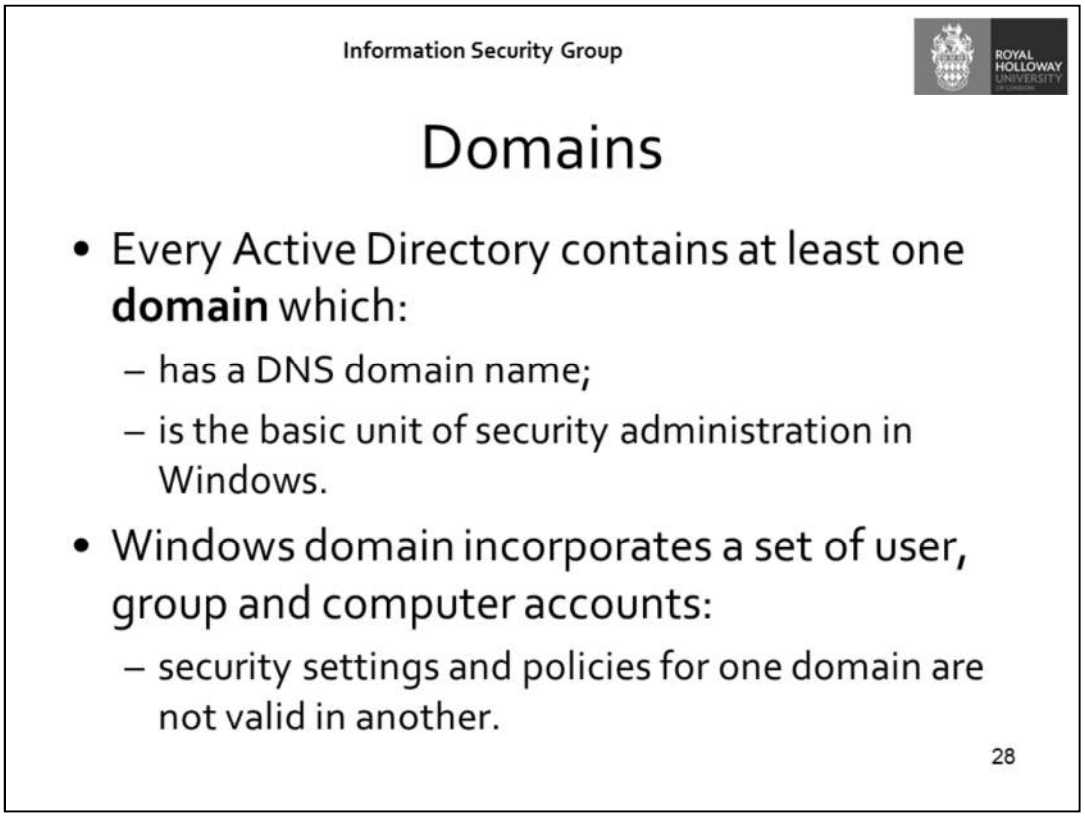
A **Globally unique identifier** (**GUID**) is a 128-bit number that is assigned to an object when that object is created. It is an attribute of every object and never changes (unlike the SID, DN, RDN and canonical name).

Information Security Group


# Agenda

- Introduction
- Active directory
- Domains
- Windows subjects and objects
- Access rights
- Authentication
- Access control
- Security management
- Resources

27

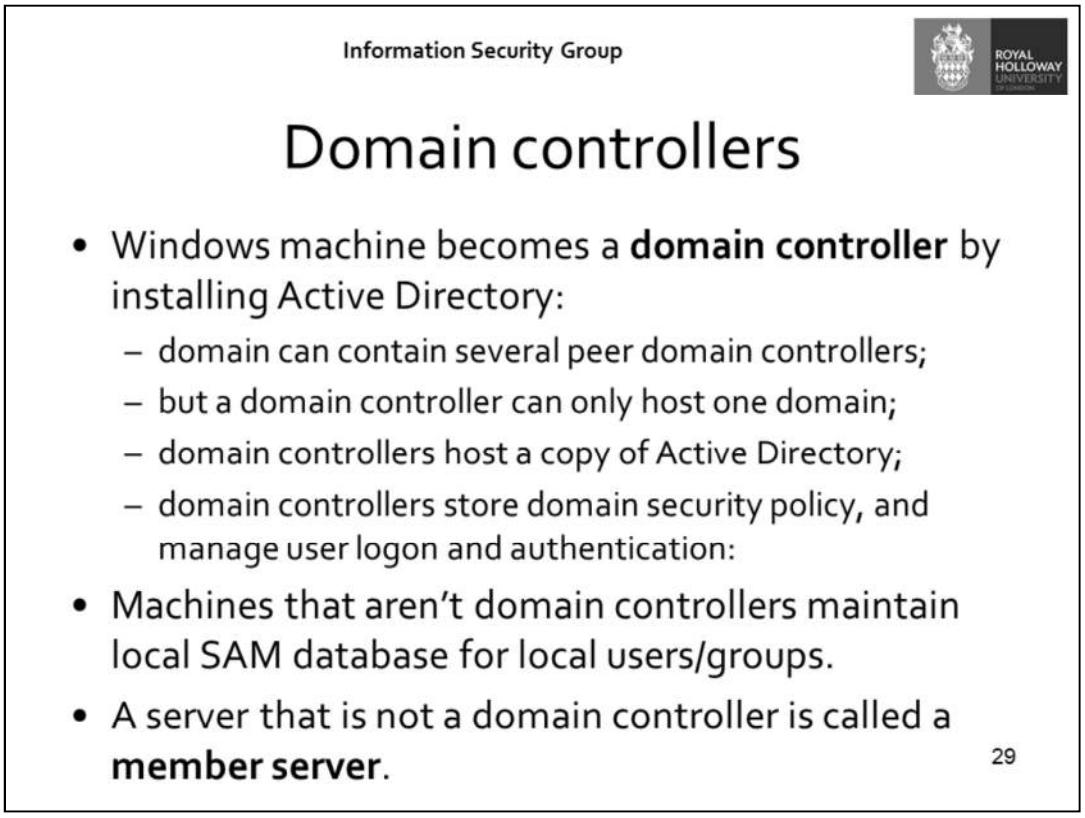



Every instance of Active Directory contains at least one **domain**:

- it is identified by a DNS (domain name service) domain name;
- it is the basic unit of security administration within Windows.

A Windows domain incorporates a set of user, group and computer accounts:

- the security settings and policies for one domain are not valid in another.

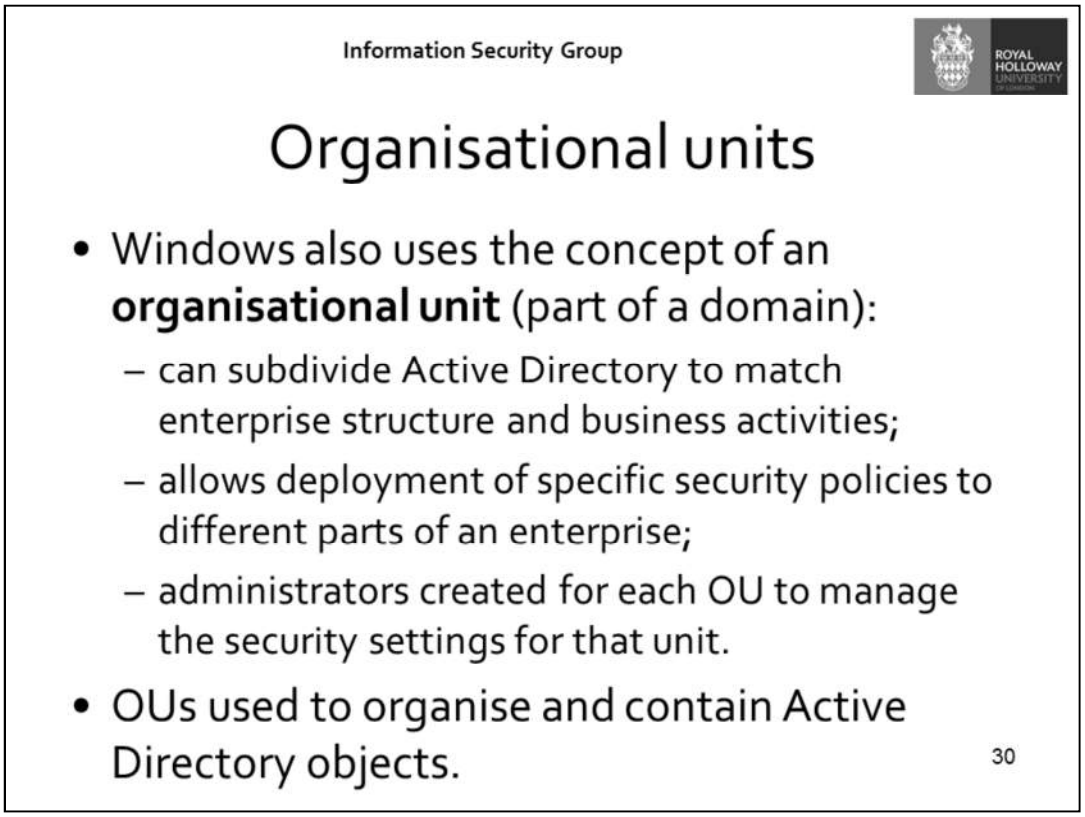



Any Windows server machine can become a **domain controller** by installing Active Directory:

- a domain can contain several domain controllers;
- however, a domain controller can only host a single domain;
- all domain controllers are peers and they each host a copy of Active Directory;
- domain controllers store the domain's security policy and manage user-domain interactions such as user logon and authentication:

Machines that are not domain controllers retain a local SAM database for locally defined users and groups.

Any server within a domain that is not a domain controller is called a **member server**.



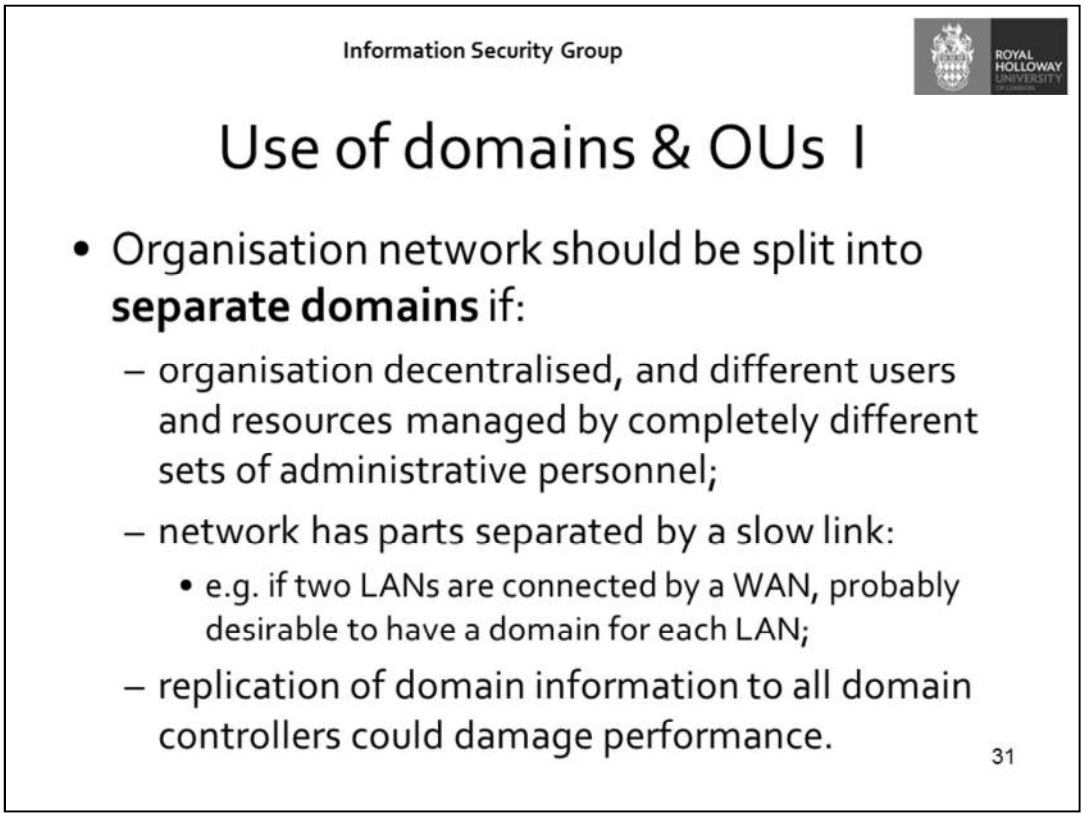

Windows also uses the concept of an **organisational unit** (OU) which forms part of a domain:

• OUs can be used to sub-divide the scope of an instance of Active Directory in a way that corresponds to an enterprise's internal structure and business activities;

• OUs can simplify security management by allowing fine-grained deployment of security policies to different parts of the enterprise;

• administrators are created for each OU to manage the security settings for that unit.

OUs are used to organise and contain Active Directory objects.



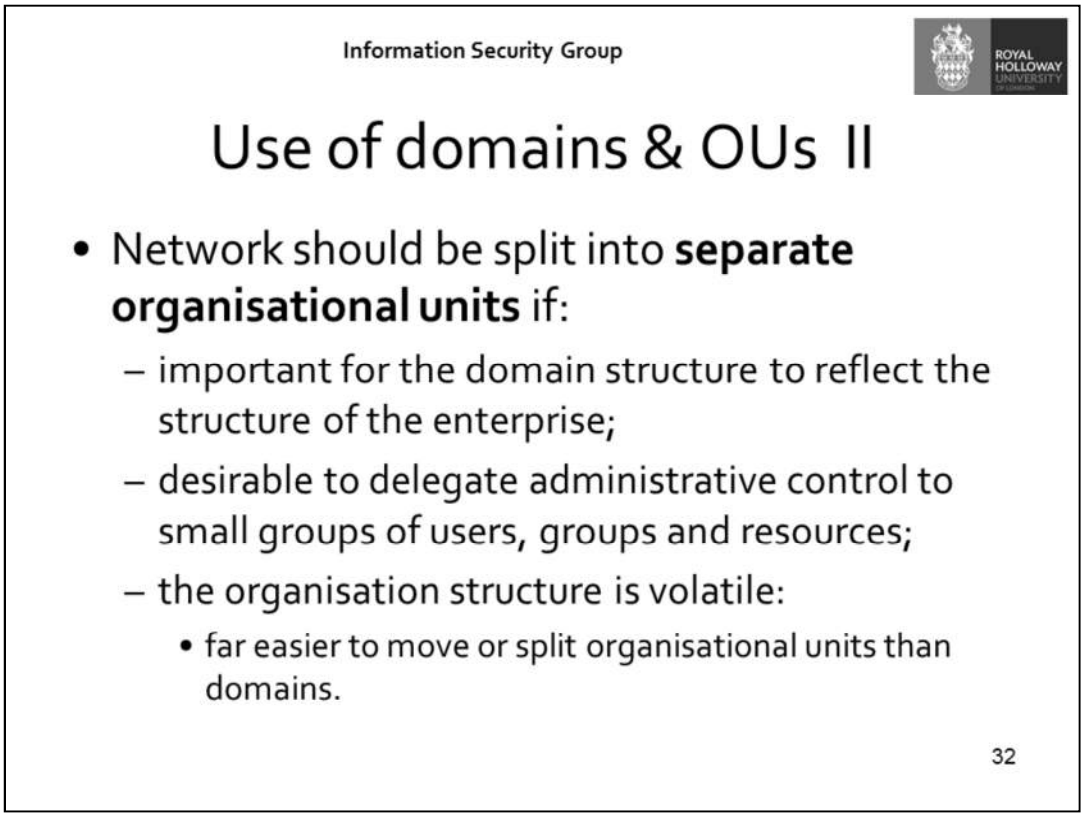

The network of an organisation should be split into separate domains in the following circumstances:

• an organisation is decentralised, and different users and resources are managed by completely different sets of administrative personnel;

• the network contains two (or more parts) separated by a slow link:

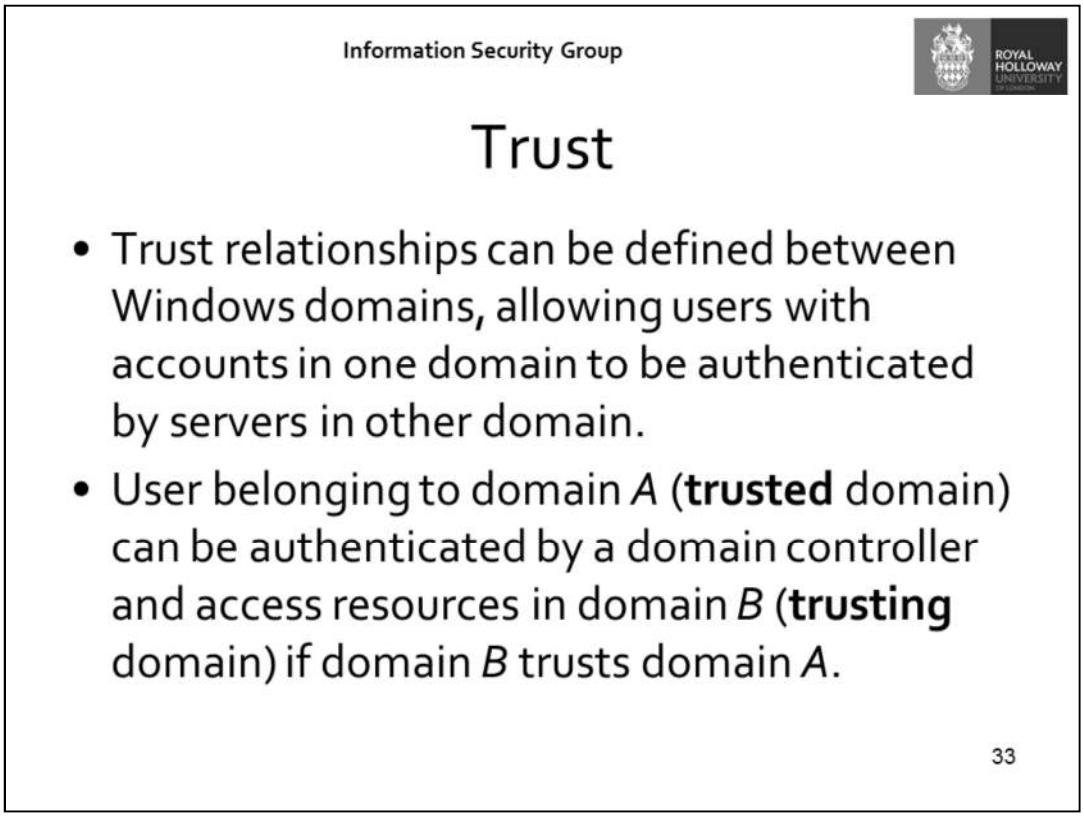
  • e.g. if two LANs are connected by a WAN it is probably desirable to have one domain for each LAN;

• a single domain would mean that the replication of domain information to all domain controllers could have a detrimental effect on performance.



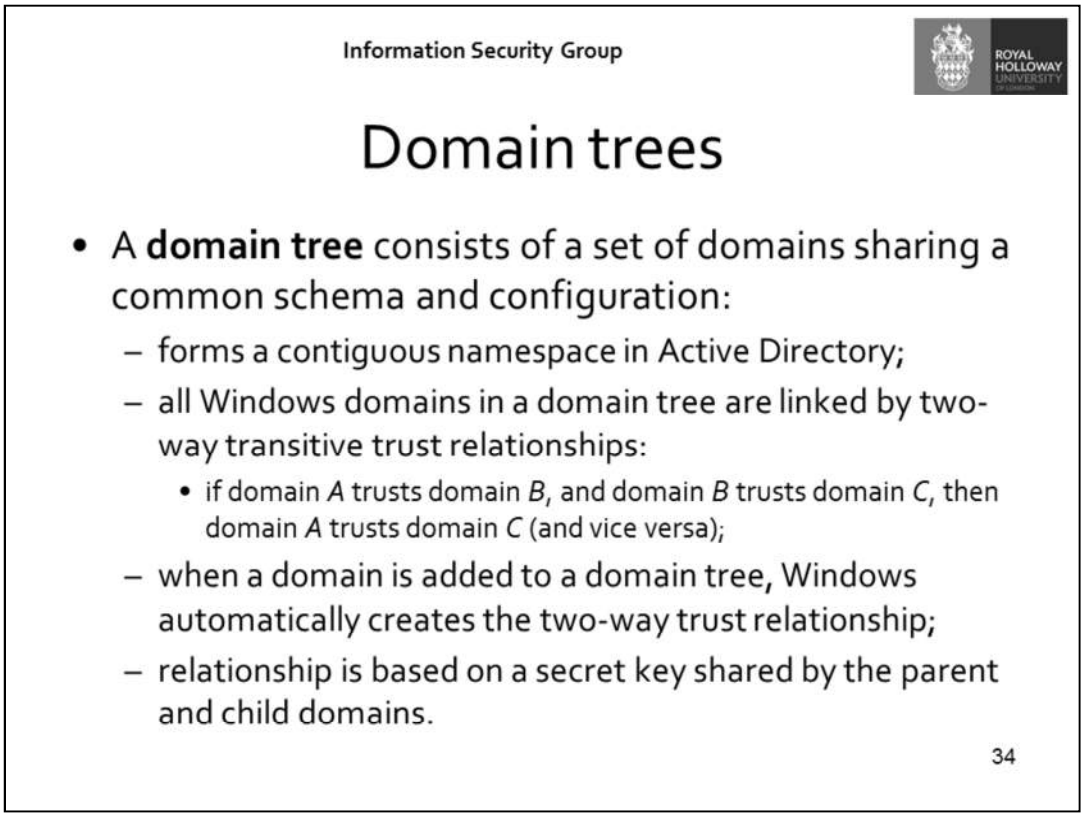

A network should be split into separate organisational units in the following circumstances:

- when it is important for the organisation of Windows to reflect the structure of the enterprise;
- when it is desirable to delegate administrative control to small groups of users, groups and resources;
- when the organisational structure is volatile:
    - it is far easier to move or split organisational units than it is for domains.



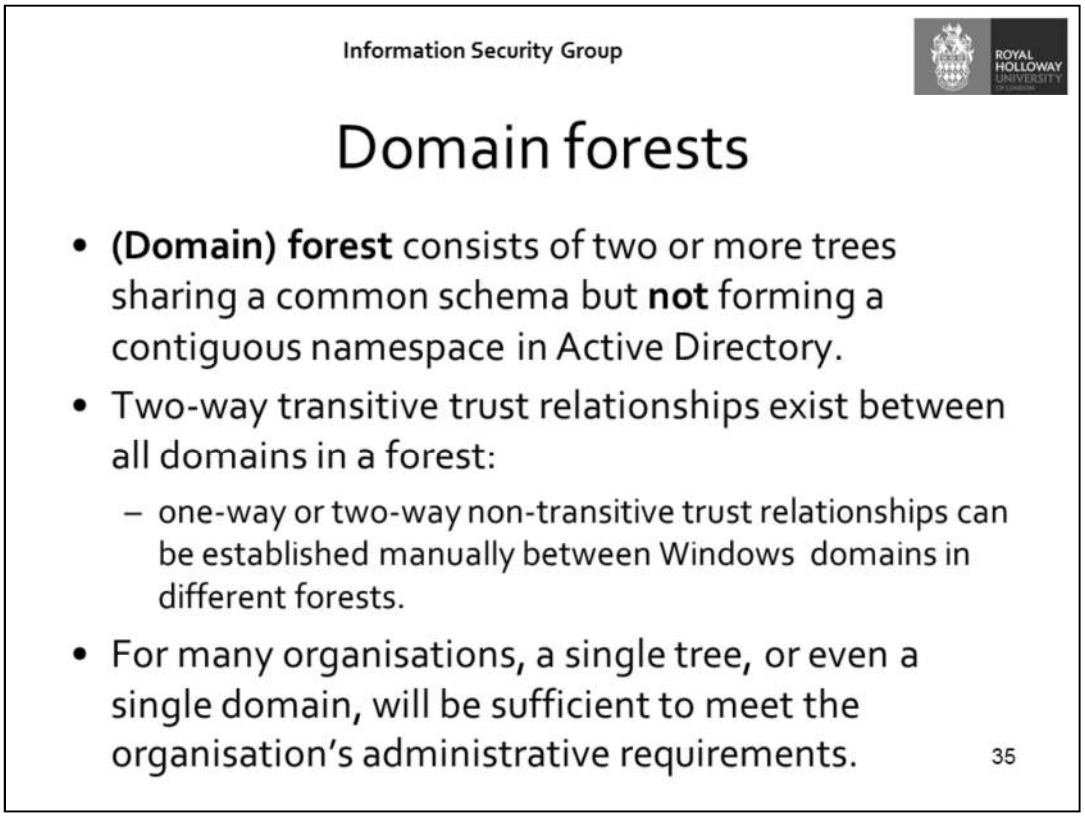

Trust relationships can be defined between Windows domains, allowing users with accounts in one domain to be authenticated by servers in another domain:

A user belonging to domain *A* (the **trusted** domain) can be authenticated by a domain controller and access resources in domain *B* (the **trusting** domain) if domain *B* trusts domain *A*.




Closely linked domains can be linked together to form a domain tree. A **domain tree** consists of a set of domains sharing a common schema and configuration:

- it forms a contiguous namespace within Active Directory;
- all Windows domains within a domain tree are linked by two-way transitive trust relationships:
    - if domain *A* trusts domain *B*, and domain *B* trusts domain *C*, then domain *A* trusts domain *C* (and vice versa);
- when a domain is added to a domain tree, Windows automatically creates the two-way trust relationship;
- the trust relationship is based on a secret key shared by the parent and child domains.




A **(domain) forest** consists of two or more trees that share a common schema but do not form a contiguous namespace in Active Directory.

Two-way transitive trust relationships exist between all domains in a forest:

• one-way or two-way non-transitive trust relationships can be established manually between Windows domains in different forests.

For many organisations, a single tree, or even a single domain, will be sufficient to meet the organisation's administrative requirements. However, a domain forest may well be necessary for large and/or de-centralised organisations.

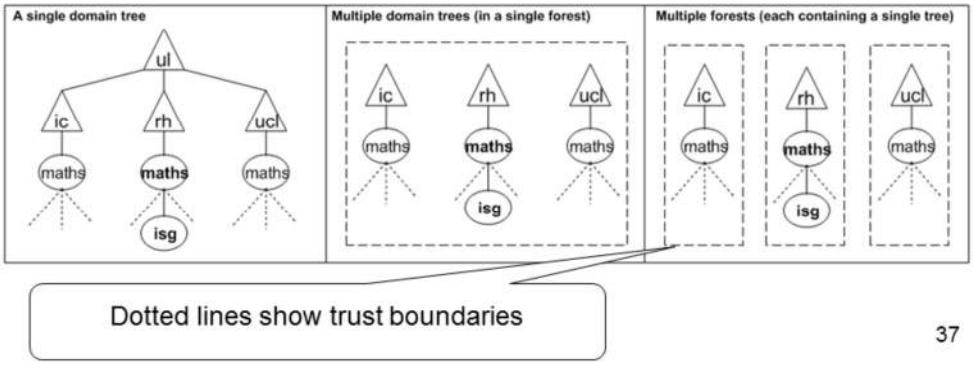

It may be necessary to isolate certain domains from other domains. Why? Well:

- administrators of any domain in forest can modify configuration information that is replicated throughout the forest:
  - thus it is important that only **managed domains**, whose administrators are trustworthy, are allowed to join a forest.
- there thus may be a need to create multiple forests:
  - users from unmanaged domains can use resources in managed domains within a forest by explicitly creating one-way trust relationships.

Information Security Group

# Domain forests – examples

- Unlikely that college sub-domains trust each other.
- Best solution is to have multiple forests:
  - even in the Royal Holloway domain different forests may be needed.



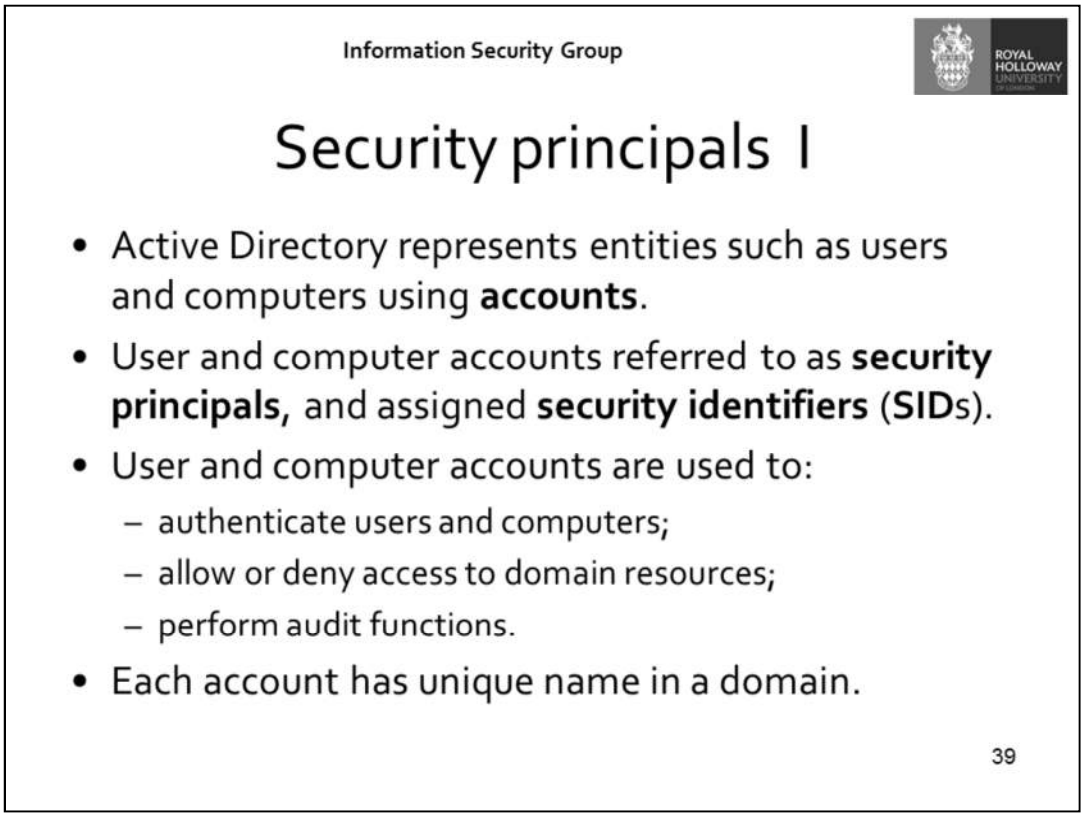

37

Returning to our university example, it is unlikely that the college sub-domains will trust each other.

The best solution is to have multiple forests.

Indeed, even within the Royal Holloway domain, different forests may be required.


We next consider the notions of subject and object, fundamental to how Windows provides access control.



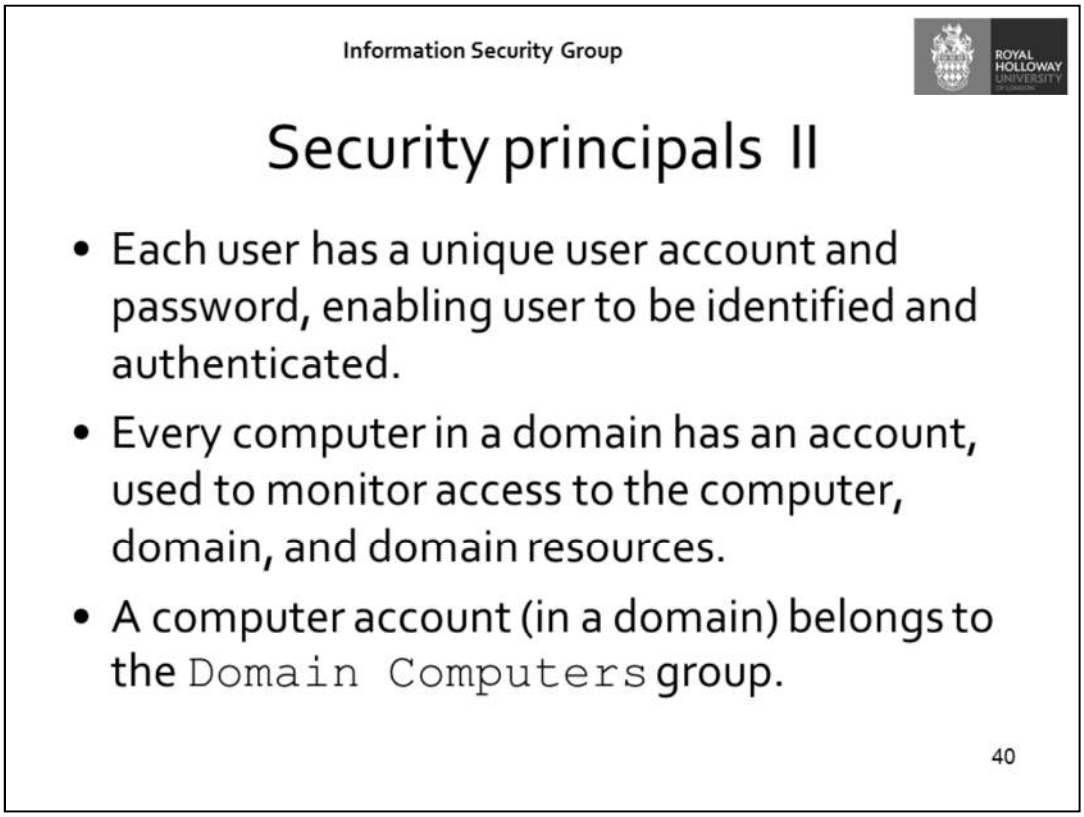

Active Directory represents physical entities, such as users and computers, using **accounts**.

User and computer accounts are referred to as **security principals** and are assigned **security identifiers** (**SID**s), discussed later.

User and computer accounts are used to:

- authenticate users and computers;
- allow or deny access to domain resources;
- perform audit functions.

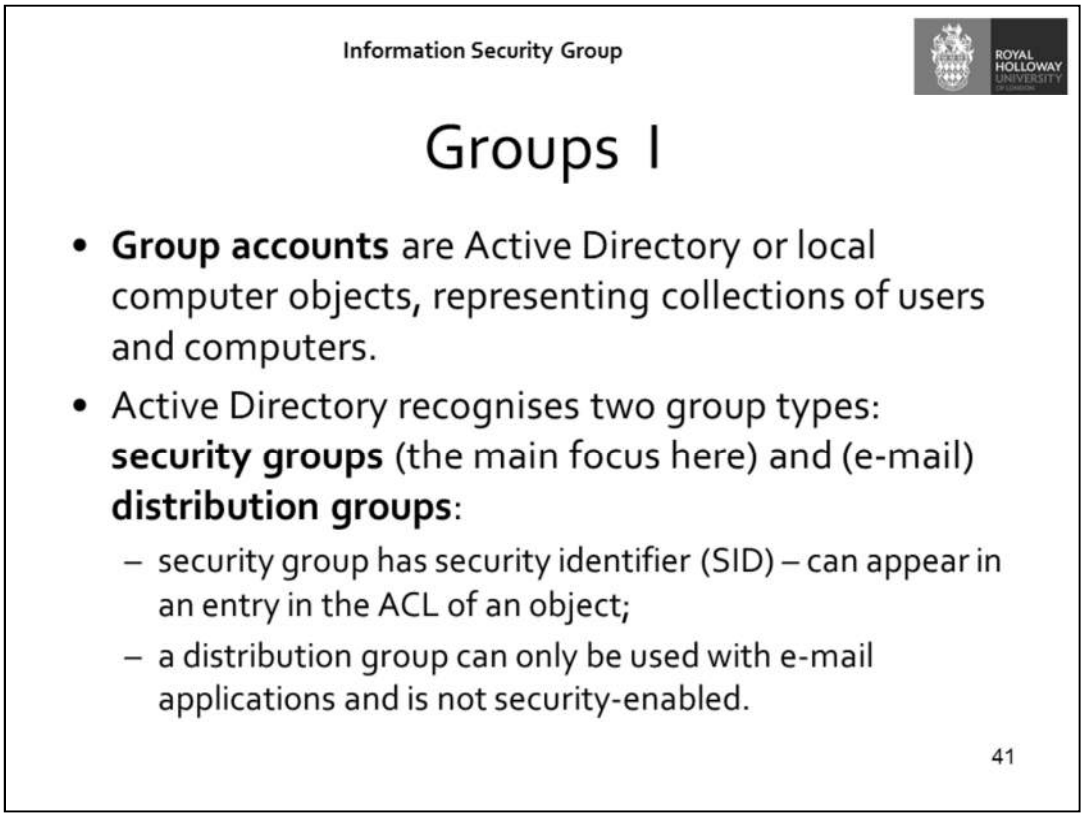
Each account has a name that is unique within a domain.




Each user has a unique user account and password, enabling the user to be identified and authenticated.

Every computer in a domain has an account, used to monitor access to the computer, the domain and domain resources.

A computer account (in a domain) belongs to the `Domain Computers` group.

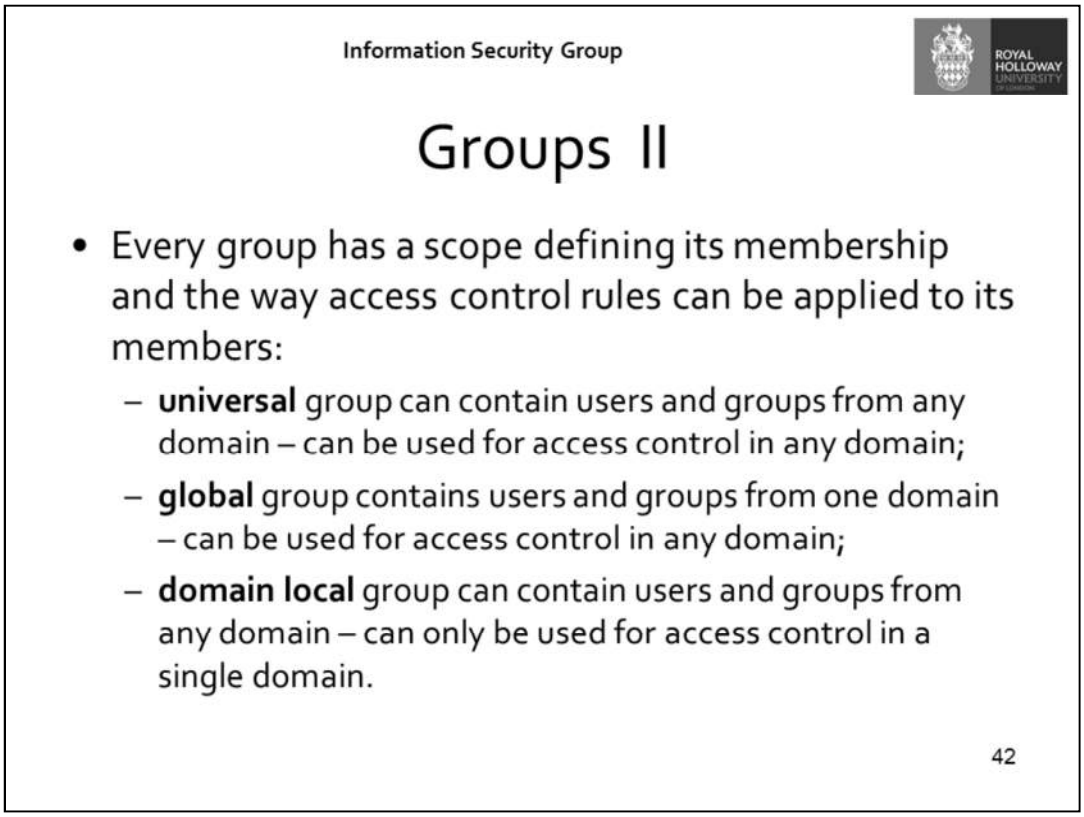



Accounts can also represent collections of entities. **Group accounts** are Active Directory or local computer objects that typically represent collections of users or computers.

Active Directory recognises two group types: **security groups** (the main focus of our interest in this course) and (e-mail) **distribution groups**:

• a security group has a security identifier (i.e. a SID) that can appear in an access control entry in the DACL (i.e. the Windows name for an ACL) of an object (thereby controlling access to that object by members of the group);

• a distribution group can only be used with e-mail applications and is not security-enabled. Distribution groups have only one function, namely to create e-mail distribution lists. Distribution groups are used with e-mail applications (such as Microsoft Exchange) to send e-mail to the members of the group. As with a security group, a contact can be added to a distribution group so that the contact receives e-mail sent to the group.

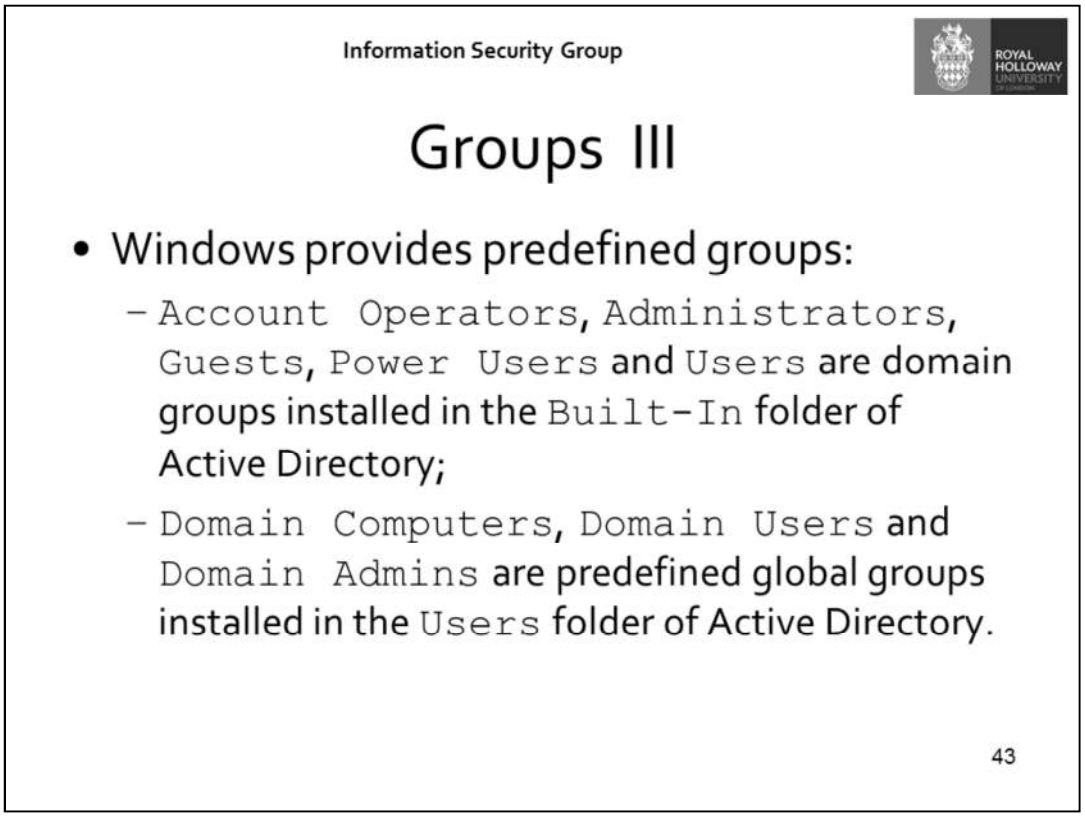



Every group has a scope which limits its membership and the way in which access control rules can be applied to its members. There are different types of group:

• a **universal** group can contain users and groups from any domain, and can be used for access control in any domain;

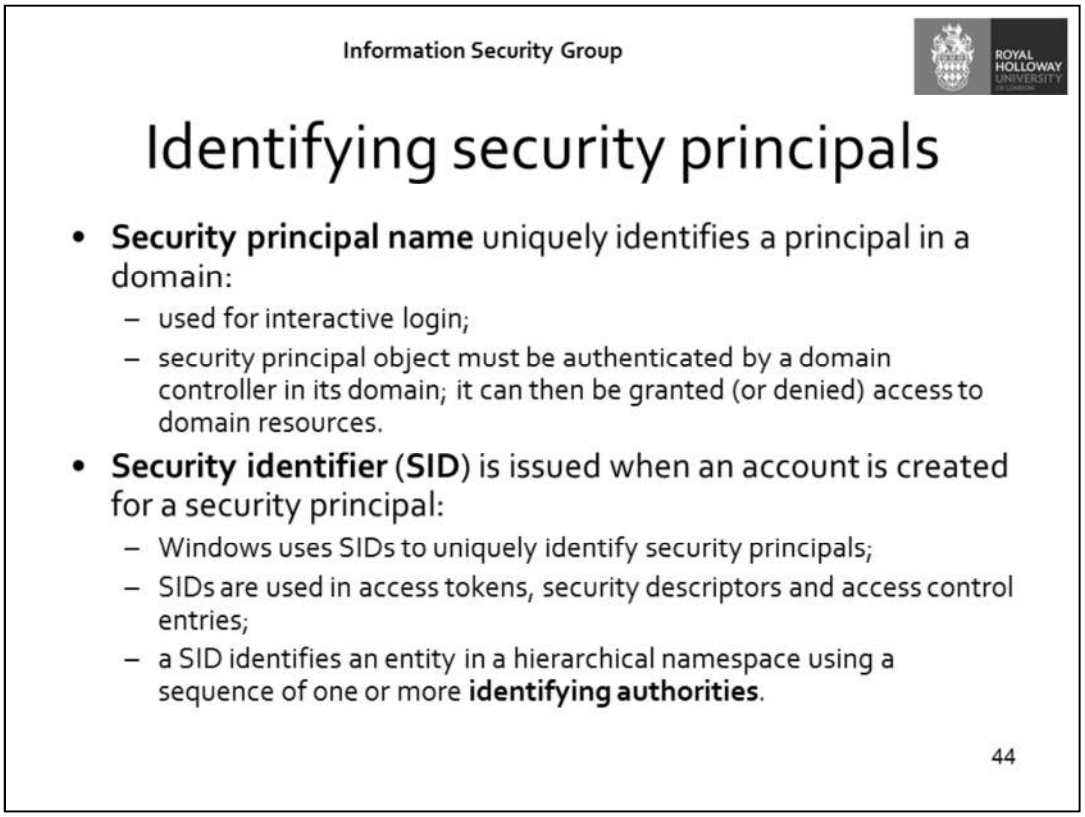
• a **global** group contains users and groups from a single domain, but can be used for access control in any domain;

• a **domain local** group can contain users and groups from any domain, and can only be used for access control within a single domain.




Windows provides certain predefined groups:

- `Account Operators`, `Administrators`, `Guests`, `Power Users` and `Users` are domain groups installed in the `Built-In` folder of Active Directory;
- `Domain Computers`, `Domain Users` and `Domain Admins` are predefined global groups installed in the `Users` folder of Active Directory.




A **security principal name** uniquely identifies a principal within a domain:

• it is used for interactive login;

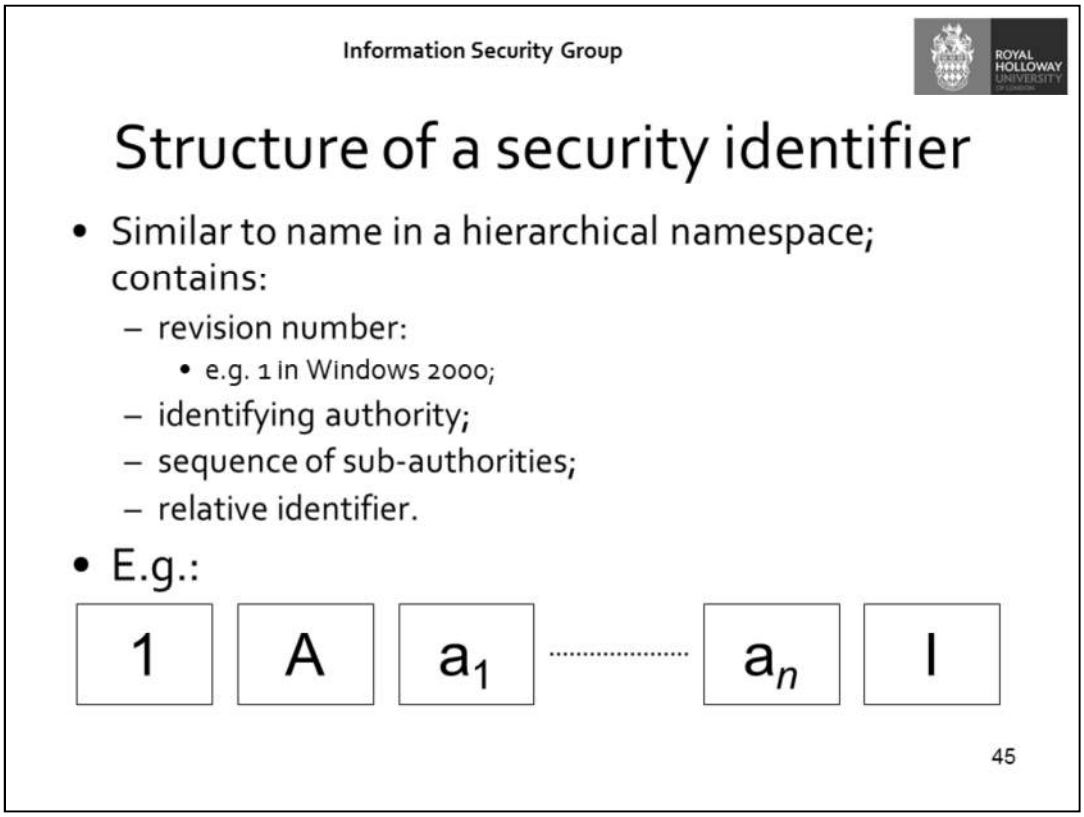
• a security principal object must be authenticated by a domain controller within its domain; it can be subsequently granted (or denied) access to domain resources.

A **security identifier** (**SID**) is issued when an account is created for a security principal:

• Windows uses SIDs to uniquely identify security principals;

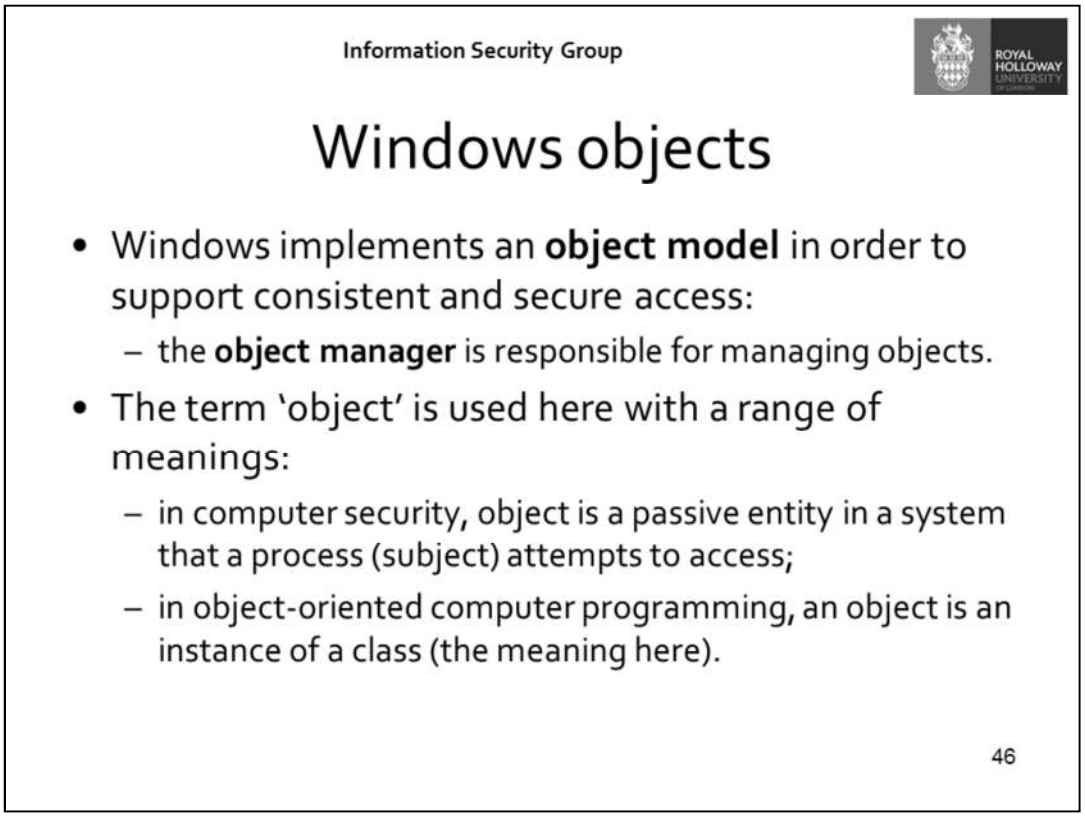
• SIDs are used in access tokens, security descriptors and access control entries;

• a SID identifies an entity within a hierarchical namespace using a sequence of one or more **identifying authorities**.




A security identifier is similar to a name in a hierarchical namespace. It consists of:

- a revision number, e.g. 1 in Windows 2000;
- an identifying authority;
- a sequence of sub-authorities;
- a relative identifier.



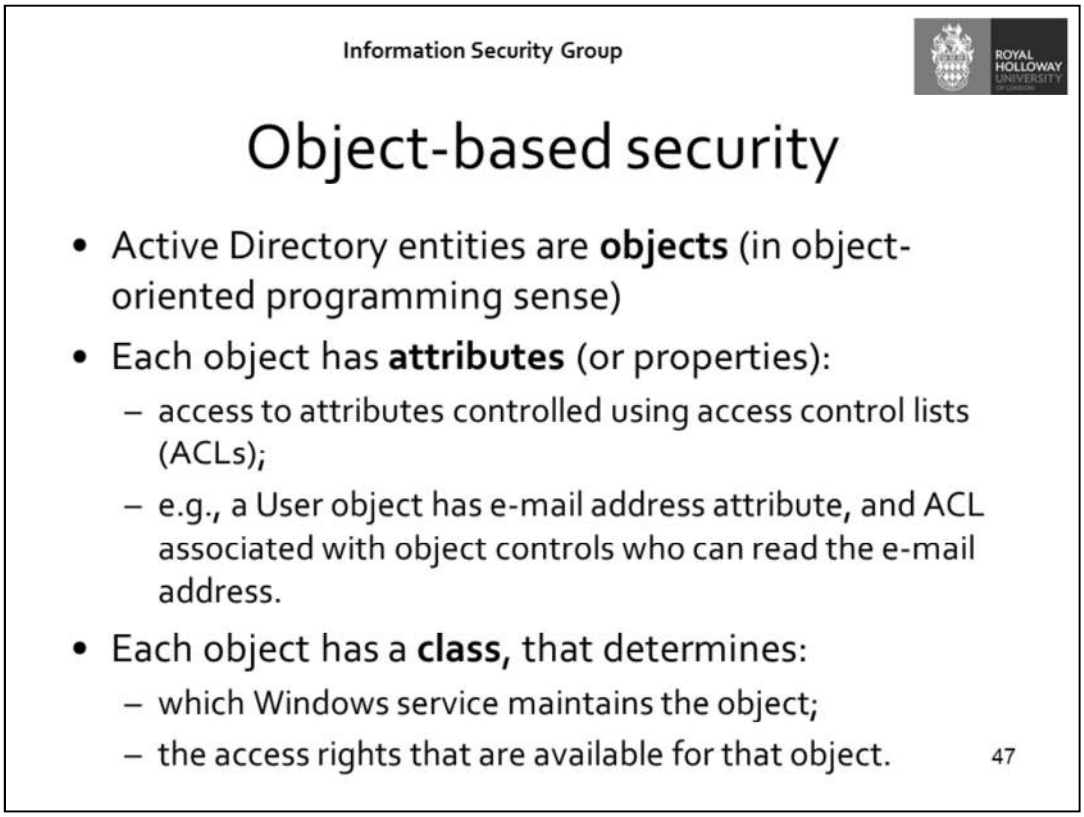

Windows implements an **object model** in order to support consistent and secure access:

• the **object manager** is responsible for managing objects.

Unfortunately, we use the term 'object' with a range of meanings:

• in computer security, an object is a passive entity within the system that a process (subject) attempts to access;

• in object-oriented computer programming, an object is an instance of a class (and this is the meaning of the term used in our description of Windows security).



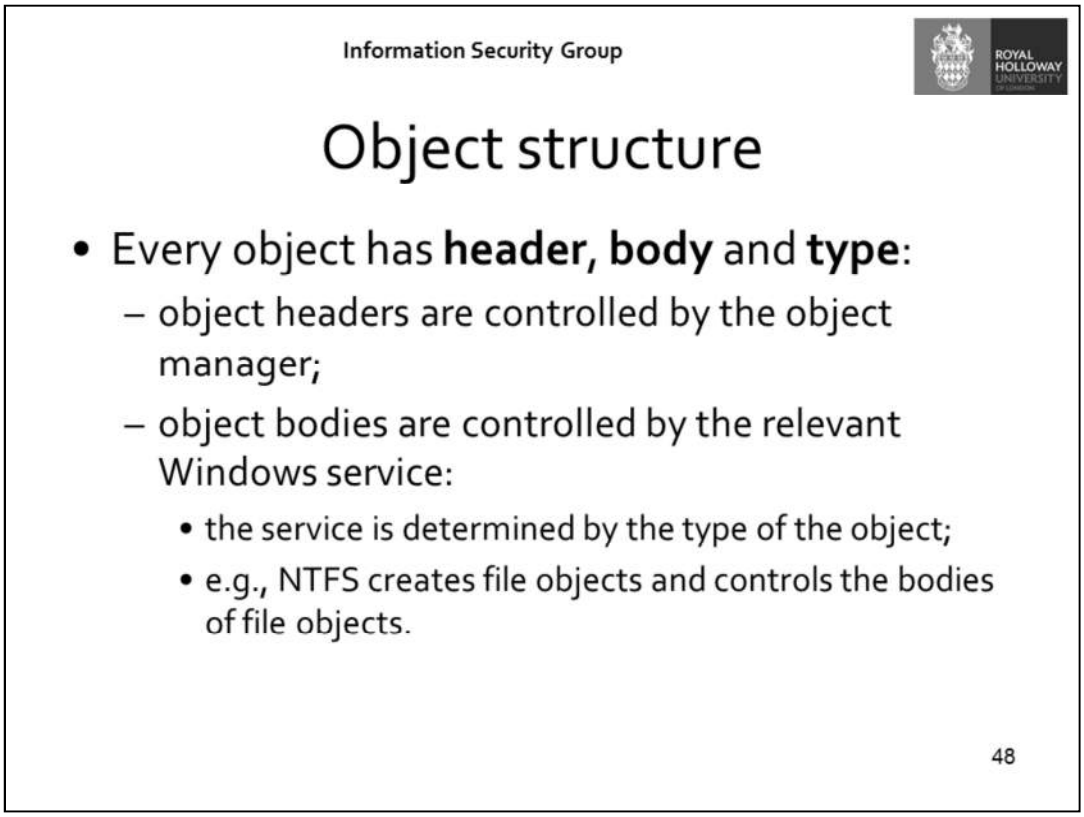

Active Directory entities are **objects** (in the object-oriented programming sense)

Each object has **attributes** (also referred to as **properties**):

- access to attributes can be controlled using access control lists (ACLs);
- e.g., a User object (i.e. an object of the user class) has an e-mail address attribute, and an ACL associated with a User object can control who can read the e-mail address of a user.

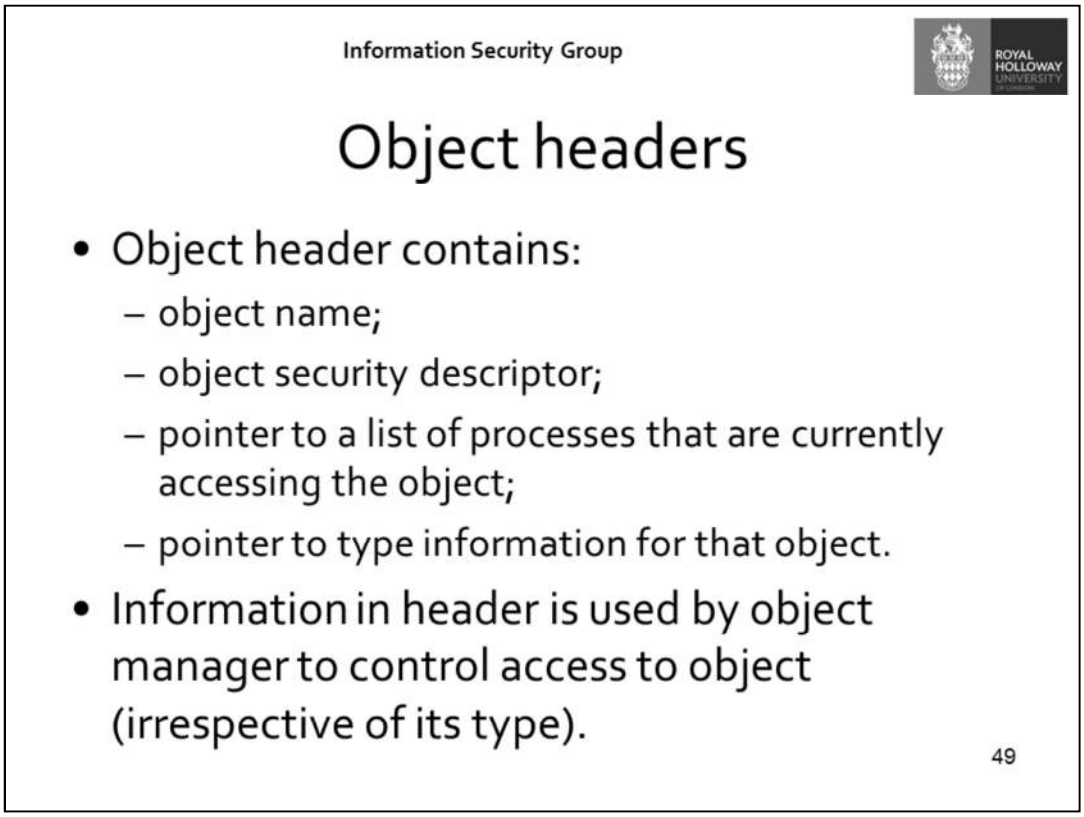
Each object has a **type** (or **class**), that determines:

- which Windows service is responsible for maintaining the object;
- the access rights that are available for that object.




Every object has a **header**, a **body** and a **type**:

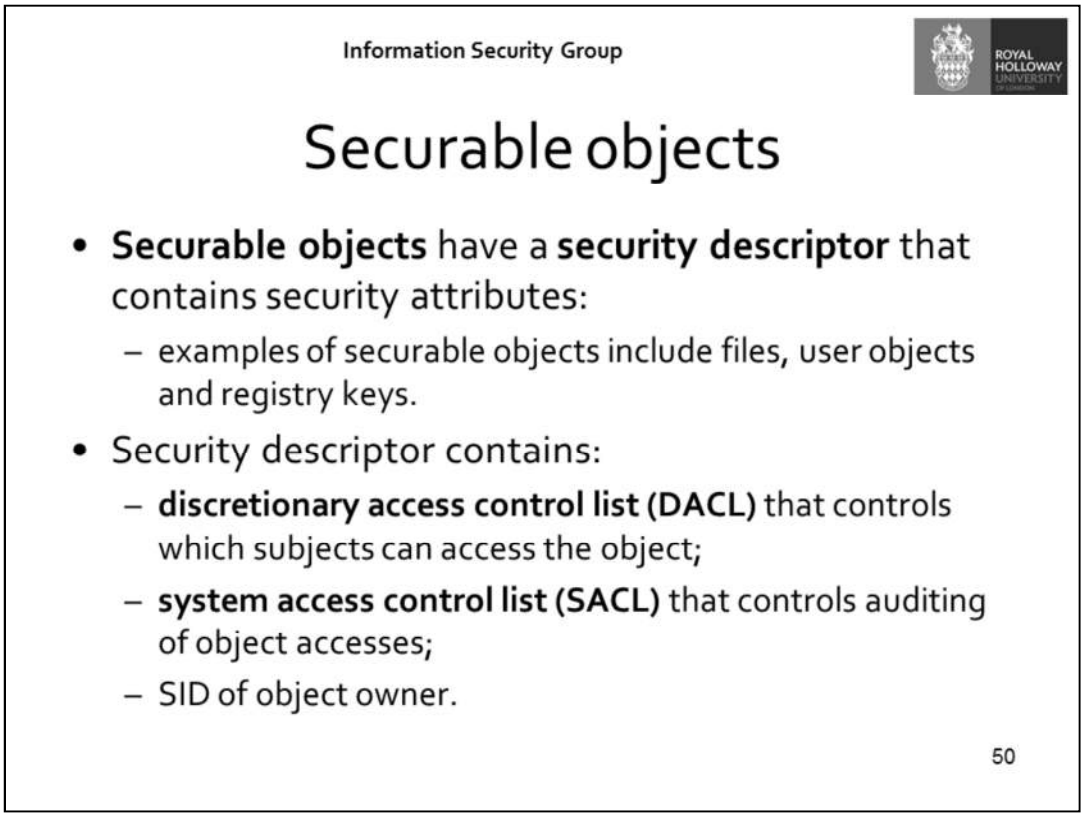
- object headers are controlled by the object manager;
- object bodies are controlled by the relevant Windows service:
  - the particular service responsible for looking after an object body is determined by the type of the object;
  - for example, NTFS (the file system) creates file objects and controls the bodies of file objects.




An object header contains (amongst other things):

- the object name;
- the object security descriptor (see next slide);
- a pointer to a list of processes that are currently accessing the object;
- a pointer to type information for that object.

Information in the header is used by the object manager to control access to the object (irrespective of its type). That is, the header is used to support the access control function.



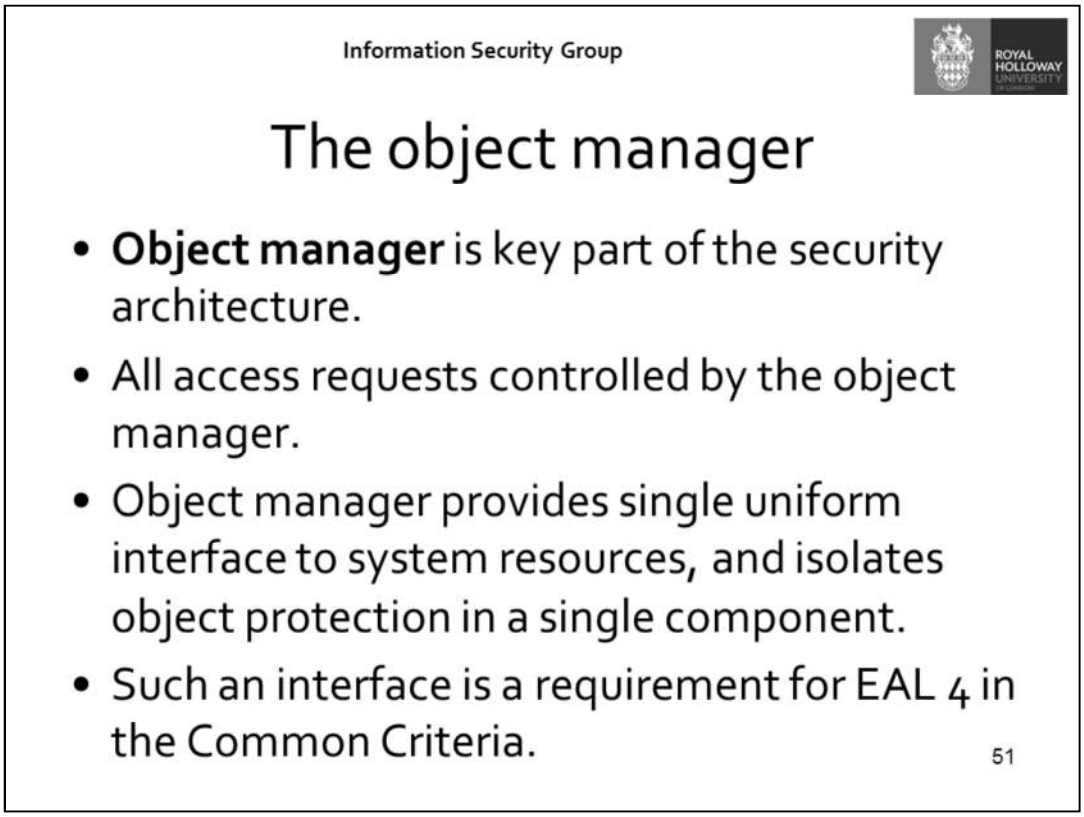

**Securable objects** have a **security descriptor** that contains security attributes. Examples of securable objects include files, user objects and registry keys.

The security descriptor contains:

• a **discretionary access control list (DACL)** that controls which users and groups are able to access the object;

• a **system access control list (SACL)** that controls which entities are permitted to audit object accesses;

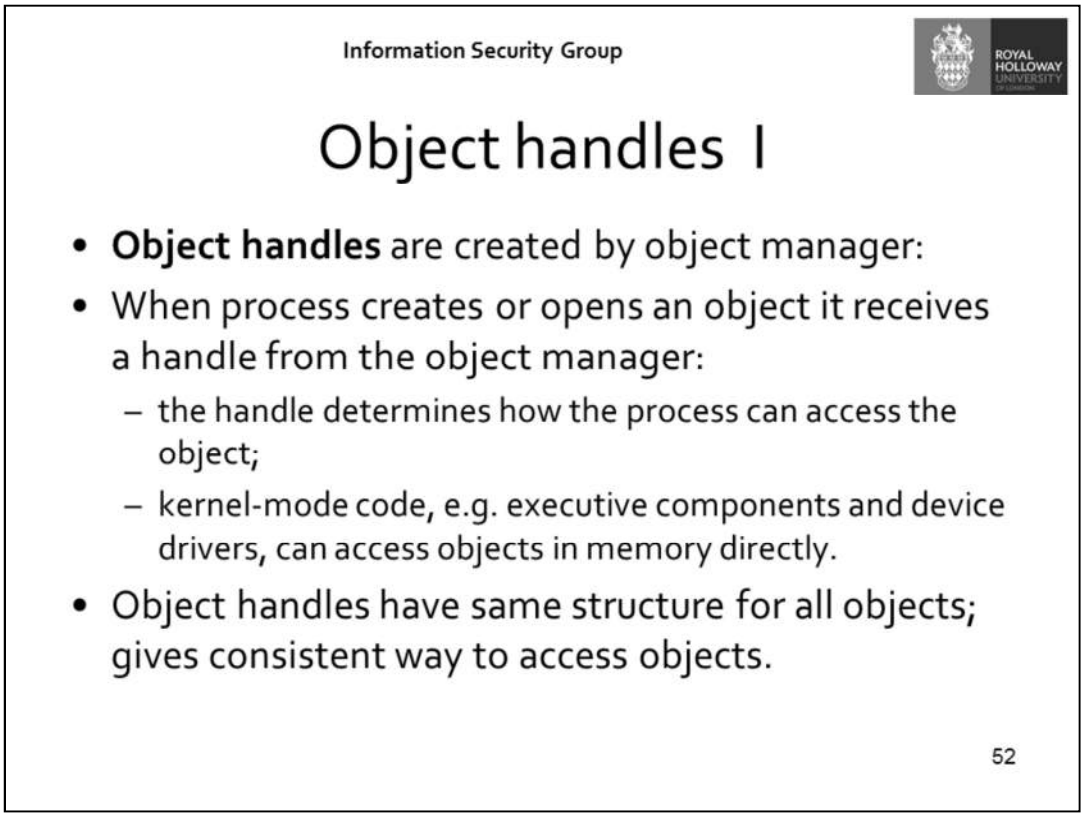
• the SID of the object owner.




The **object manager** is a vital part of the Windows security architecture. All access requests are controlled by the object manager in a uniform way.

The object manager provides a single uniform interface to system resources, and hence isolates object protection in a single component.

Such an interface is a requirement for EAL 4 in the Common Criteria.

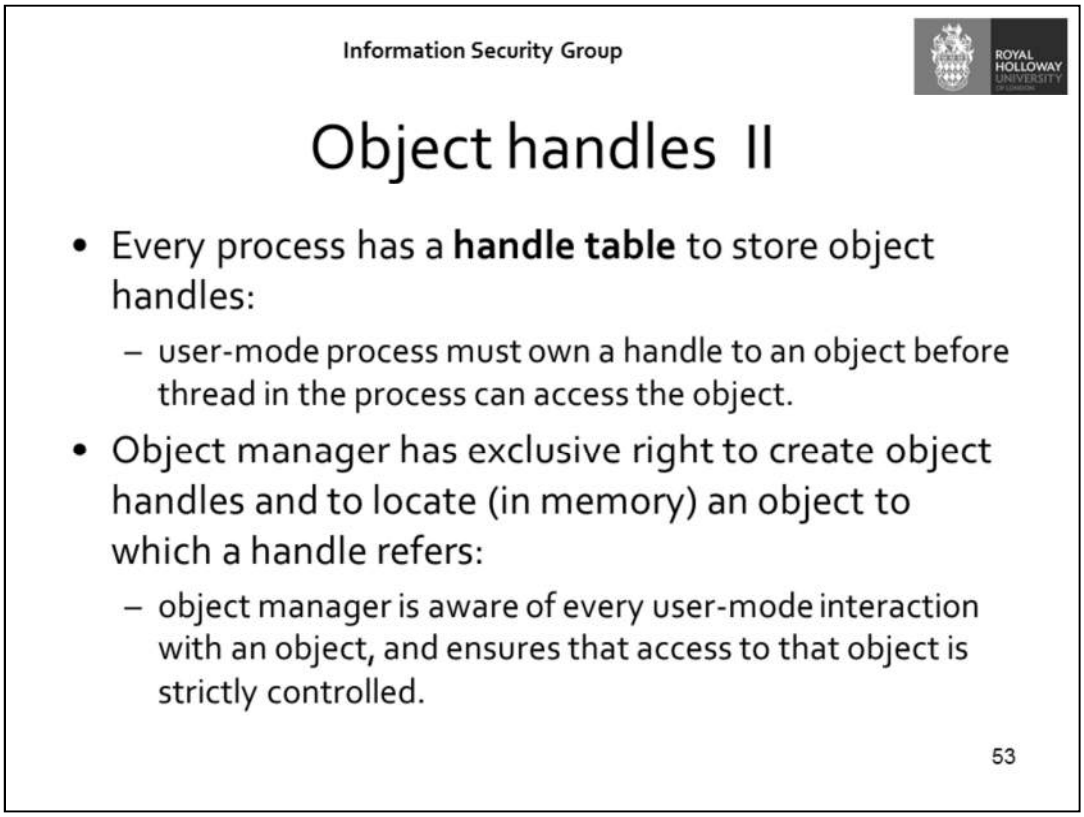



**Object handles** (a temporary reference (label) used by a process accessing an object) can only be created by the object manager.

When a process creates or opens an object it receives a **handle** from the object manager:

- the handle determines the way in which the process can access the object;
- note that kernel-mode code, including executive components and device drivers, can access objects in memory directly, i.e. without using a handle.

Object handles have the same structure for every type of object, and provide a consistent interface through which to access objects.




Every process maintains a **handle table** that stores object handles:

• a user-mode process must own a handle to an object before a threads in the process can access the object.

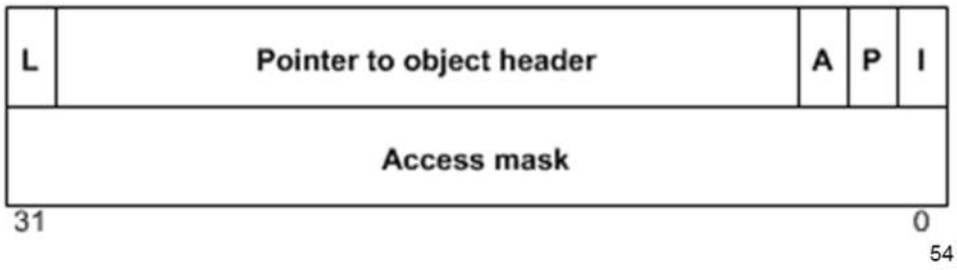
The object manager has the exclusive right to create object handles and to locate (in memory) an object to which a handle refers:

• the object manager is aware of every user-mode interaction with an object, and ensures that access to that object is strictly controlled.

Information Security Group

# Object handles III

- Object handle is 64-bit data structure containing:
  - 28-bit address (pointer) to object header;
  - 32-bit access mask stating how object can be accessed;
  - lock (L), protect (P), audit (A) and inherit (I) flags, used to control: access, generation of audit information, and creation of additional handles to object for child processes.




54

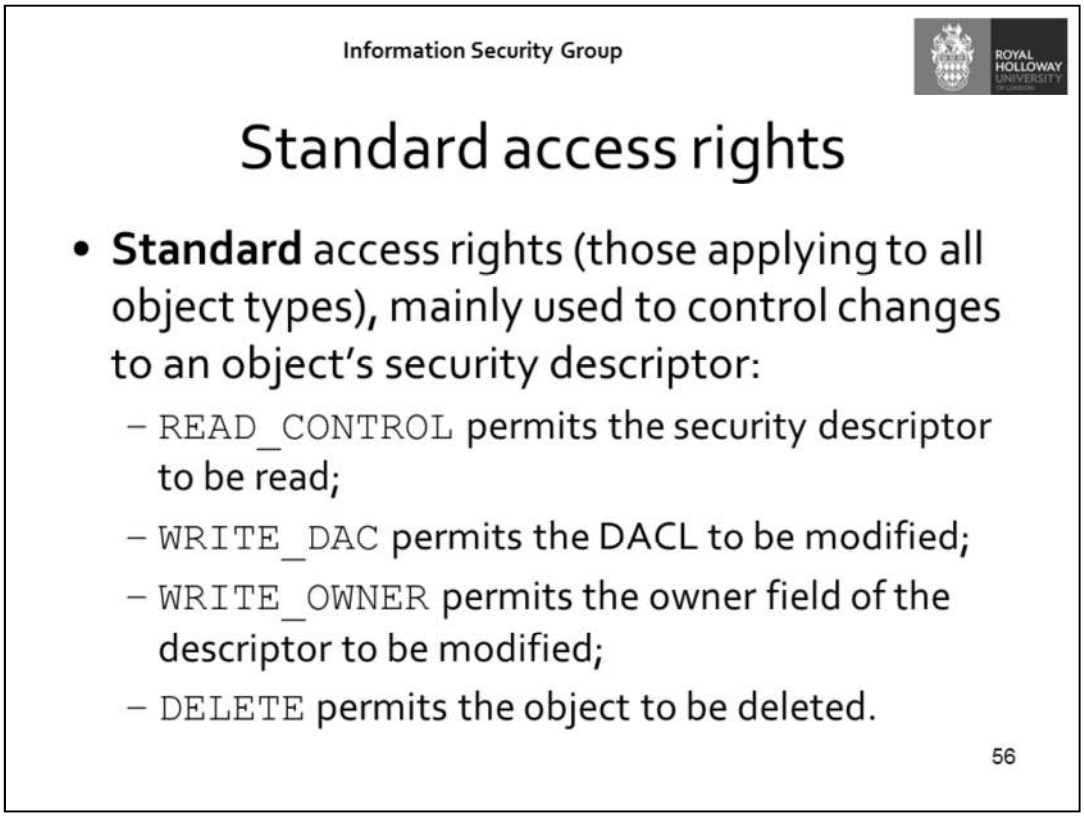
An object handle is a 64-bit data structure that contains:

• a 28-bit address (pointer) to an object header;

• a 32-bit access mask that indicates the ways in which the object can be accessed;

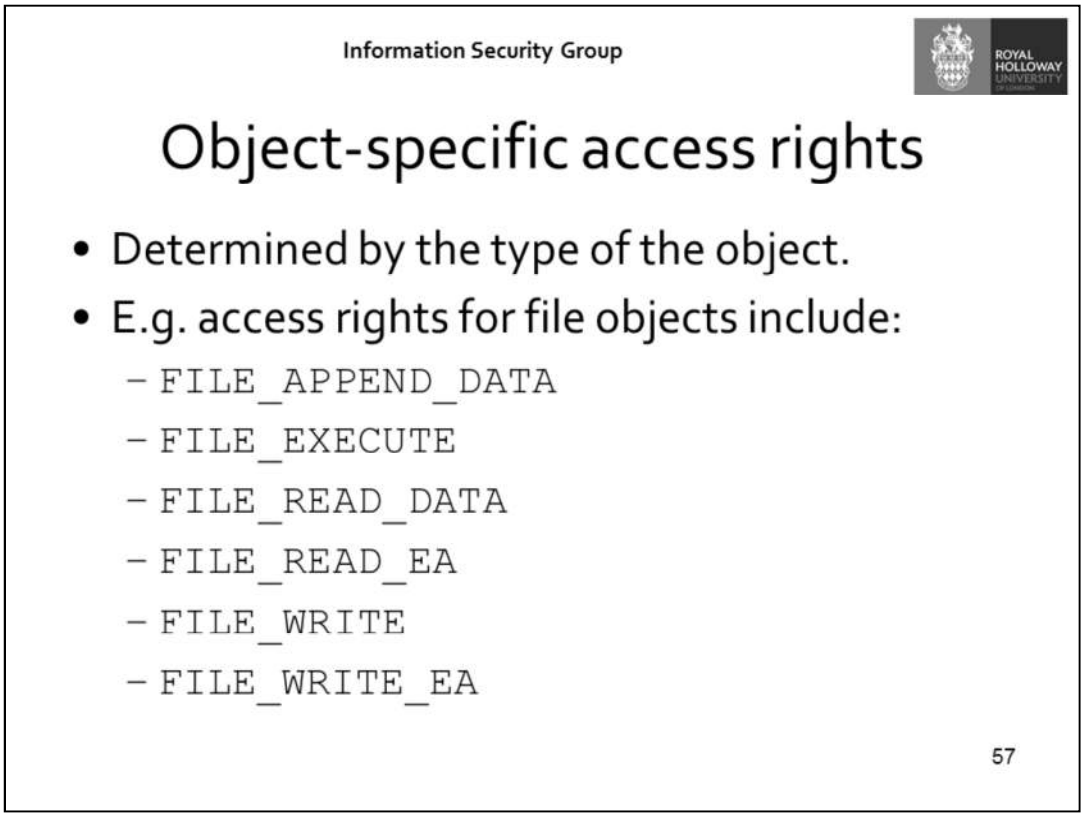
• lock (L), protect (P), audit (A) and inherit (I) flags, that are used by the object manager to control access to the object, the generation of audit information, and the creation of additional handles to the object for child processes.


Within an authorisation system, access rights specify what sorts of accesses subjects may have to objects.



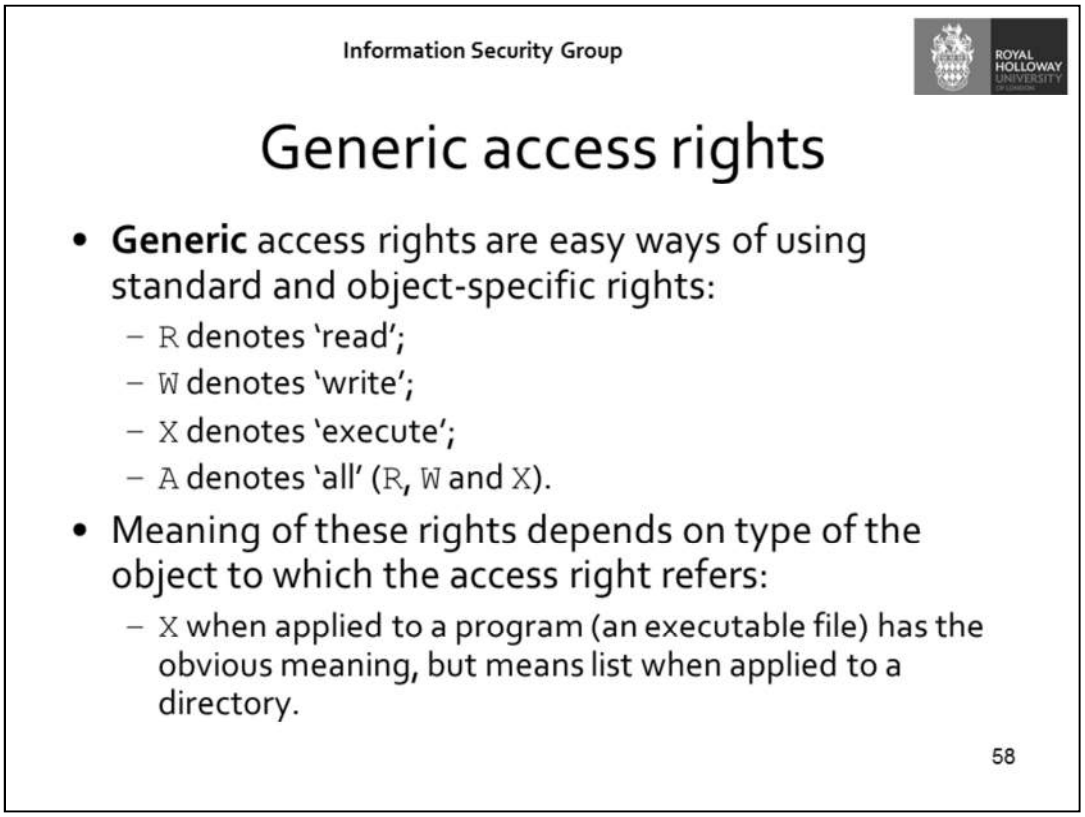

**Standard** access rights, i.e. those access rights which apply to objects of all types, are mainly used to control who can make changes to an object's security descriptor:

- `READ_CONTROL` permits the security descriptor to be read;
- `WRITE_DAC` permits the DACL to be modified;
- `WRITE_OWNER` permits the owner field of the descriptor to be modified;
- `DELETE` permits the object to be deleted.




**Object-specific** access rights are determined by the type of the object. For example, access rights for **file objects** (i.e. objects of type file) include:

```
FILE_APPEND_DATA
FILE_EXECUTE
FILE_READ_DATA
FILE_READ_EA
FILE_WRITE
FILE_WRITE_EA
```

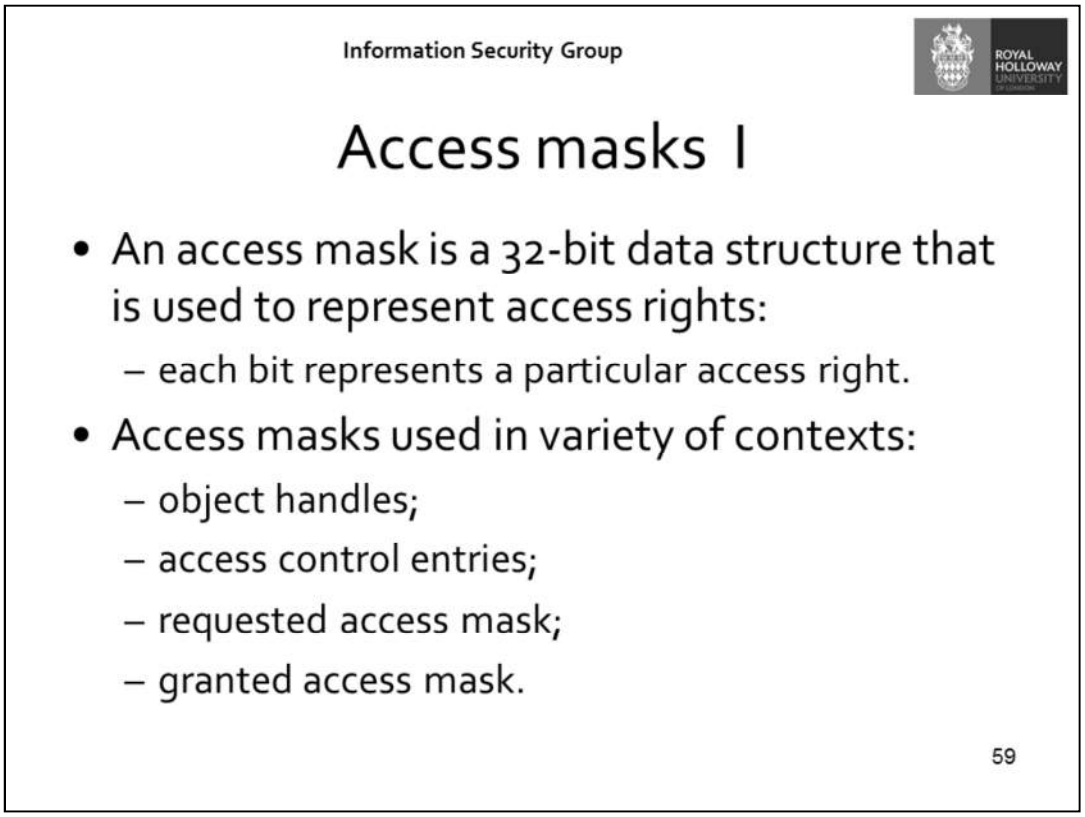



**Generic** access rights are designed to make it easier to use access rights. They map onto specific standard and object-specific rights:

- `R` denotes 'read';
- `W` denotes 'write';
- `X` denotes 'execute';
- `A` denotes 'all' (`R`, `W` and `X`).

The precise meaning of each of these rights depends on the type of the object to which the access right refers:

- `X` when applied to a program (an executable file) has the obvious meaning, but it means the right to list when applied to a directory.



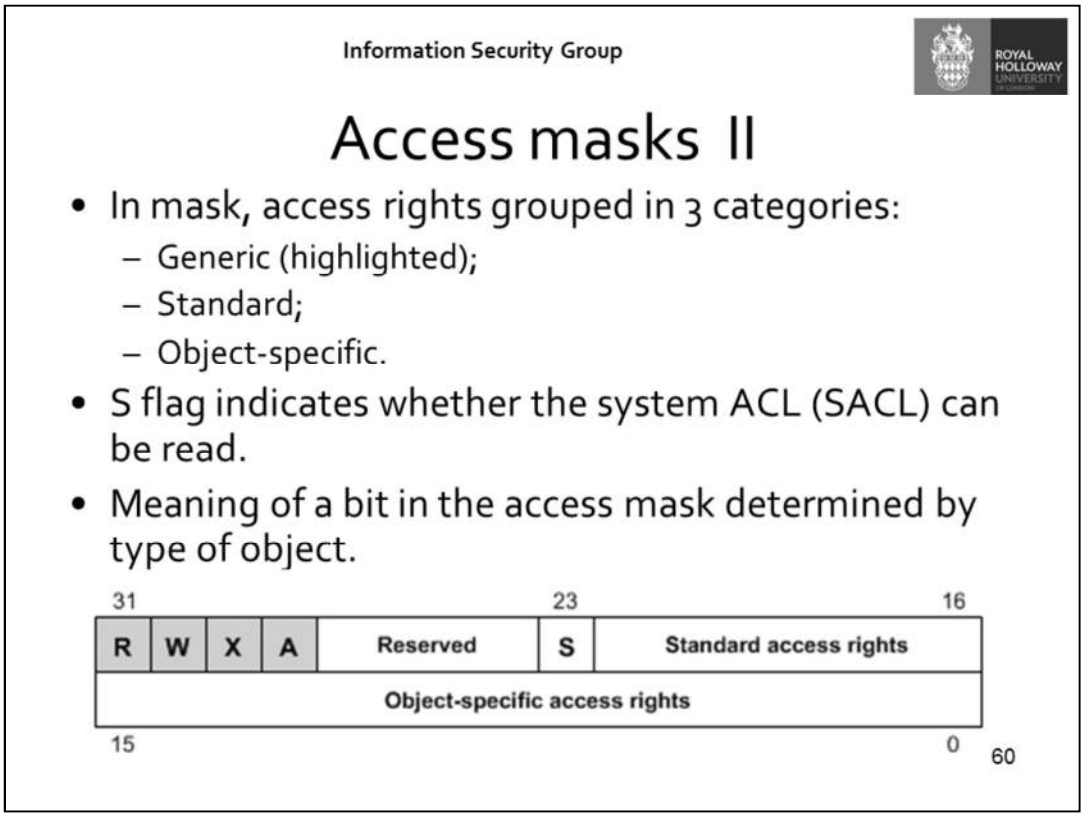

An access mask is a 32-bit data structure that is used to represent a set of access rights:

• each bit represents a particular access right, i.e. a bit set to 1 indicates possession of the right corresponding to that bit position.

Access masks are used in a variety of contexts:

- object handles;
- access control entries;
- the **requested access mask**;
- the **granted access mask**.

As we discuss later, the requested access mask and the granted access mask are generated by the Security Reference Monitor during the process of deciding whether a particular request for access should be granted.

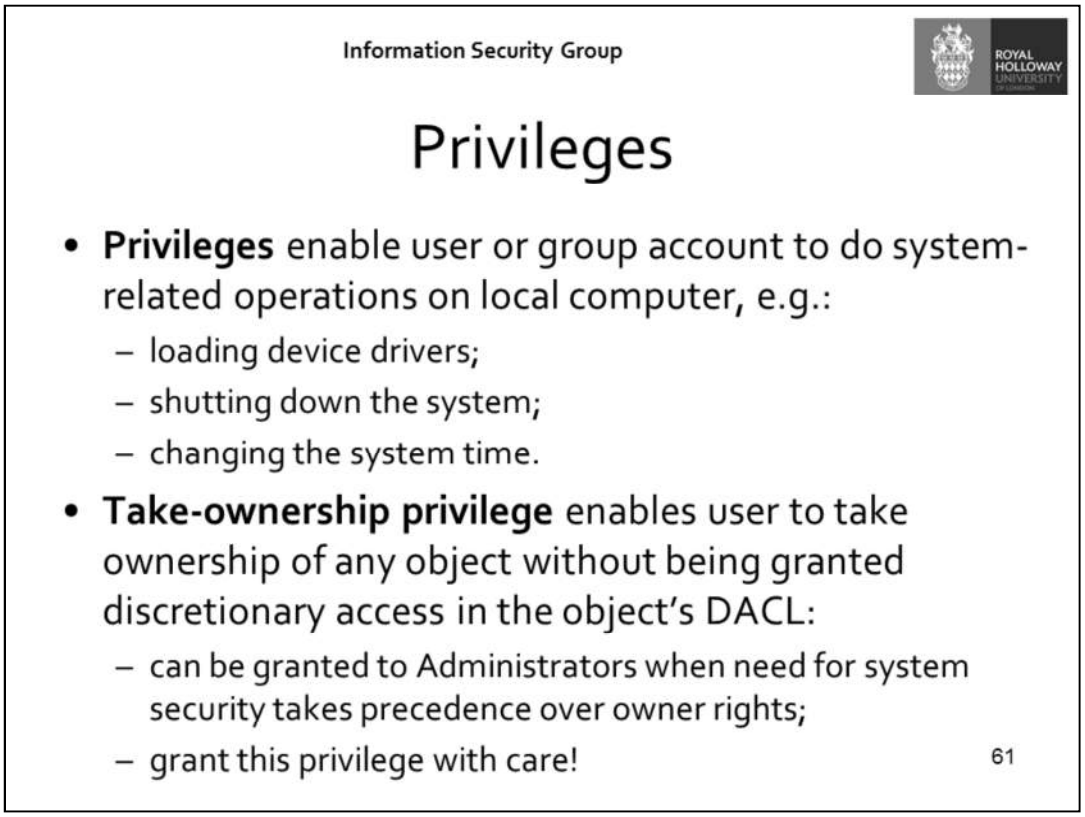



Within an access mark, access rights are grouped into three categories:

- Generic (shaded-in in the diagram on the slide);
- Standard;
- Object-specific.

The S flag indicates whether the system ACL (SACL) can be read.

The precise meaning of a particular bit in the access mask is determined by the type of the object to which the mask refers.

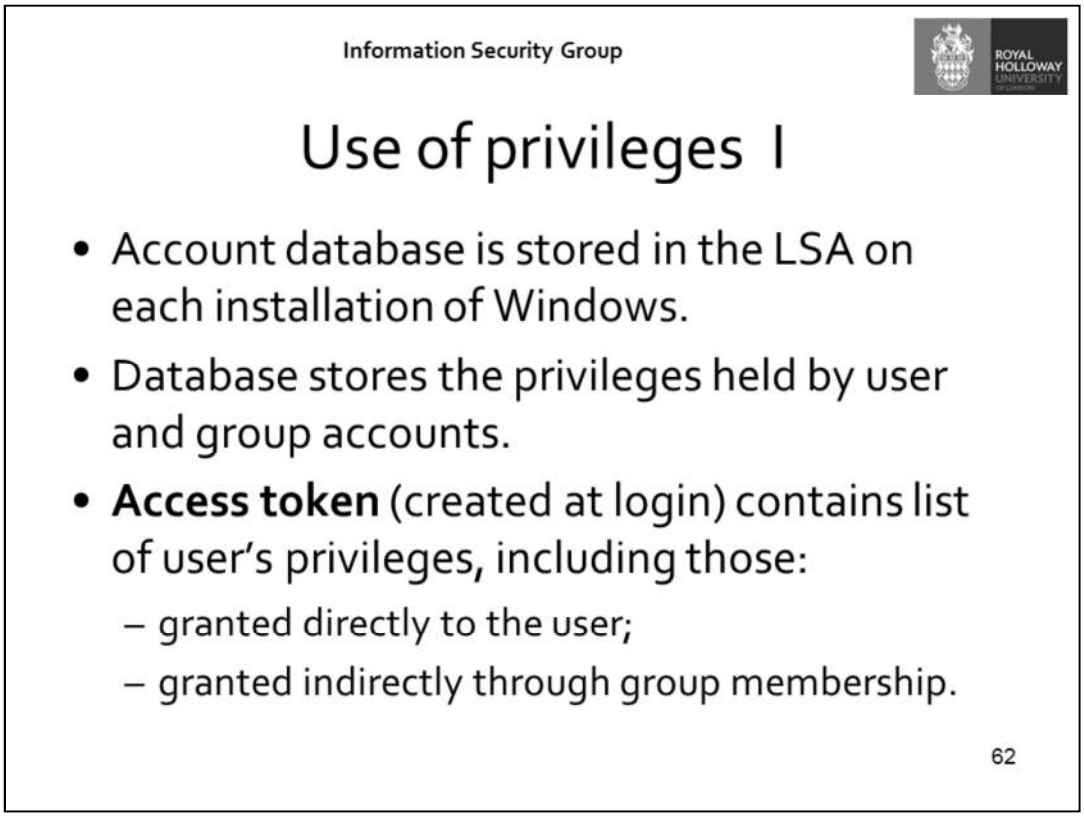



A **privilege** enables a user or group account to perform system-related operations on the local computer, for example:

- loading device drivers;
- shutting down the system; or
- changing the system time.

The **`take-ownership` privilege** enables a user to take ownership of any object without being granted discretionary access (in the object's DACL):

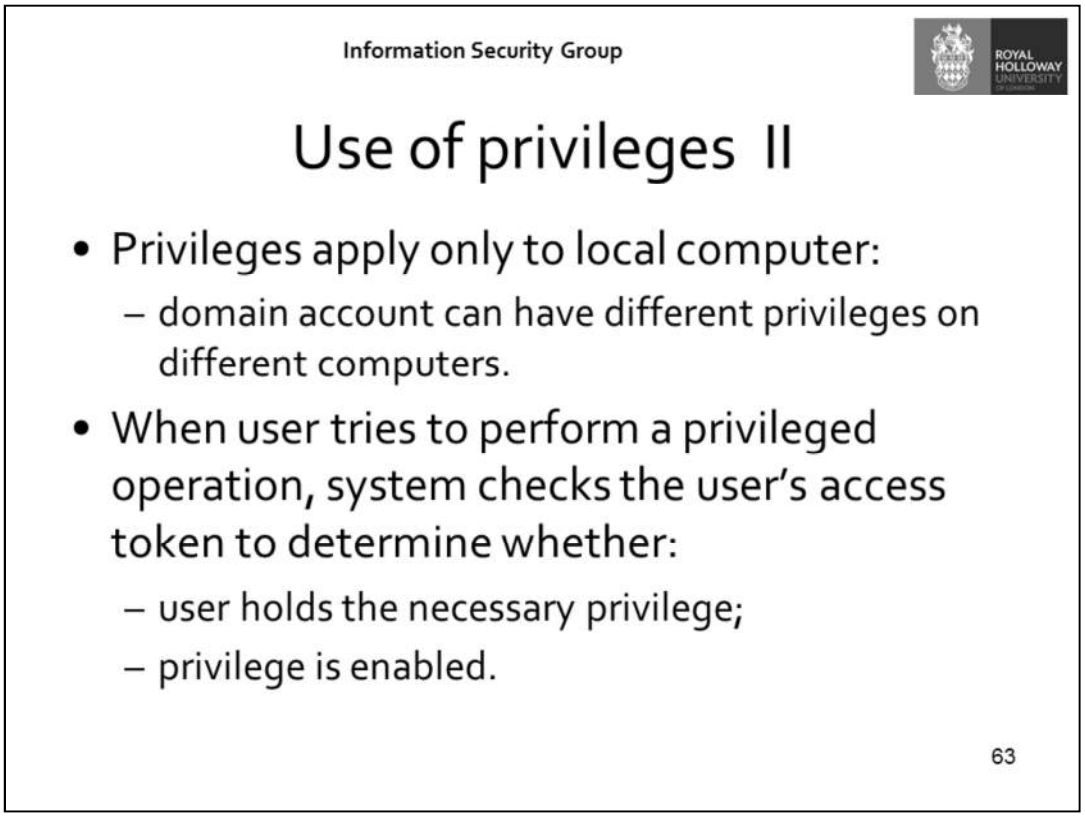
- can be granted to Administrators when the need to maintain the system security takes precedence over the owners' right to control access;
- this privilege is clearly very powerful, and so should therefore only be granted with care!




An account database is stored in the LSA (Local Security Authority) on each installation of Windows.

The database stores the privileges held by user and group accounts.

The access token (created when a user logs in) contains a list of the user's privileges, including those granted directly to the user and those granted indirectly through group membership.

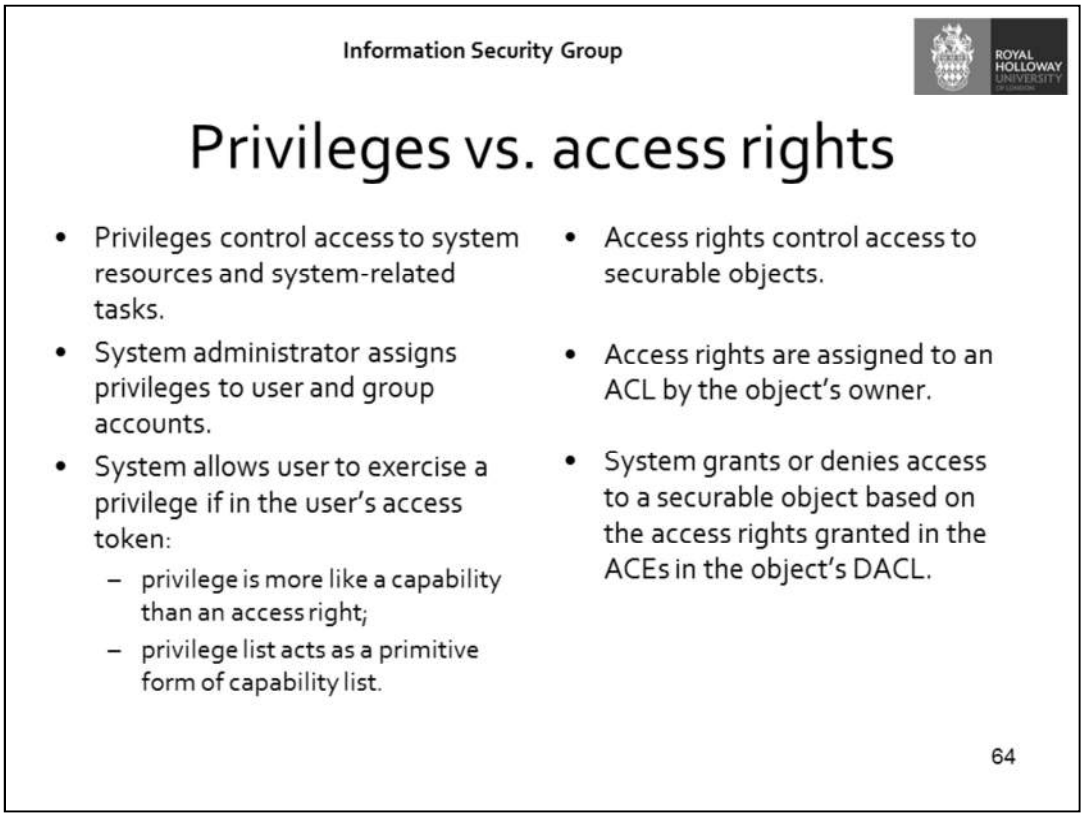



Note that the privileges apply only to the local computer:

- a domain account can have different privileges on different computers.

When the user tries to perform a privileged operation, the system checks the user's access token to determine whether:

- the user holds the necessary privilege;
- the privilege is enabled.




Privileges control access to system resources and system-related tasks. Access rights control access to securable objects.

A system administrator assigns privileges to user and group accounts. Access rights are assigned to an ACL by the object's owner.

The system permits a user to exercise a privilege if it is contained in the user's access token:

- a privilege is more like a capability than an access right;
- the privilege list acts as a primitive form of capability list.

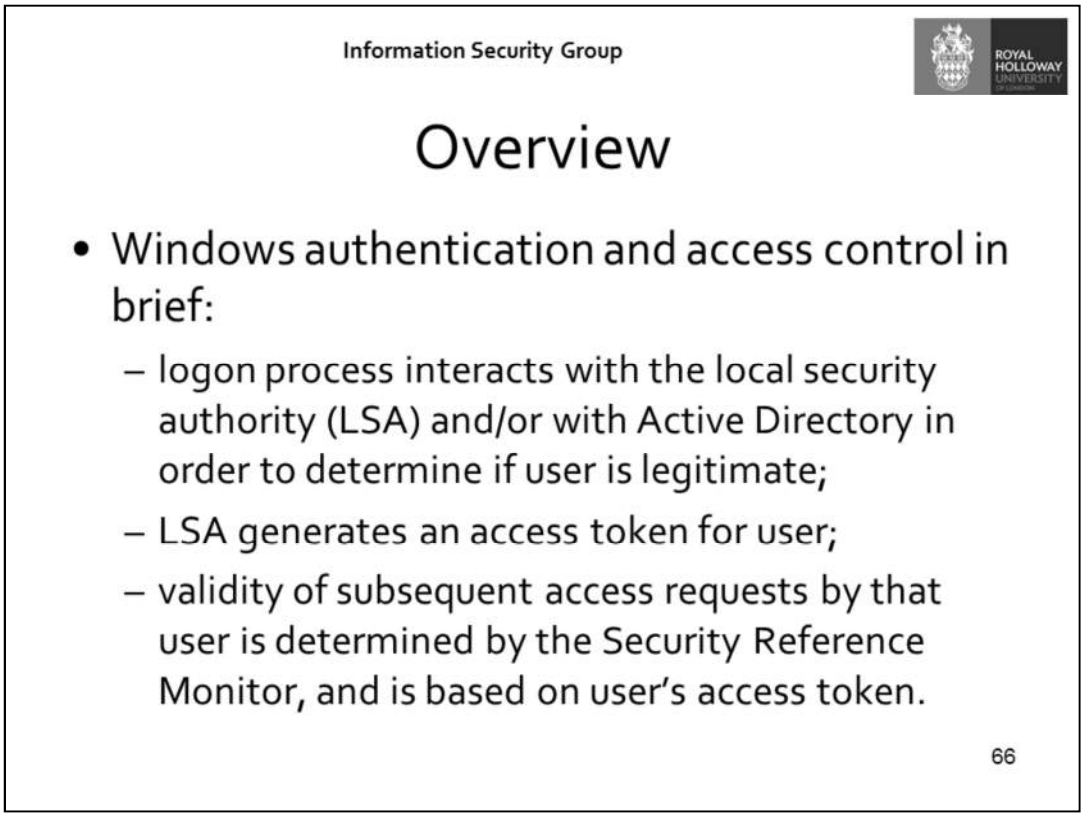
The system grants or denies access to a securable object based on the access rights granted in the ACEs in the object's DACL.

Information Security Group


# Agenda

- Introduction
- Active directory
- Domains
- Windows subjects and objects
- Access rights
- Authentication
- Access control
- Security management
- Resources

65

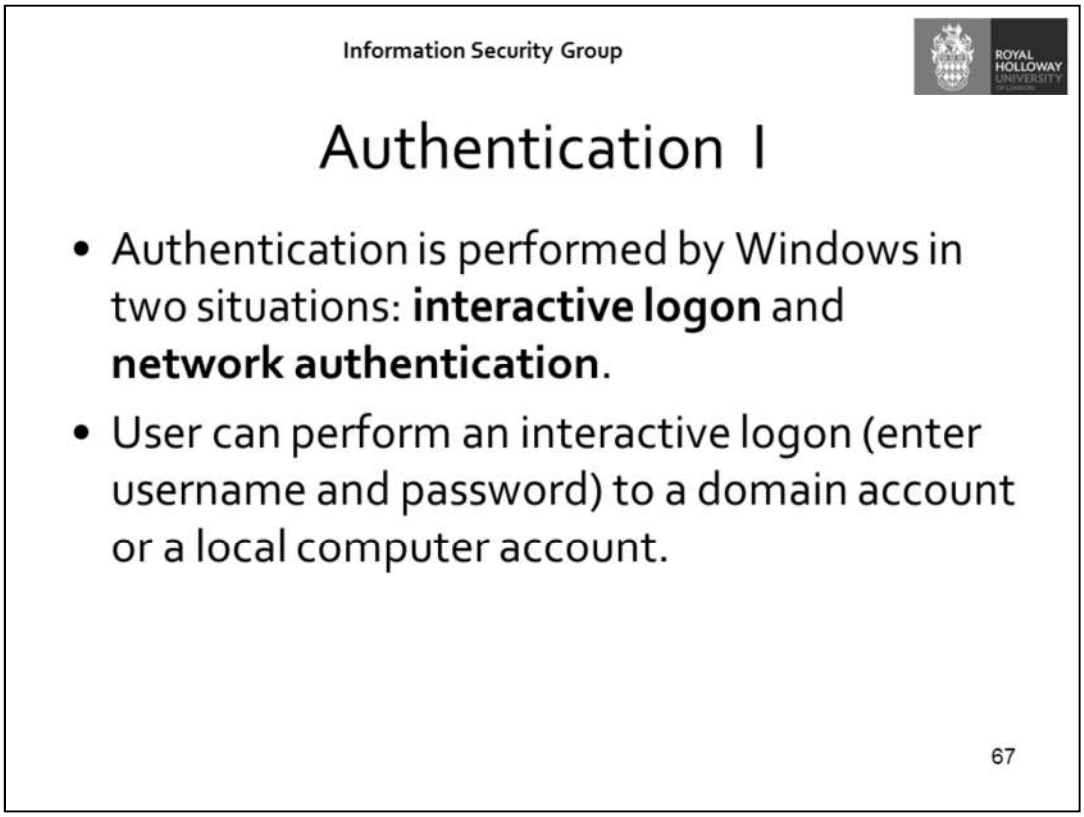



In brief, Windows authentication and access control operates as follows:

• the logon process interacts with the local security authority (LSA) and/or with Active Directory (depending on whether it is a local or a domain account) in order to determine if the user is legitimate;

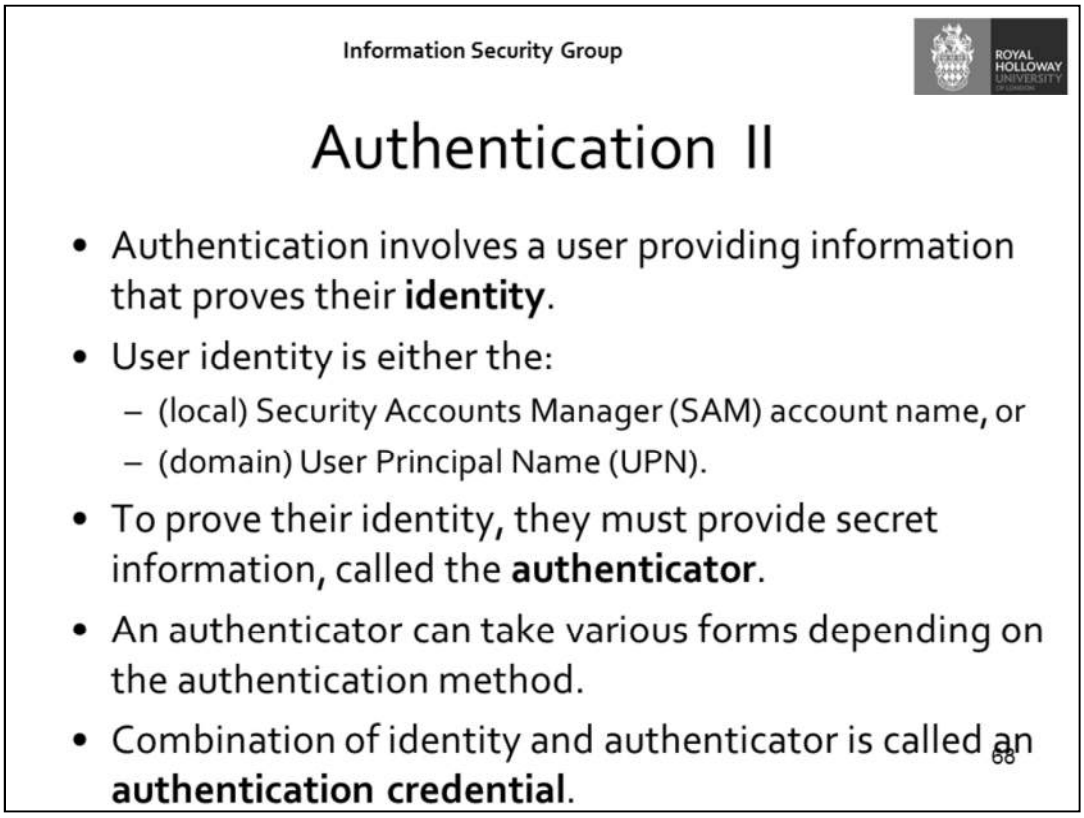
• if authentication succeeds, the LSA generates an access token for the authenticated user;

• the validity of subsequent access requests made by that user (or more specifically made by processes acting on behalf of that user) is determined by the Security Reference Monitor (the Windows term for the reference monitor), and is based on the user's access token.




Authentication is performed by Windows in two situations: **interactive logon** and **network authentication**, as described in the next few slides.

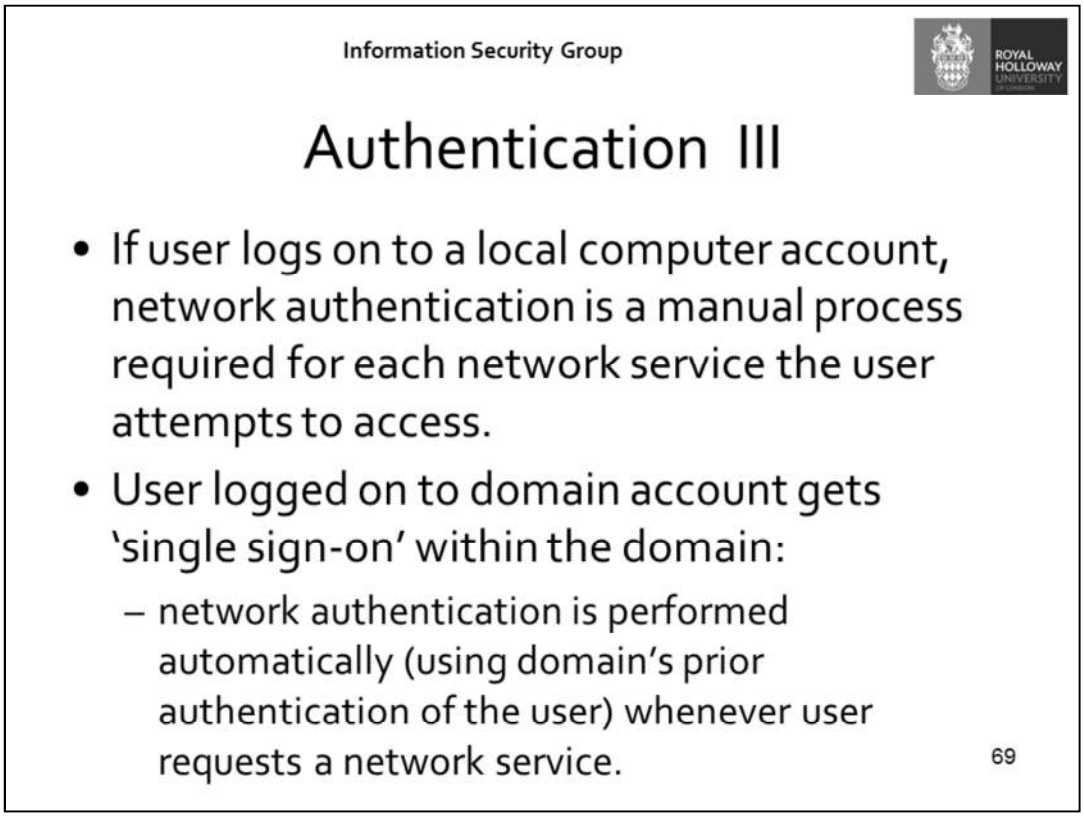
A user can perform an interactive logon (e.g. enter a username and password) to either a domain account (i.e. a managed account) or to a local computer account (i.e. one which only applies on the machine being used).




When a user or service wants to access a computing resource, they must provide information that proves their identity. This identity is typically in the form of an account user name. This might be a (local) user name in the form of a Security Accounts Manager (SAM) account name or a (domain) User Principal Name (UPN).

To prove ownership of an identity, the user must provide secret information, called the *authenticator*. An authenticator can take various forms depending on the authentication protocol and method. The combination of an identity and an authenticator is called an *authentication credential*.

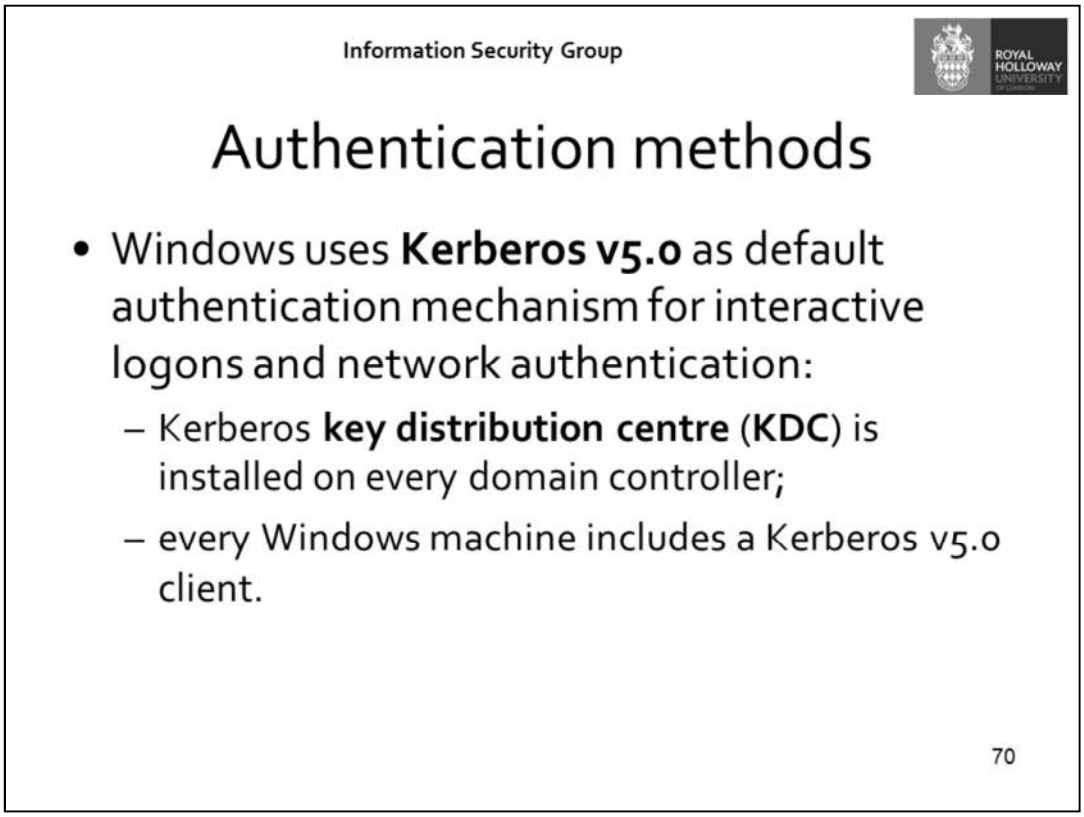
The process of creating, submitting, and verifying credentials is described simply as authentication, which can be implemented using a variety of authentication protocols, such as the Kerberos protocol. Authentication establishes the identity of the user, but not necessarily the user's permission to access or change a specific computing resource. That process is known as *authorisation*.




If a user only logs on to a local computer account, network authentication is a manual process that is required for each network service the user attempts to access.

On the other hand, a user logged on to a domain account benefits from 'single sign-on' within the domain:

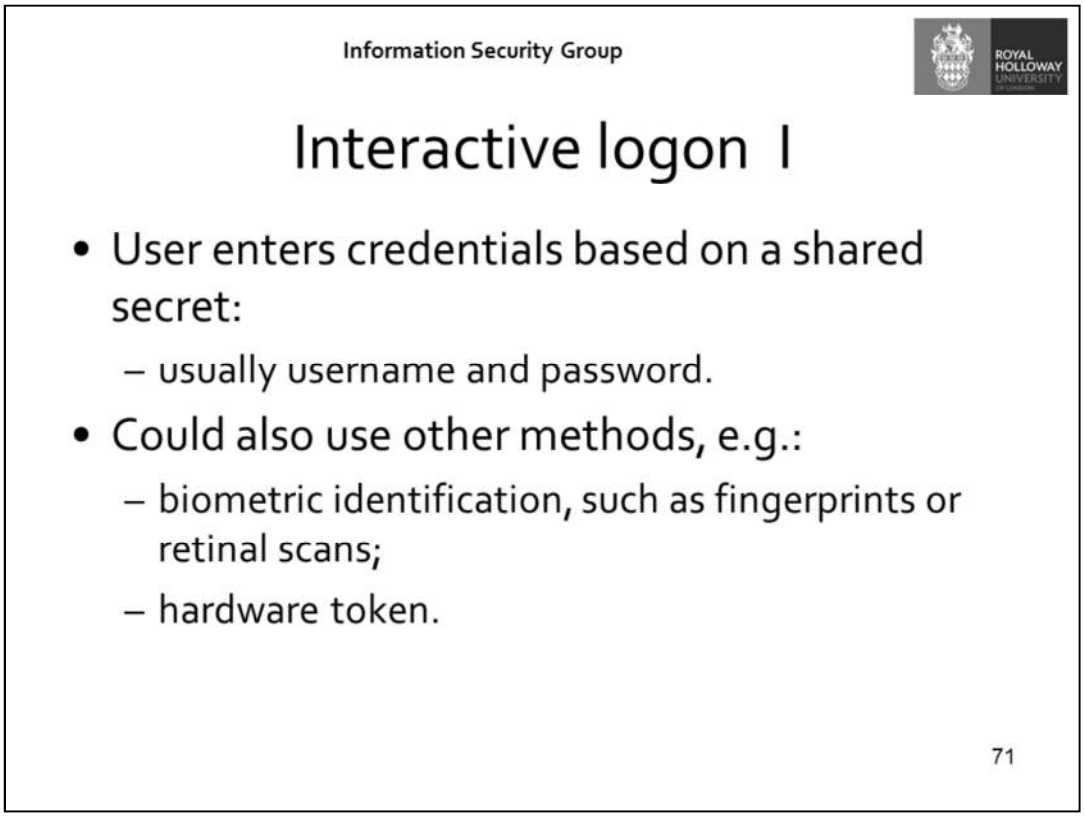
• network authentication is performed automatically (based on the domain's prior authentication of the user) whenever a user requests a network service.




Windows uses a modified version of **Kerberos v5.0** as the default authentication mechanism for interactive logons and network authentication:

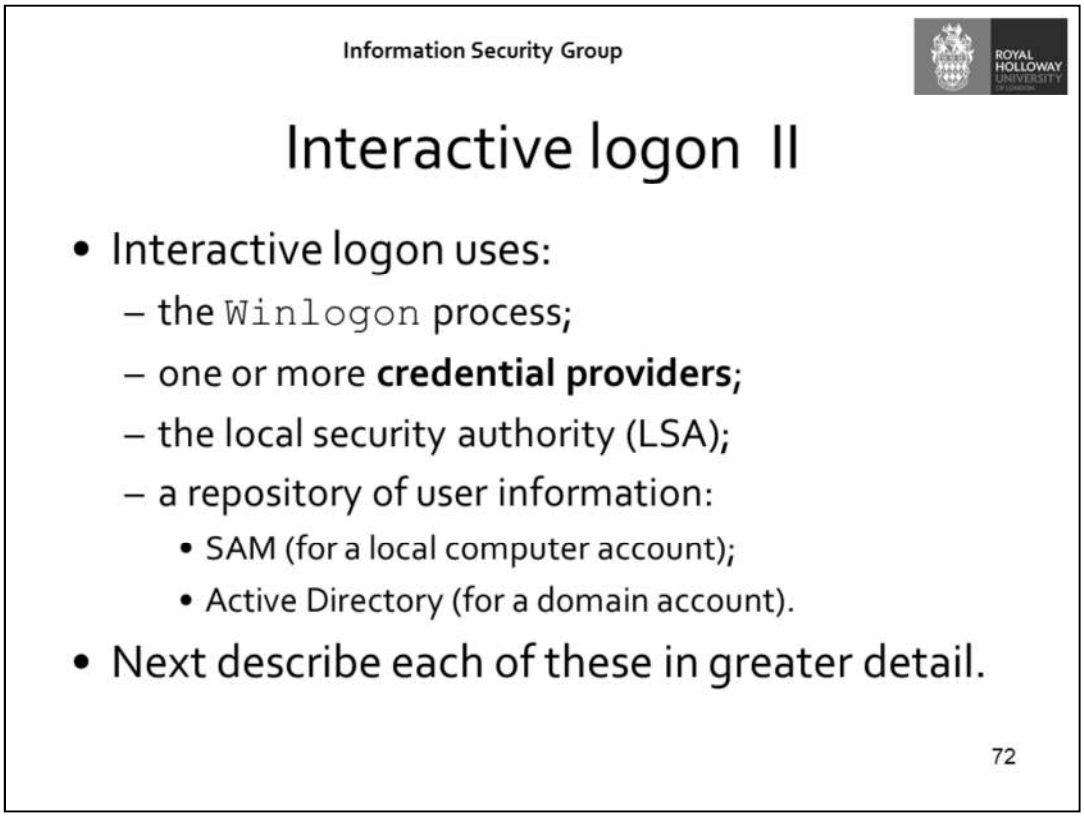
• a Kerberos **key distribution centre** (**KDC**) is installed on every domain controller;

• every Windows machine includes a Kerberos v5.0 client.




The user starts the process of interactive logon by entering credentials based on a shared secret. This is typically a username and password. However, the system could also be configured to employ other techniques, for example:

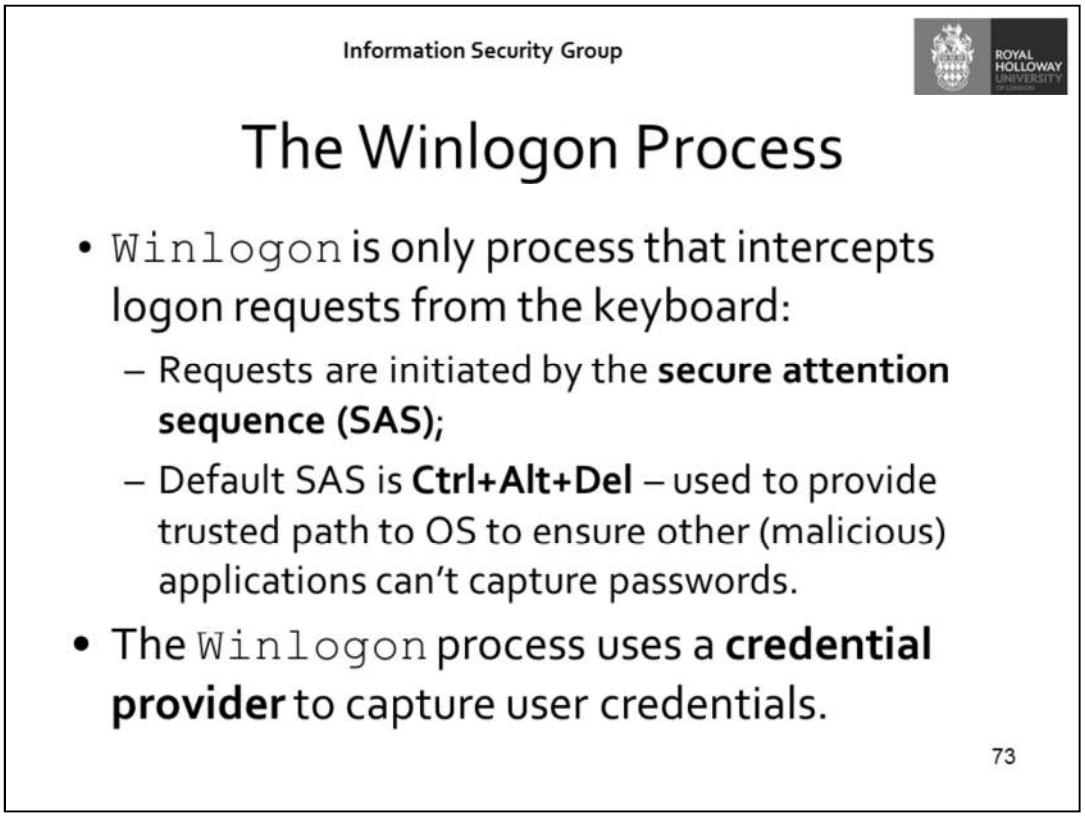
- biometric identification, such as fingerprints or retinal scans; or
- a hardware token.




The interactive logon procedure uses:

- the `Winlogon` Windows module (a process);
- one or more **credential providers**;
- the local security authority (LSA);
- a repository of user information, which is either:
  - the Security Account Manager (SAM) – in the case of a local computer account;
  - Active Directory – in the case of a domain account.

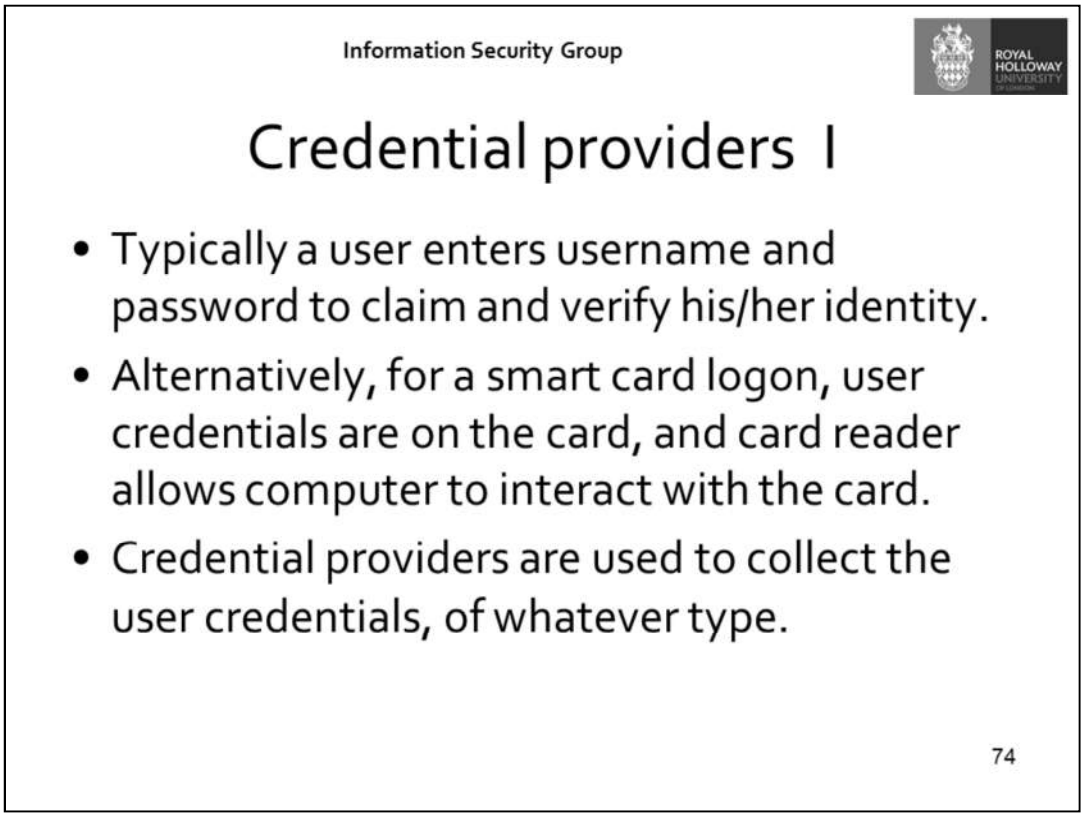
In the next few slides we describe each of these in greater detail.




`Winlogon` is the only process that intercepts logon requests from the keyboard:

- a logon request is initiated by the **secure attention sequence** (SAS);
- the default SAS is **Ctrl+Alt+Del**, and is used to provide a trusted path to the operating system to ensure that other (malicious) applications cannot capture user passwords.

The `Winlogon` process uses a credential provider to capture the user credentials, i.e. the information used to identify and authenticate the user.

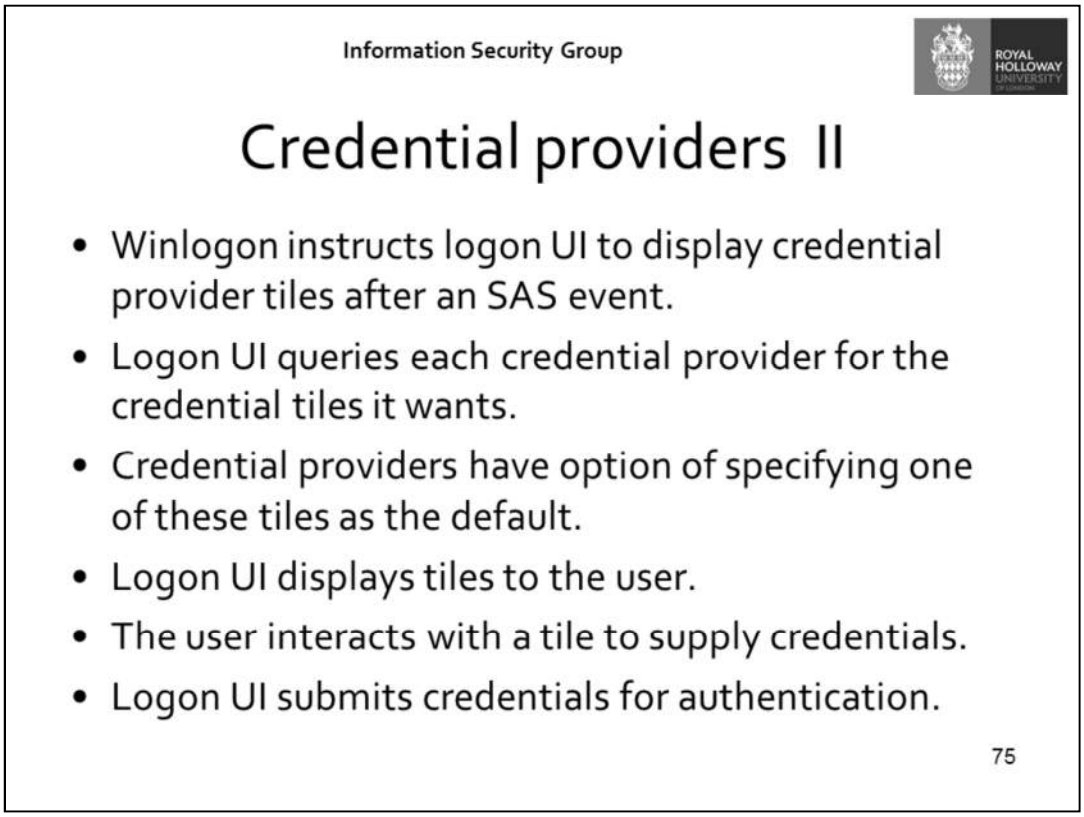



Typically, a user who logs on to a computer must enter a user name and password. These credentials are used to identify the user and verify the user's identity.

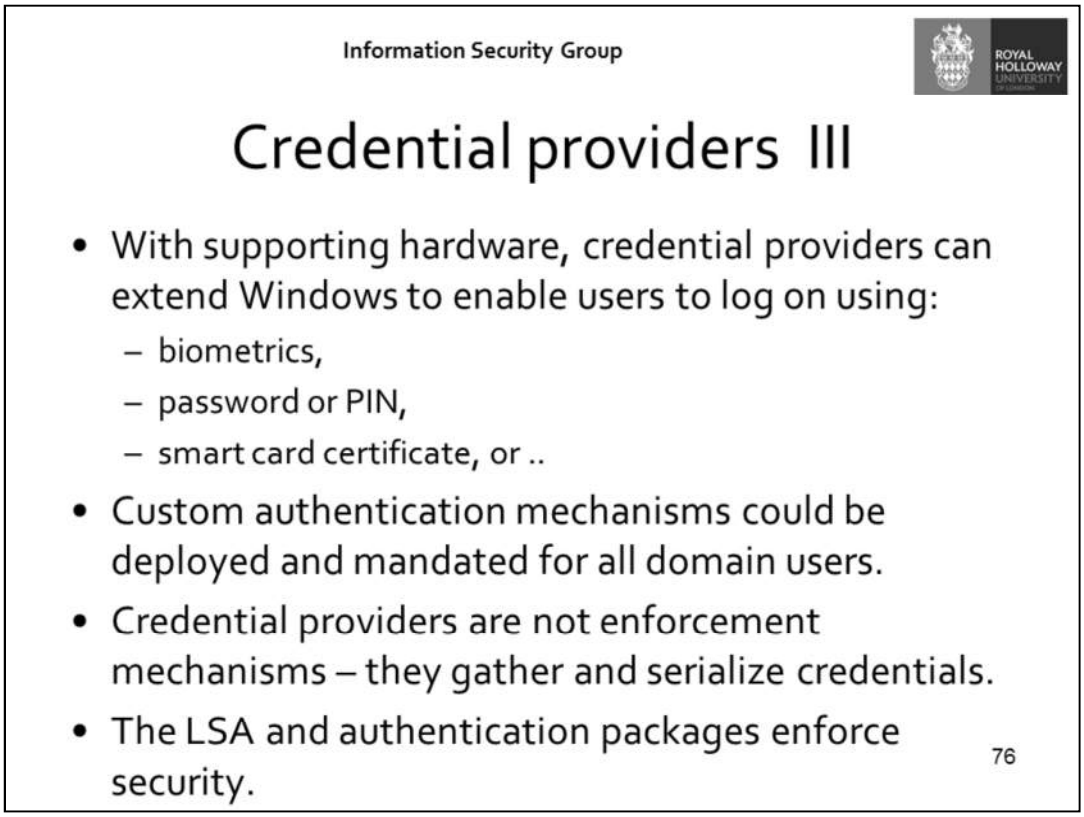
Alternatively, for a smart card logon, a user's credentials are contained on the smart card's security chip, and a smart card reader lets the computer interact with the security chip on the smart card. During a smart card logon, a user enters a personal identification number (PIN) instead of a user name and password.

Credential providers are Windows components used to collect credentials, whether they are passwords, smart card interactions, biometrics, or whatever.

In summary, the logon user interface (UI) provides interactive UI rendering, Winlogon provides the interactive logon infrastructure, and credential providers work with both of these components to help gather and process credentials.



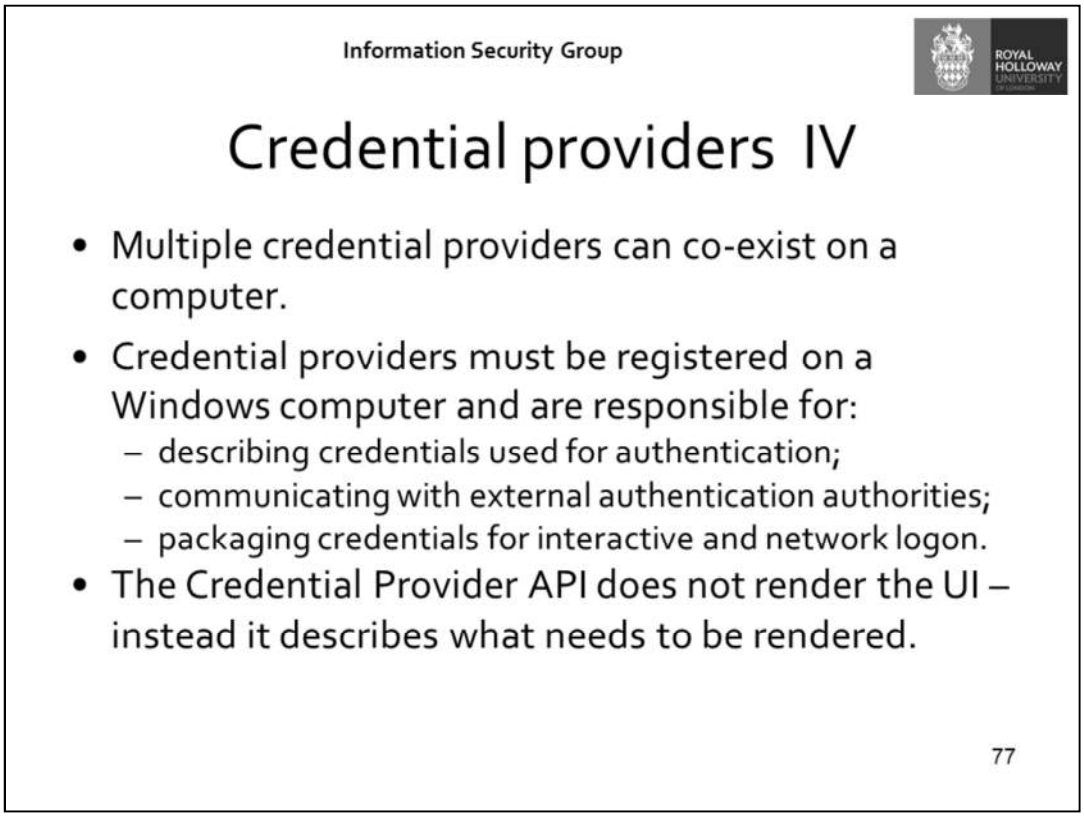

Winlogon instructs the logon UI to display credential provider tiles after it receives a SAS event. The logon UI queries each credential provider for the number of credentials it wants to display to the user. Credential providers have the option of specifying one of these tiles as the default. After all the providers have specified their tiles, the logon UI displays them to the user. The user interacts with a tile to supply his or her credentials. The logon UI submits these credentials for authentication.




Combined with supporting hardware, credential providers can extend Windows to enable users to log on by using biometrics (for example, fingerprint, retinal, or voice recognition), password, PIN, smart card certificate, or any custom authentication package.

Enterprises and IT professionals could develop and deploy custom authentication mechanisms for all domain users and can choose to explicitly require users to use this custom logon mechanism.

Credential providers are not enforcement mechanisms. They are used to gather and serialize credentials. The LSA and authentication packages (e.g. NTLM and Kerberos) enforce security.



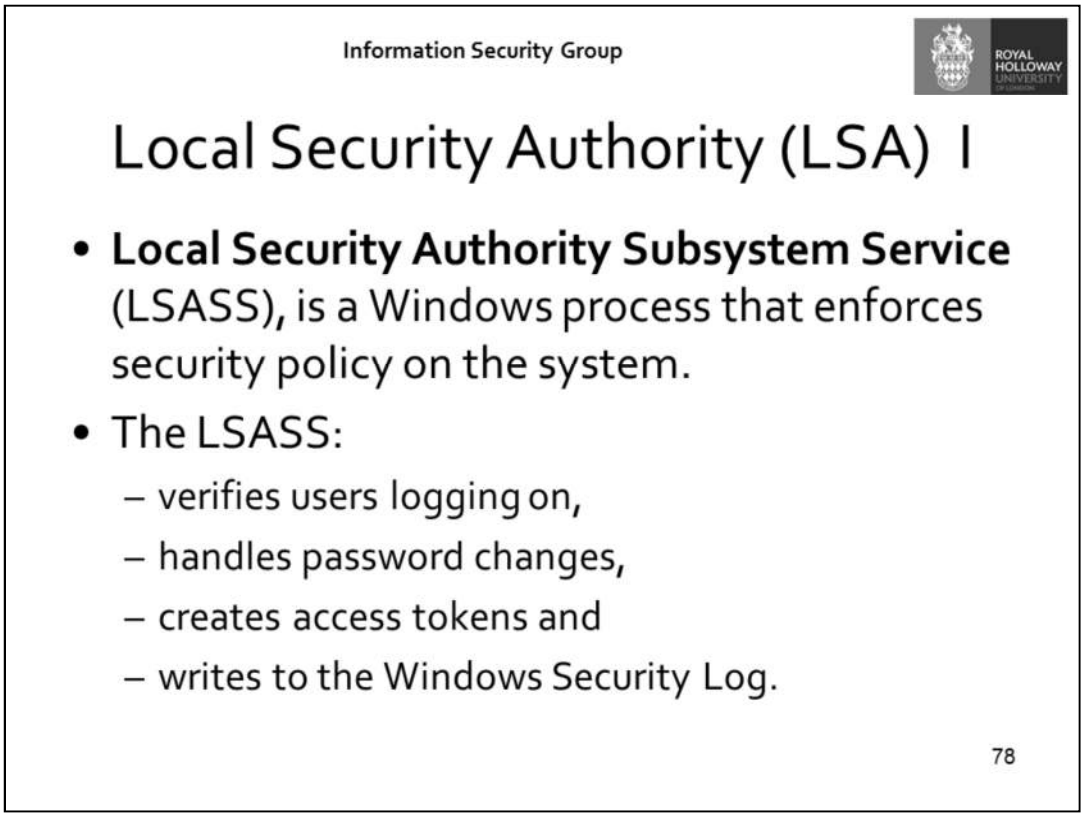

Credential providers may be designed to support single sign-on (SSO), authenticating users to a secure network access point (by using RADIUS and other technologies) and computer logon. Credential providers are also designed to support application-specific credential gathering, and may be used for authentication to network resources, joining computers to a domain, or to provide administrator consent for User Account Control (UAC).

Multiple credential providers may co-exist on a computer.

Credential providers must be registered on a Windows computer and are responsible for:

- Describing the credential information required for authentication.
- Handling communication and logic with external authentication authorities.
- Packaging credentials for interactive and network logon.

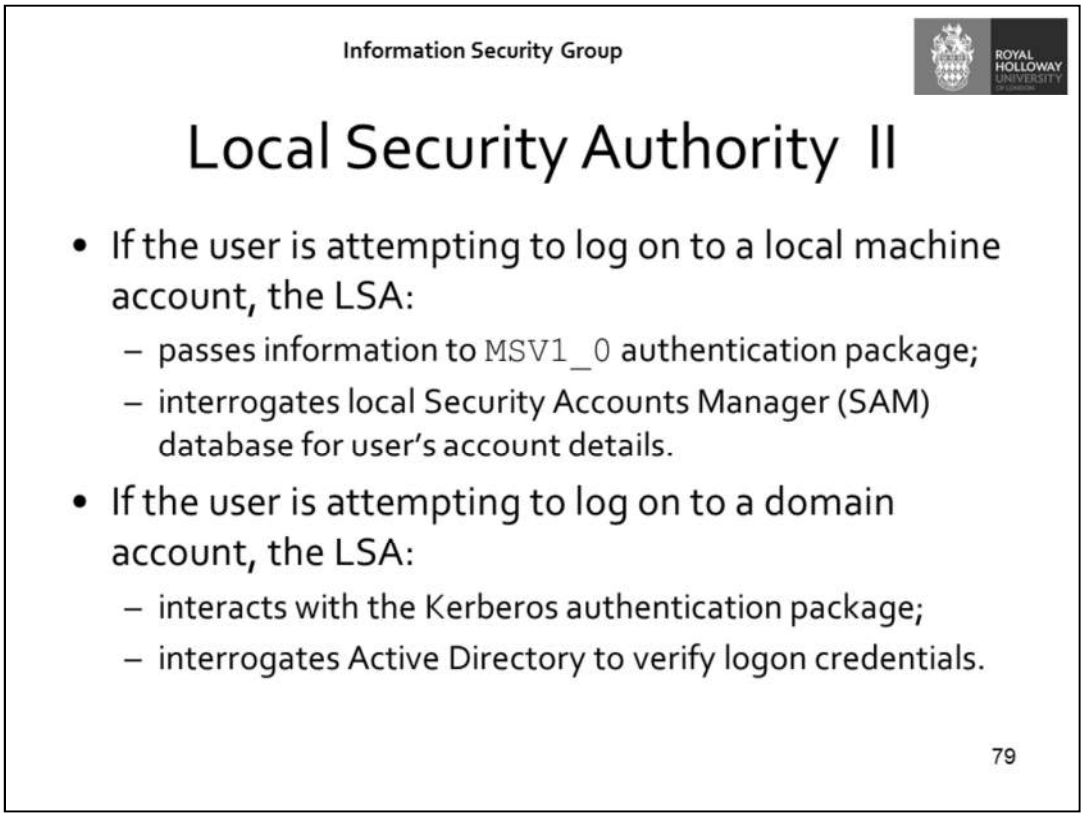
The Credential Provider API does not render the UI. Instead it describes to the logon UI what needs to be rendered.




The **Local Security Authority Subsystem Service** (LSASS), is a process in Windows responsible for enforcing the security policy on the system.

It verifies users logging on to a Windows computer or server, handles password changes, and creates access tokens. It also writes to the Windows Security Log.

Forcible termination of lsass.exe (the LSASS process) will result in the Welcome screen losing its accounts, prompting a restart of the machine.

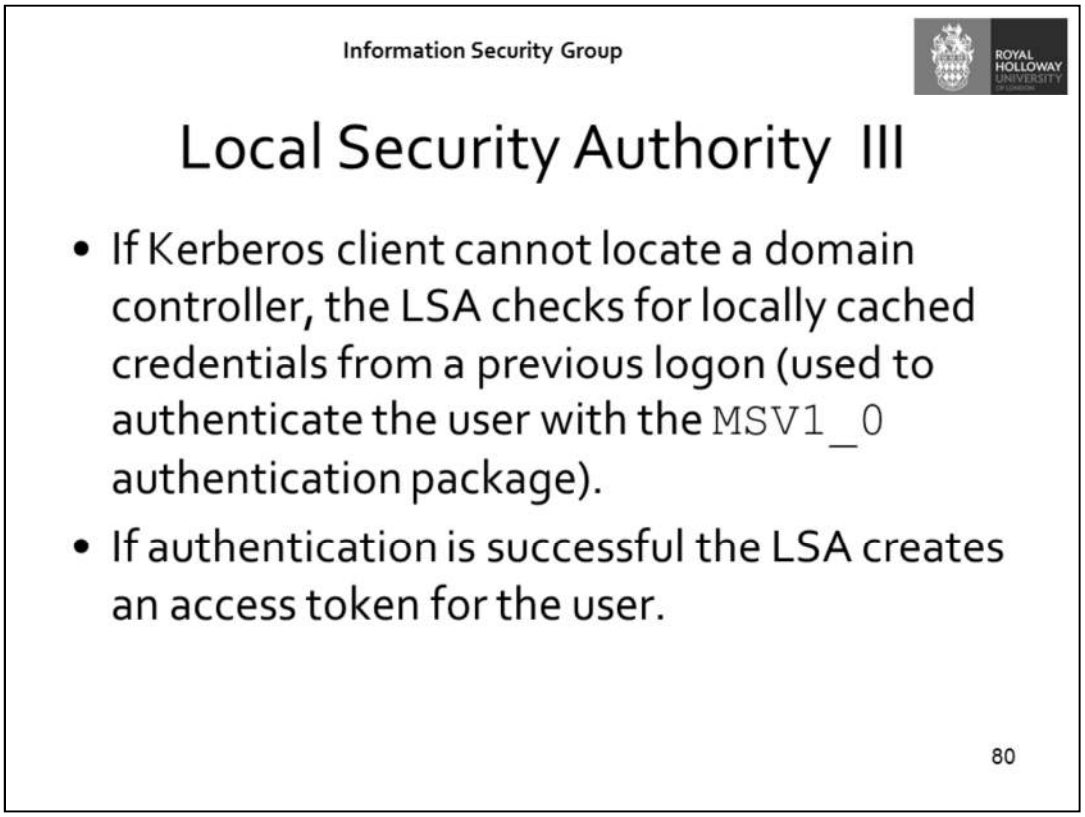



If the user is attempting to log on to a local machine account, the LSA:

- passes the information to the `MSV1_0` authentication package;
- interrogates the local Security Accounts Manager (SAM) database for the user's account details.

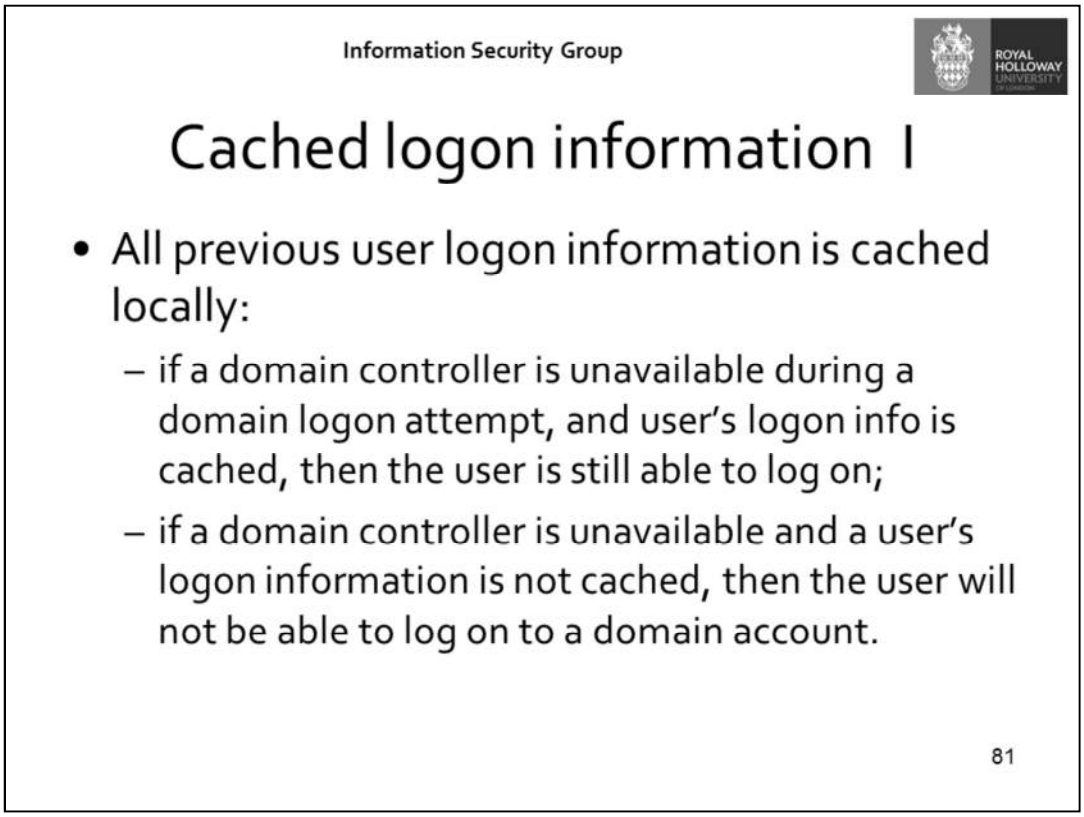
If the user is attempting to log on to a domain account, the LSA:

- interacts with the Kerberos authentication package;
- interrogates Active Directory to verify the logon credentials.




If a Kerberos client cannot locate a domain controller, the LSA checks for locally cached credentials from a previous logon (which are used to authenticate the user with the `MSV1_0` authentication package).

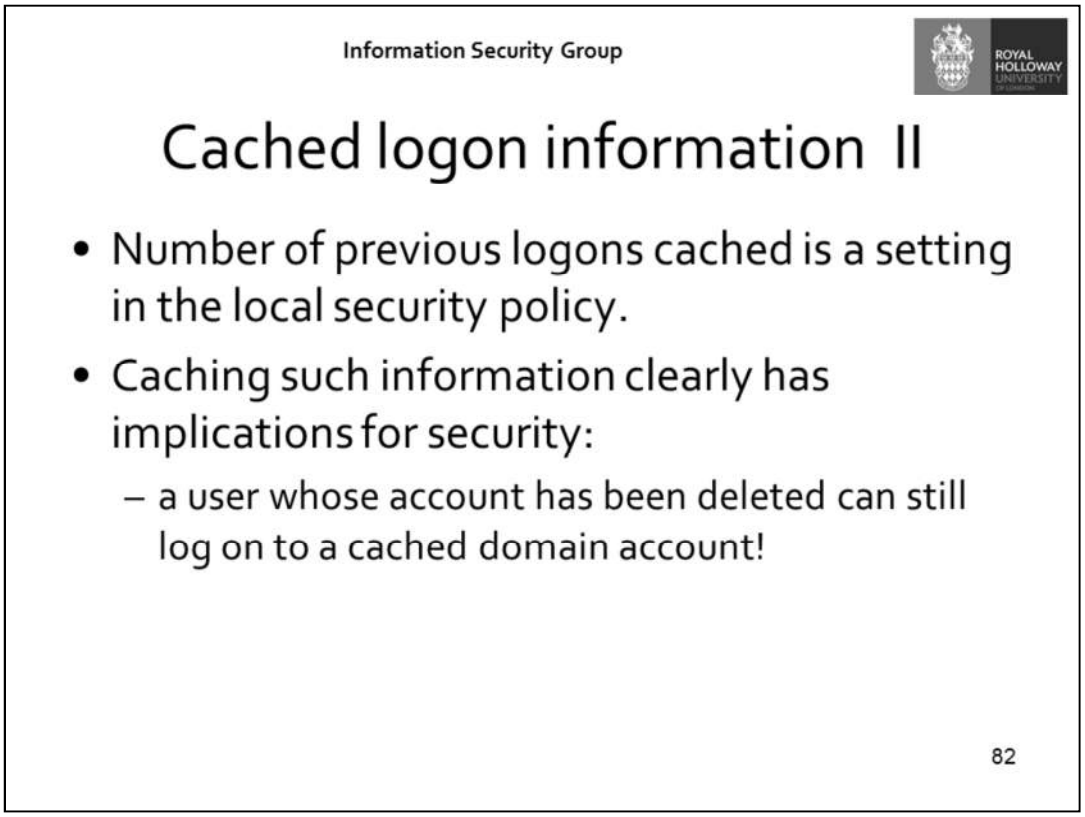
If authentication is successful the LSA creates an access token for the user.




All previous user logon information is cached locally. As a result:

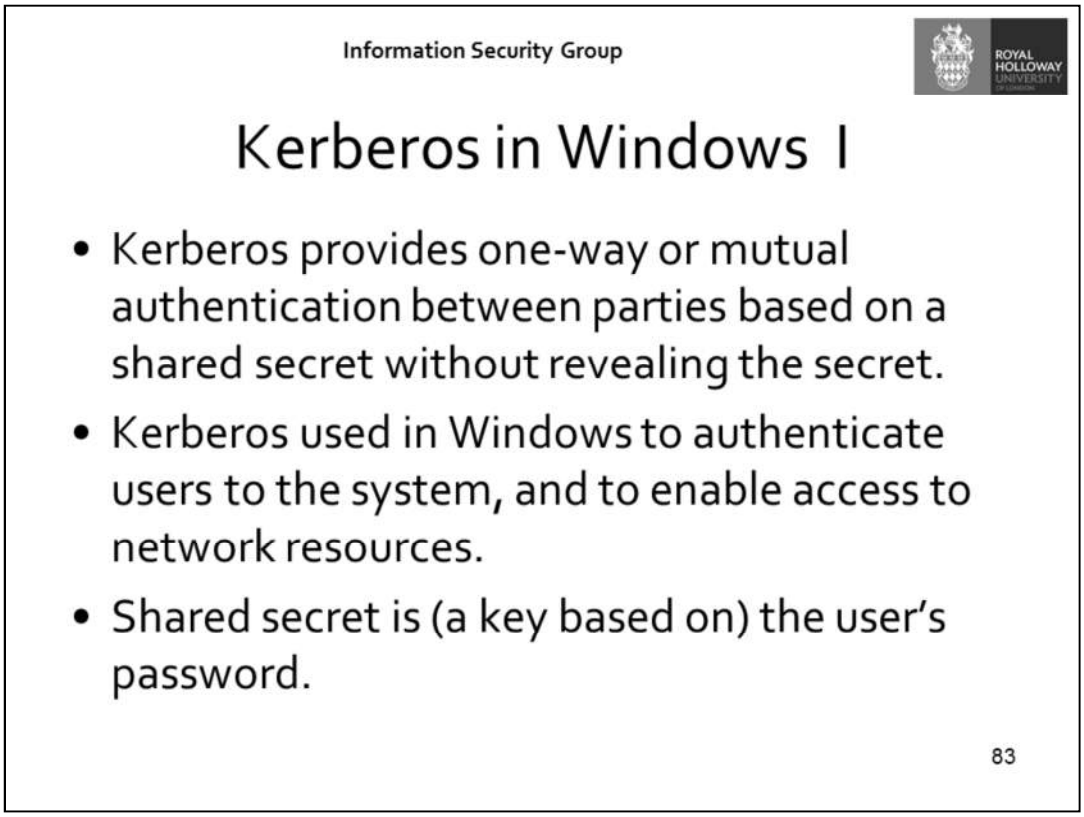
• if a domain controller is unavailable during subsequent domain account logon attempts, then the user is still able to log on;

• if a domain controller is unavailable and a user's logon information is not cached, then the user will not be able to log on to a domain account.




The number of previous logons that are cached is a setting in the local security policy.

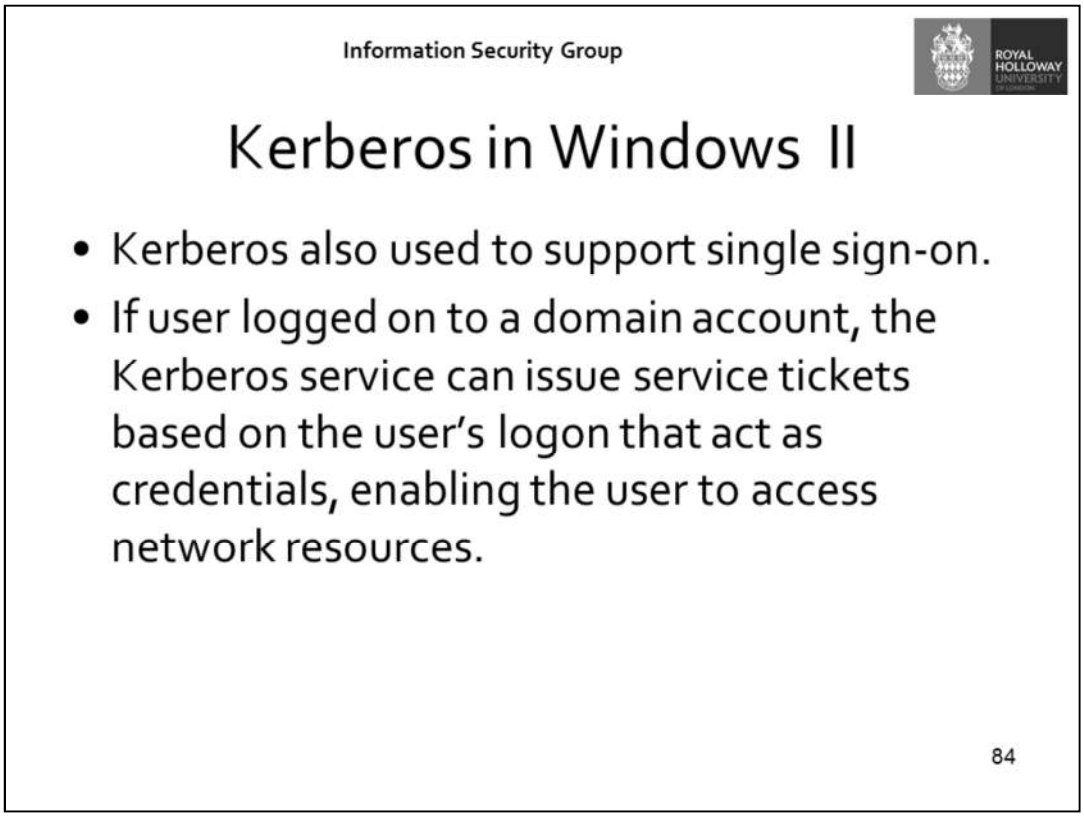
Caching such information clearly has implications for security. For example, a user whose account has been deleted can still log on to a cached domain account!




The Kerberos protocol provides either one-way or mutual authentication between two parties based on a shared secret, without revealing the secret.

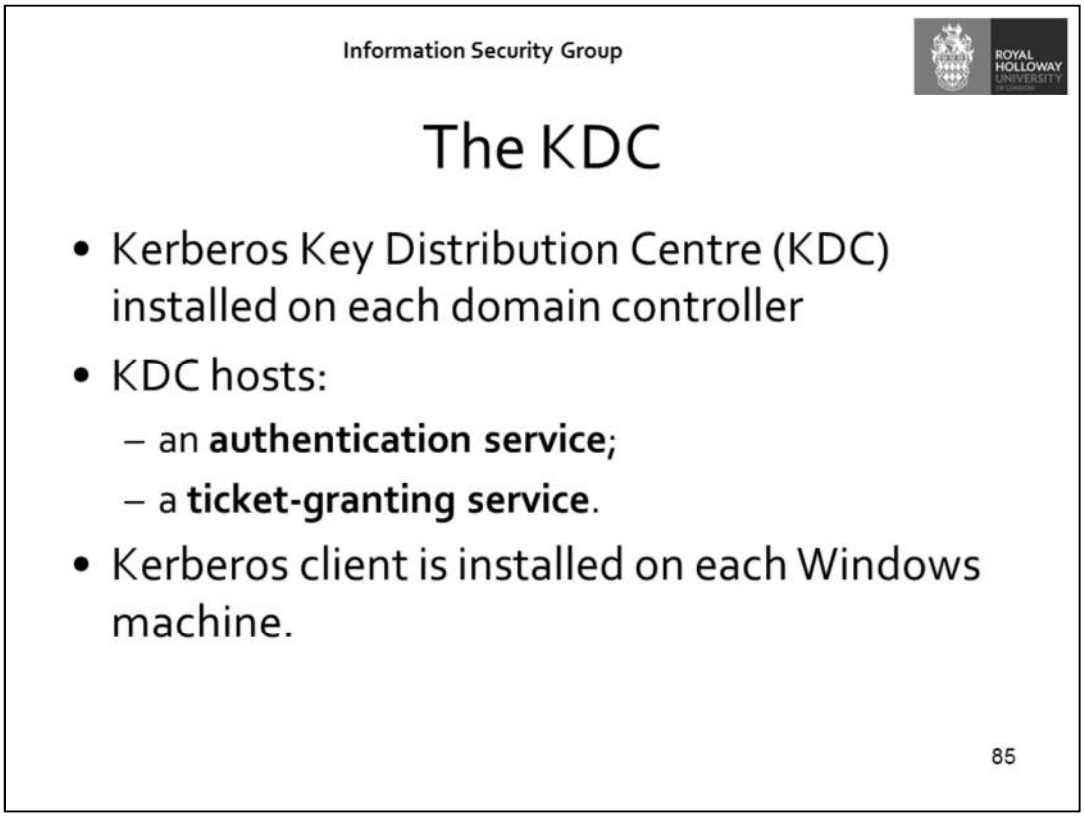
Kerberos is used in Windows to authenticate users to the system, and to enable access to network resources:

The shared secret is a cryptographic key derived from the user's password.




Kerberos is also used to support single sign-on.

If a user is logged on to a domain account, the Kerberos service can issue service tickets based on the user's logon; these serve as credentials, enabling the user to access network resources.

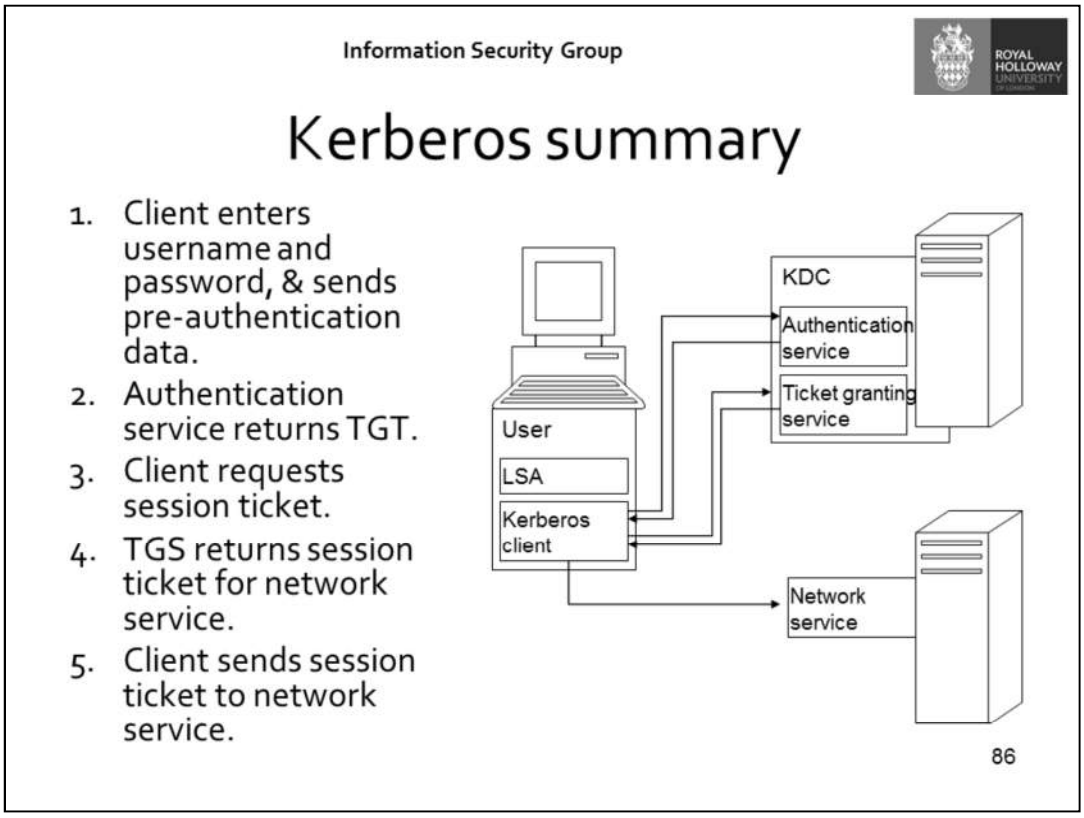



A Kerberos Key Distribution Centre (KDC) is installed on each domain controller

The KDC hosts:

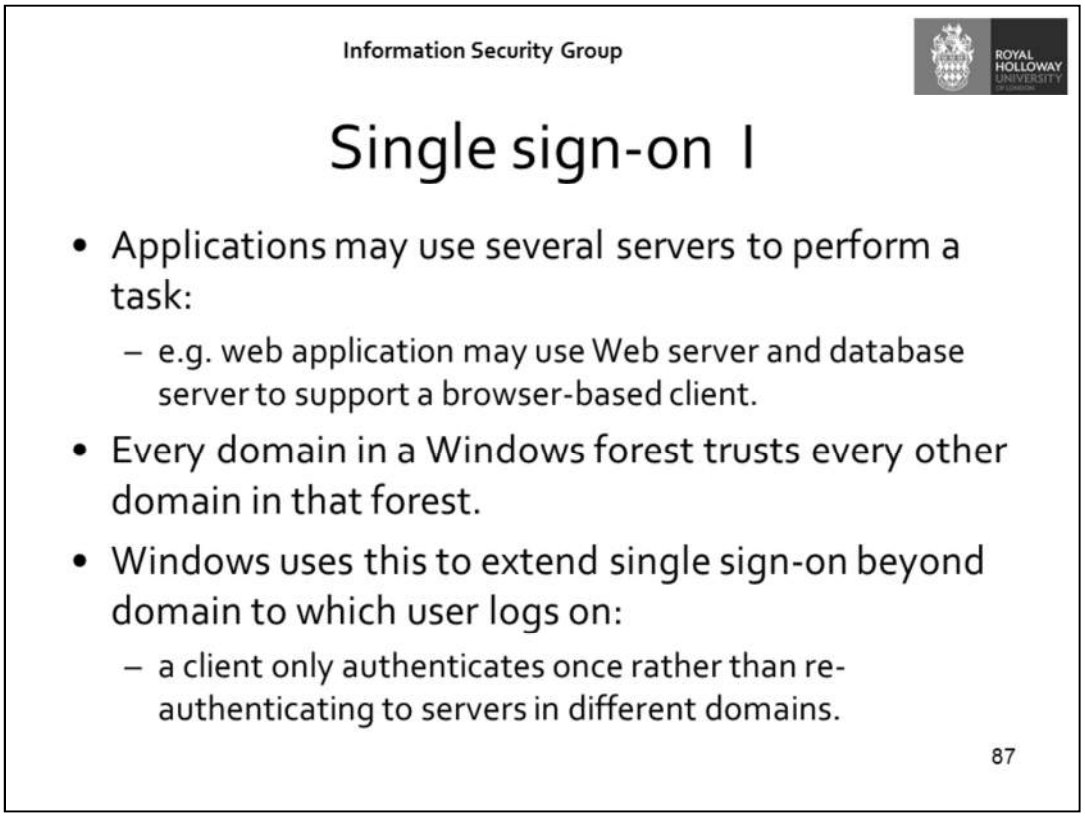
- a Kerberos **authentication service**; and
- a Kerberos **ticket-granting service**.

A Kerberos client is installed on each Windows machine.



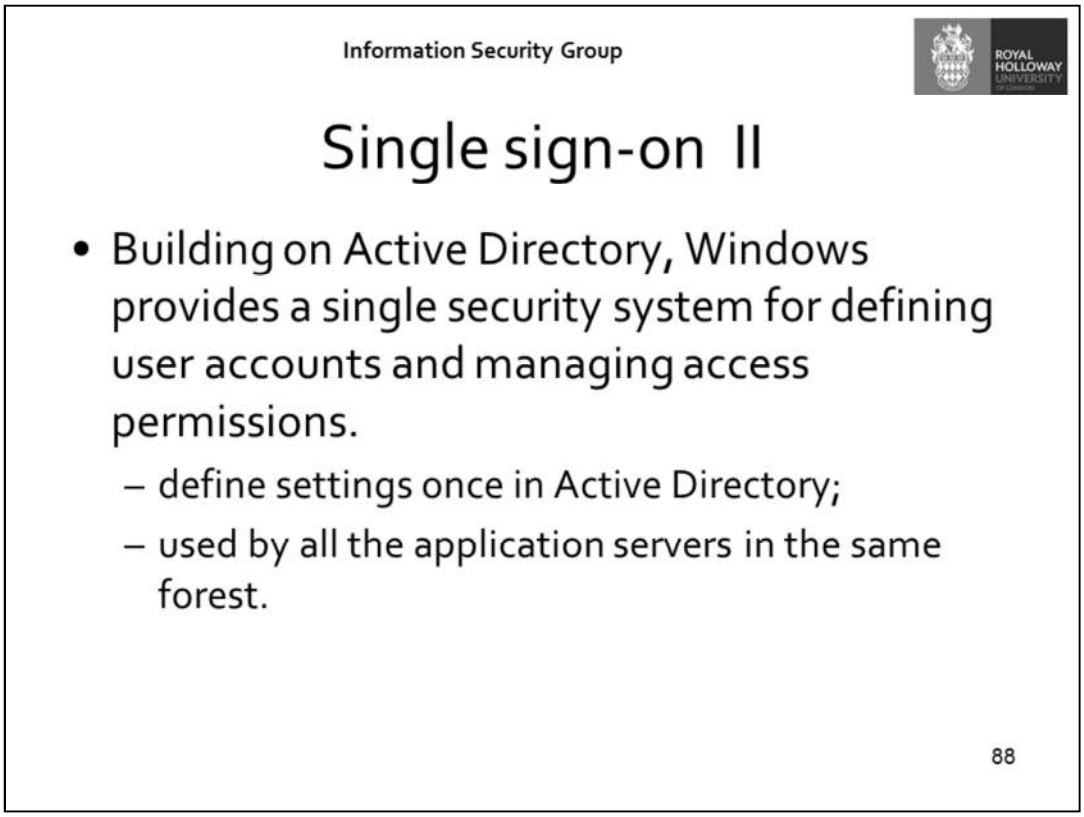

The operation of Kerberos in Windows can be summarised as follows.

1. The Client receives the username and password from the user, and sends the pre-authentication data to the Authentication Service.
2. The Authentication service returns a ticket-granting ticket (TGT).
3. The Client requests a session ticket (for a specific network service).
4. The Ticket-granting server (TGS) returns a session ticket for the specified network service.
5. The Client sends the session ticket to the network service.



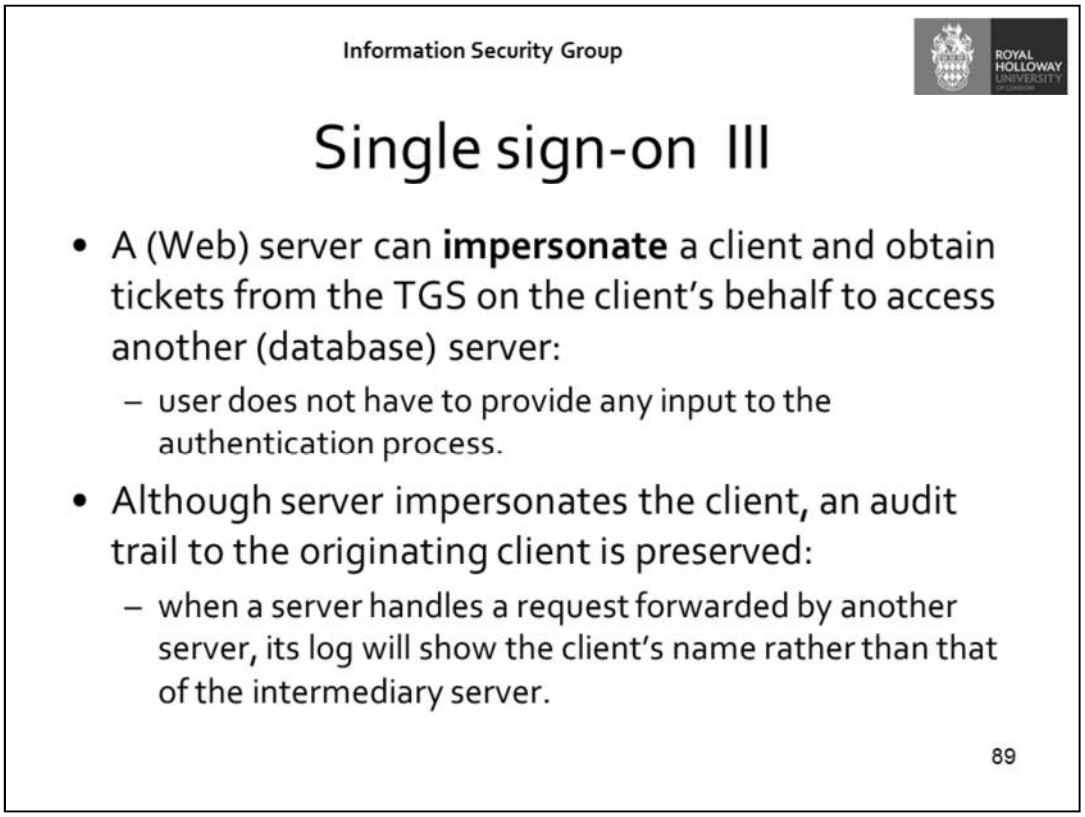

It is becoming increasingly common for applications to use several servers to perform a task. For example, a Web-based application might use both a Web server and a database server to provide information to a browser-based client. These servers might be in different domains.

Every domain in a Windows forest trusts every other domain in that forest. Windows uses this to extend single sign-on beyond the boundary of the domain to which a user logs on. A client only needs to authenticate once, rather than re-authenticating to servers in different domains.




Building on Active Directory, Windows provides a single security model and infrastructure for defining user accounts and for managing access permissions. As a result it is only necessary to define settings once in the Active Directory.

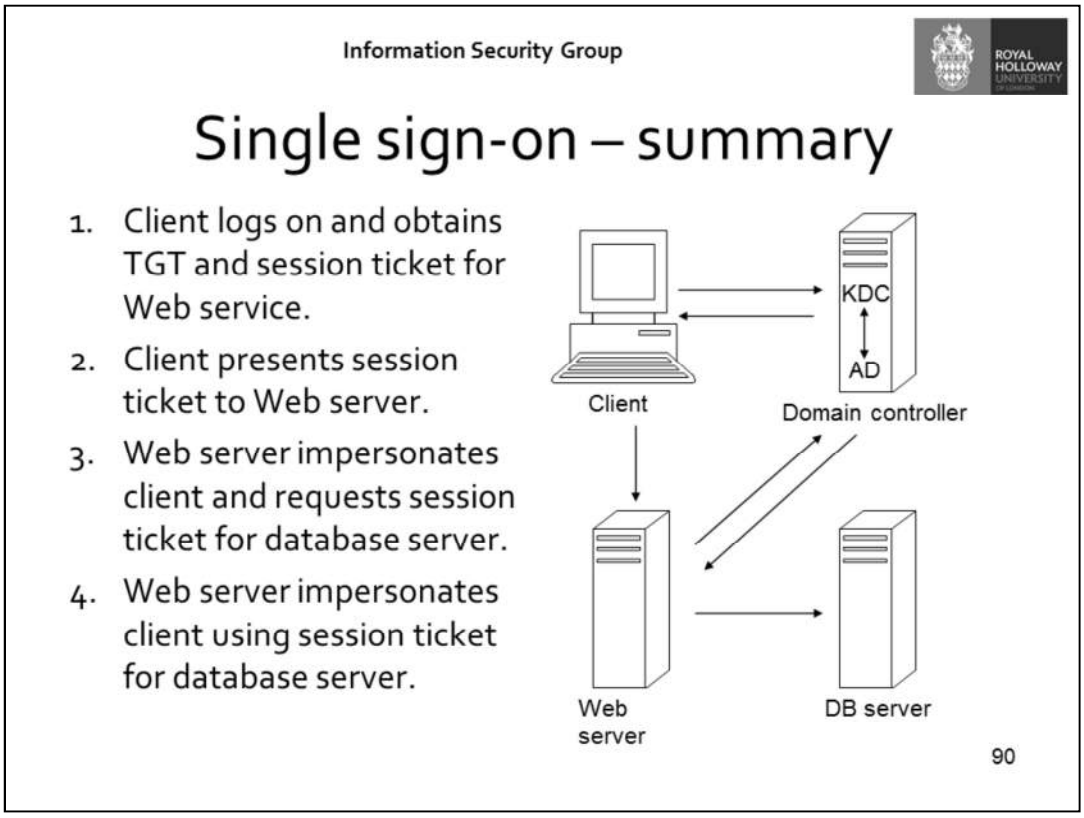
Active Directory is used by all the application servers in the same forest.




To enable a type of delegation, a (Web) server can **impersonate** a client and obtain tickets from the TGS on the client's behalf to access another (database) server. The user does not have to provide any input to the authentication process.

Although the server 'impersonates' the client, an audit trail to the originating client is preserved:

• when a server handles a request forwarded by another server, its log will show the client's name rather than that of the intermediary server.




The single sign-on process can be summarised as follows.

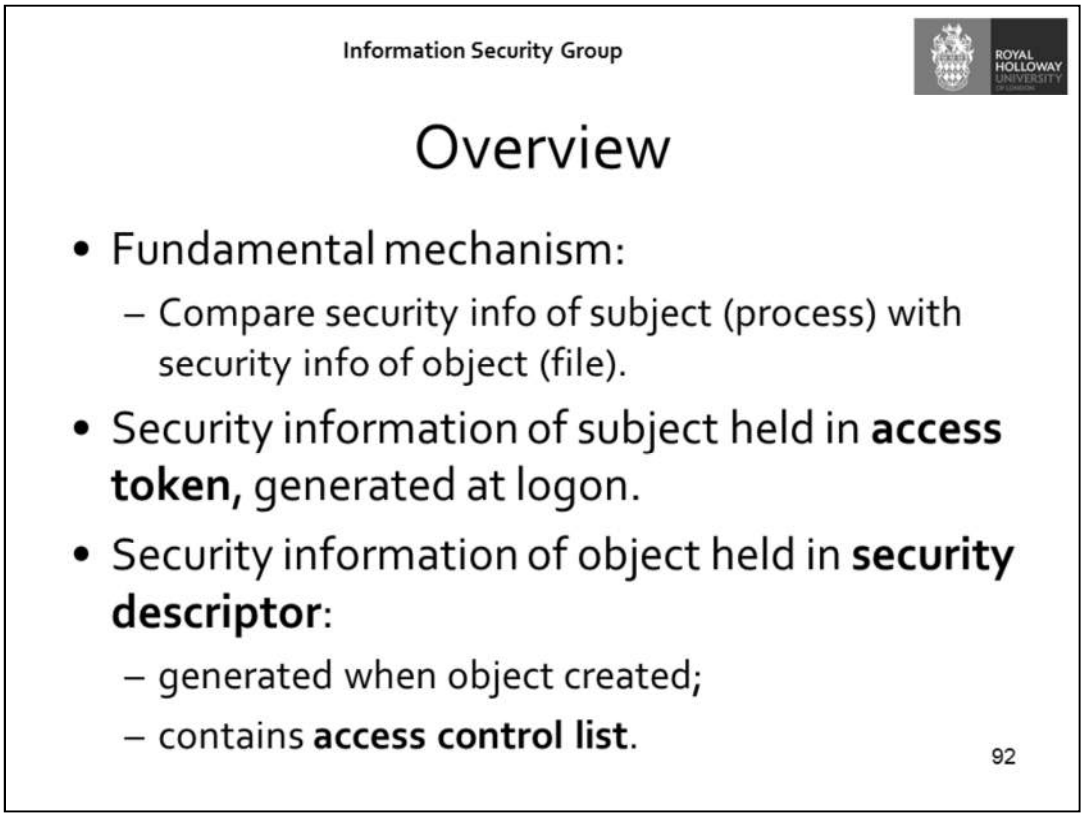
1. The Client logs on and obtains a TGT and a session ticket for the Web service.
2. The Client presents the session ticket to the Web server.
3. The Web server impersonates the client and requests a session ticket for the database server.
4. The Web server impersonates the client using the session ticket for the database server.

Information Security Group


# Agenda

- Introduction
- Active directory
- Domains
- Windows subjects and objects
- Access rights
- Authentication
- Access control
- Security management
- Resources

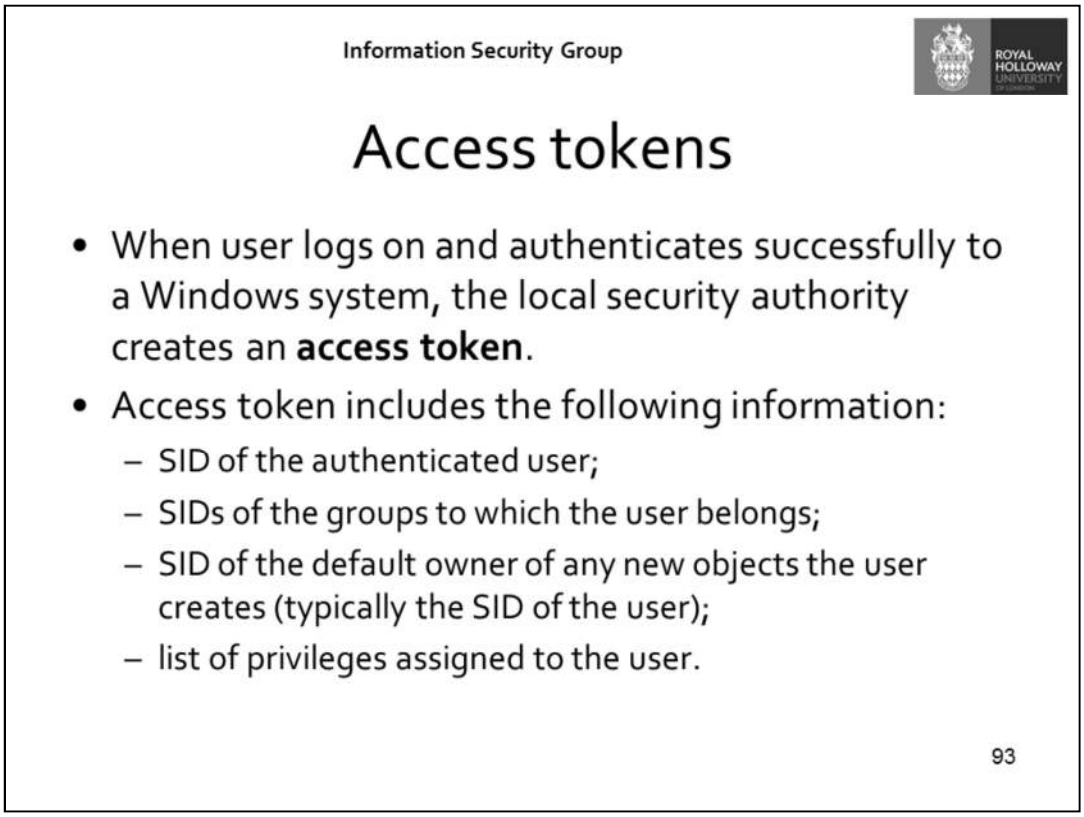
91




The fundamental mechanism used by the Windows **Security Reference Monitor** is to compare the security information of the subject (process) with the security information of object (file).

The security information of the subject is held in an **access token**, which is generated at logon.

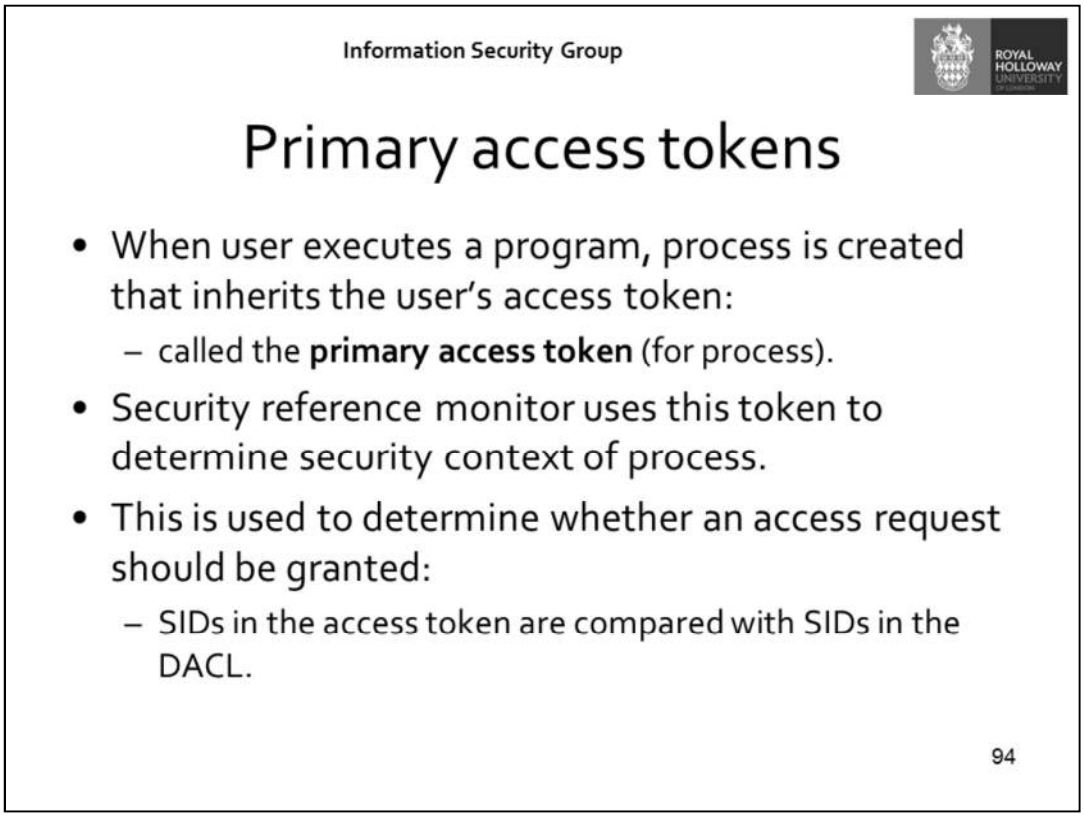
The security information of the object is held in the **security descriptor**, which is generated when the object is created. The security descriptor contains an **access control list** (the DACL).




When a user logs on and authenticates successfully to a Windows system, the local security authority creates an **access token**.

An access token includes the following information:

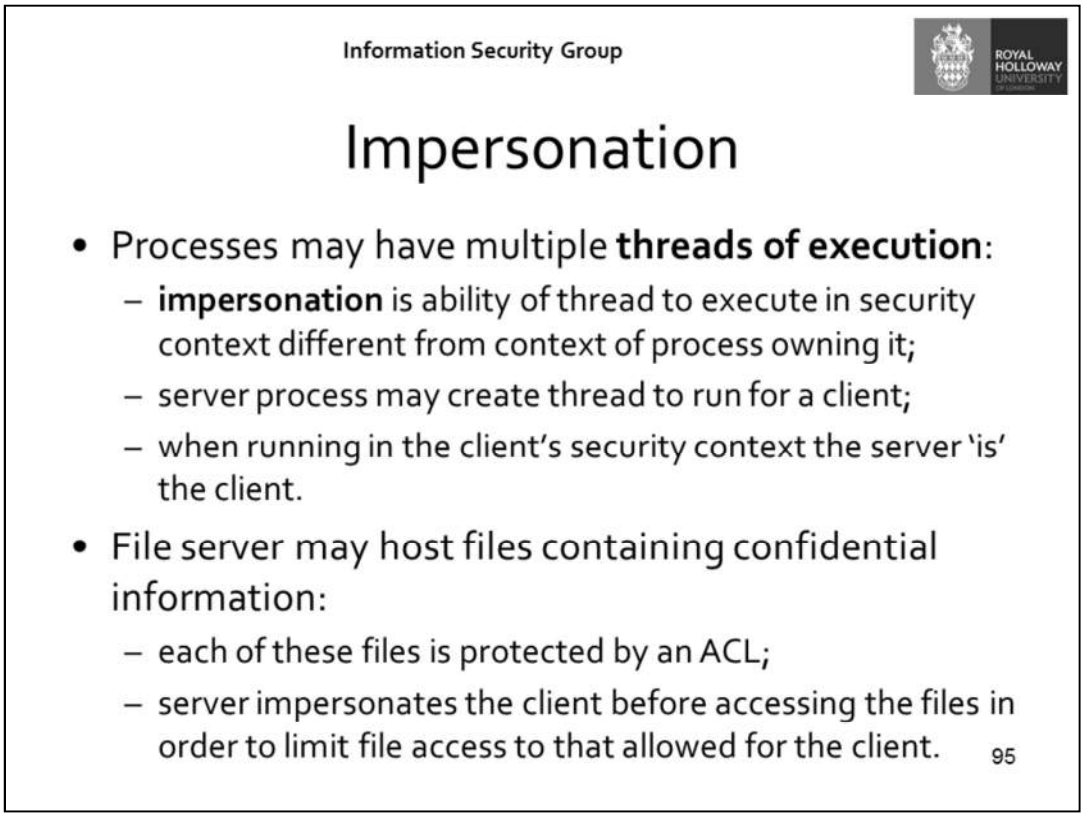
- the SID of the authenticated user;
- the SIDs of the groups to which the user belongs;
- the SID of the default owner of any new objects the user creates (typically the SID of the user);
- a list of privileges assigned to the user.




When a user executes a program, a process is created that inherits a copy of the user's access token. This is called the **primary access token** (for that process).

The security reference monitor uses this token to determine the security context of the process and determine whether an access request should be granted:

• SIDs in the access token are compared with SIDs in the DACL of the object to which access is requested.




Processes can run multiple **threads of execution**:

• **impersonation** is the ability of a thread to execute in a security context that is different from the context of the process that owns the thread;

• a server process may create a thread to run on behalf of a client;

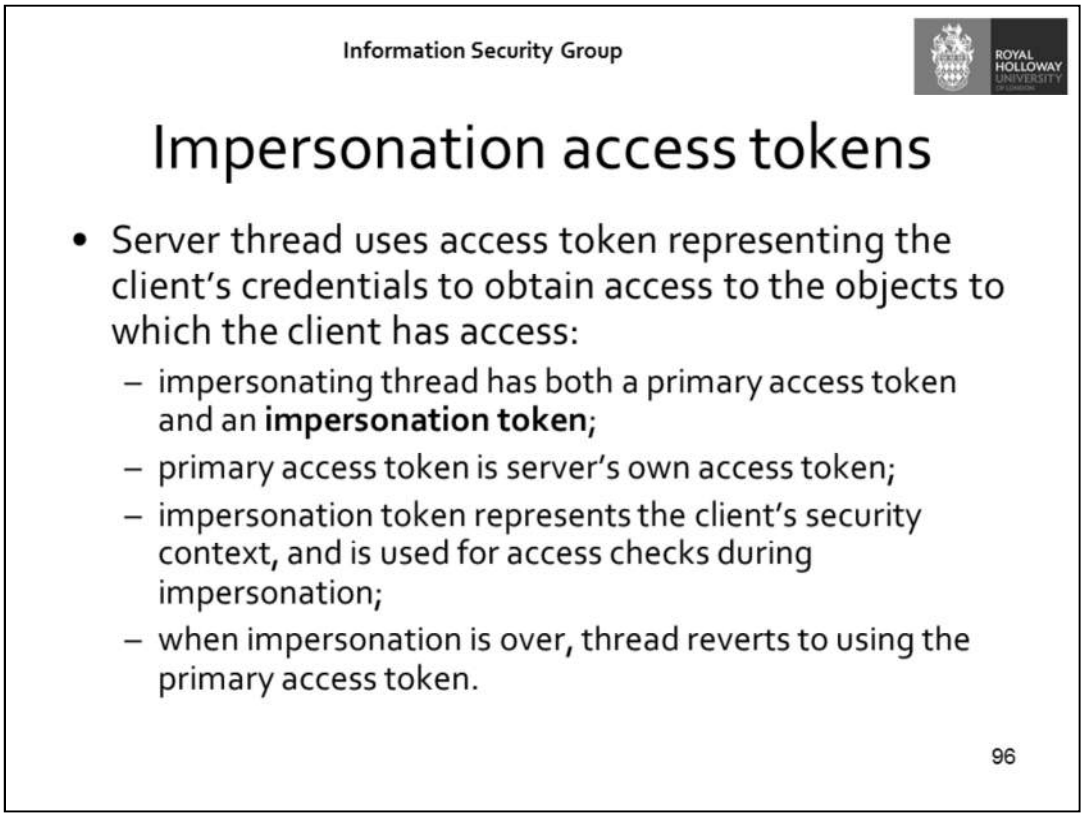
• when running in the client's security context the server 'is' the client.

A file server may host files containing confidential information:

• each of these files is protected by an ACL;

• the server impersonates the client before accessing the files in order to limit file access to that allowed for the client.

This is the approach used by Windows to support the notion of **delegation**.




The server thread uses an access token representing the client's credentials to obtain access to the objects to which the client has access:

• the impersonating thread has both a primary access token and an **impersonation token**;

• the primary access token is a copy of the server's own access token;

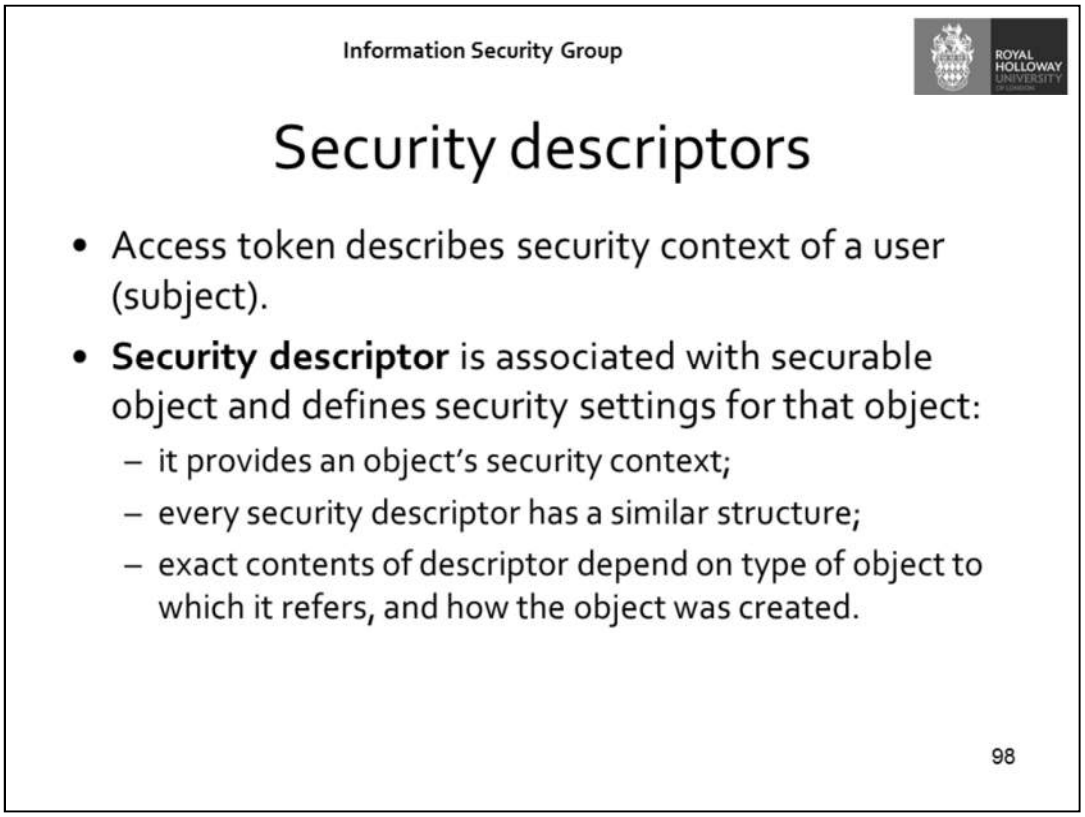
• the impersonation token represents the client's security context, and is used for access checks during impersonation;

• when impersonation is over, the thread reverts to using the primary access token.


An impersonation token can both expand or limit the ability of an impersonating thread to access securable objects and to perform privileged operations.

An impersonation token is a copy of the client process's (primary) access token but can be restricted by:

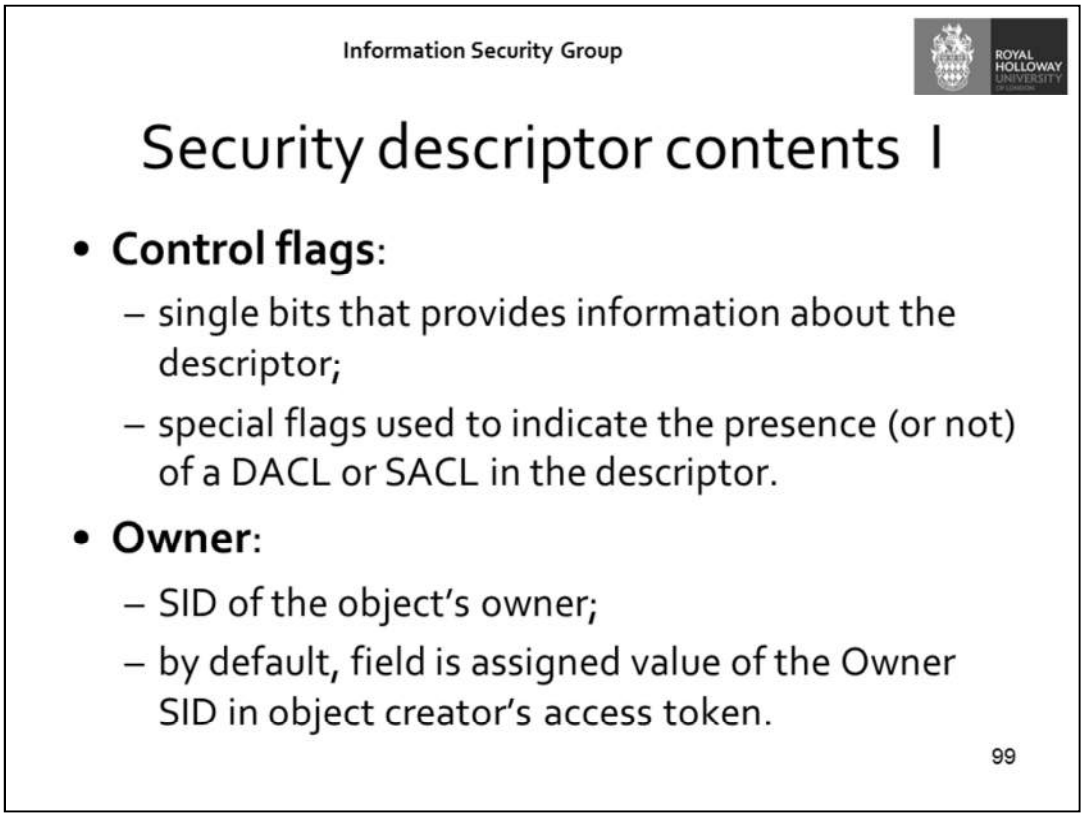
- removing privileges from the client process's primary token;
- applying the deny-only attribute to SIDs in the token so that such SIDs cannot be used to access secured objects.




An access token describes the security context of a user (subject).

A **security descriptor** is associated with a securable object and defines security settings for that object:

- it provides an object's security context;
- every security descriptor has a similar structure;
- the exact contents of the descriptor depend on the type of object to which it refers, and how the object was created.




**Control flags**:

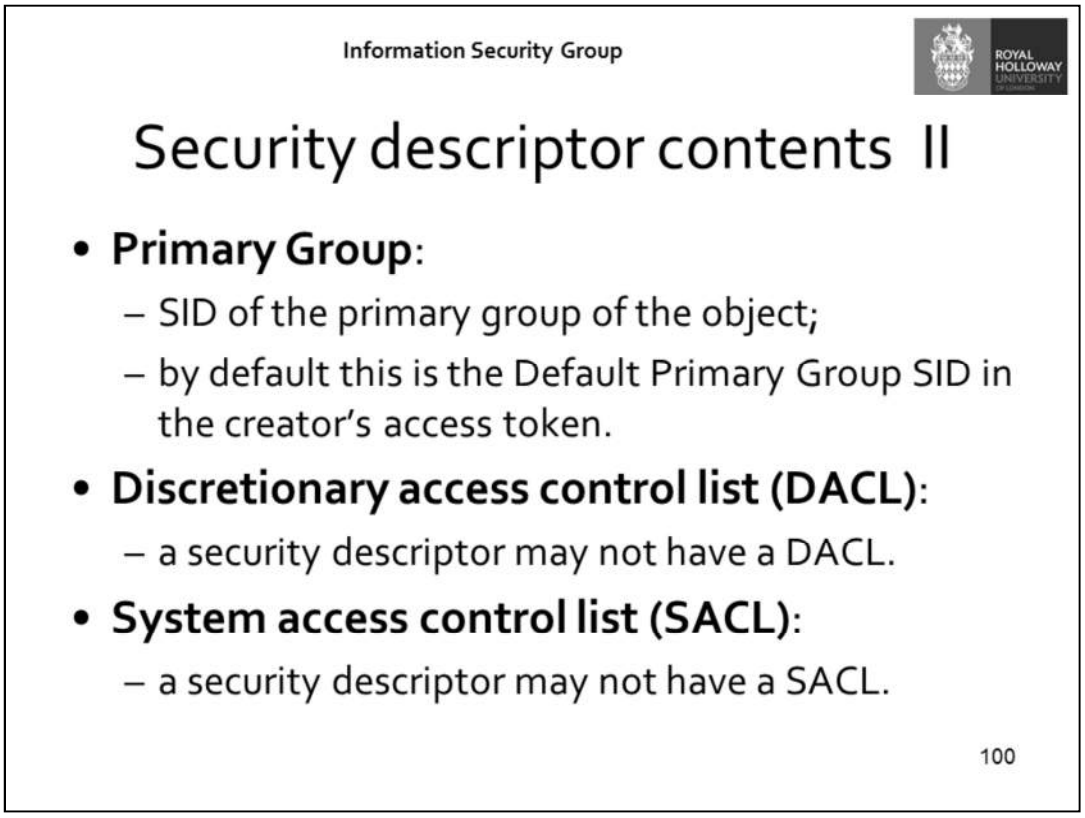
• each flag is a single bit that provides information about the descriptor;

• these 'special' flags are used to indicate the presence (or not) of a DACL or SACL in the descriptor.

**Owner**:

• the SID of the object's owner;

• by default, this field is assigned the value of the Owner SID in the object creator's access token.




**Primary Group**:

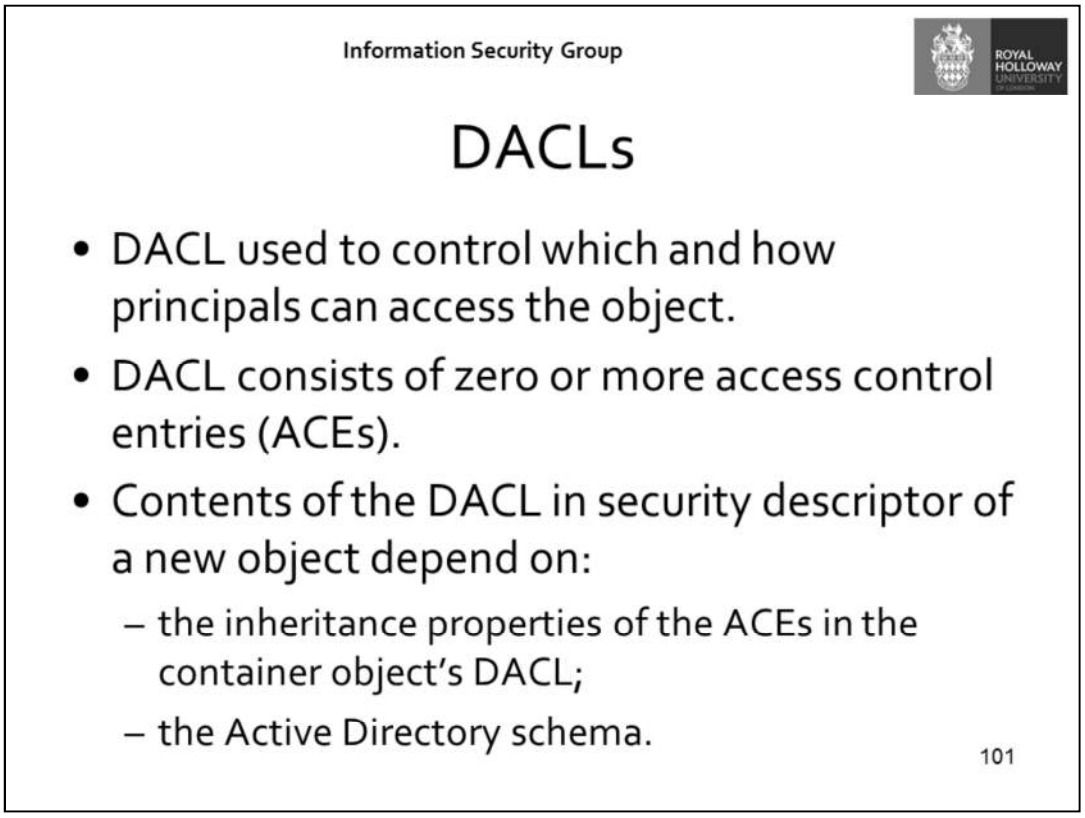
- the SID of the primary group of the object;
- by default this is the Default Primary Group SID in the creator's access token.

**Discretionary access control list (DACL)**:

- a security descriptor may not have a DACL.

**System access control list (SACL)**:

- a security descriptor may not have a SACL.

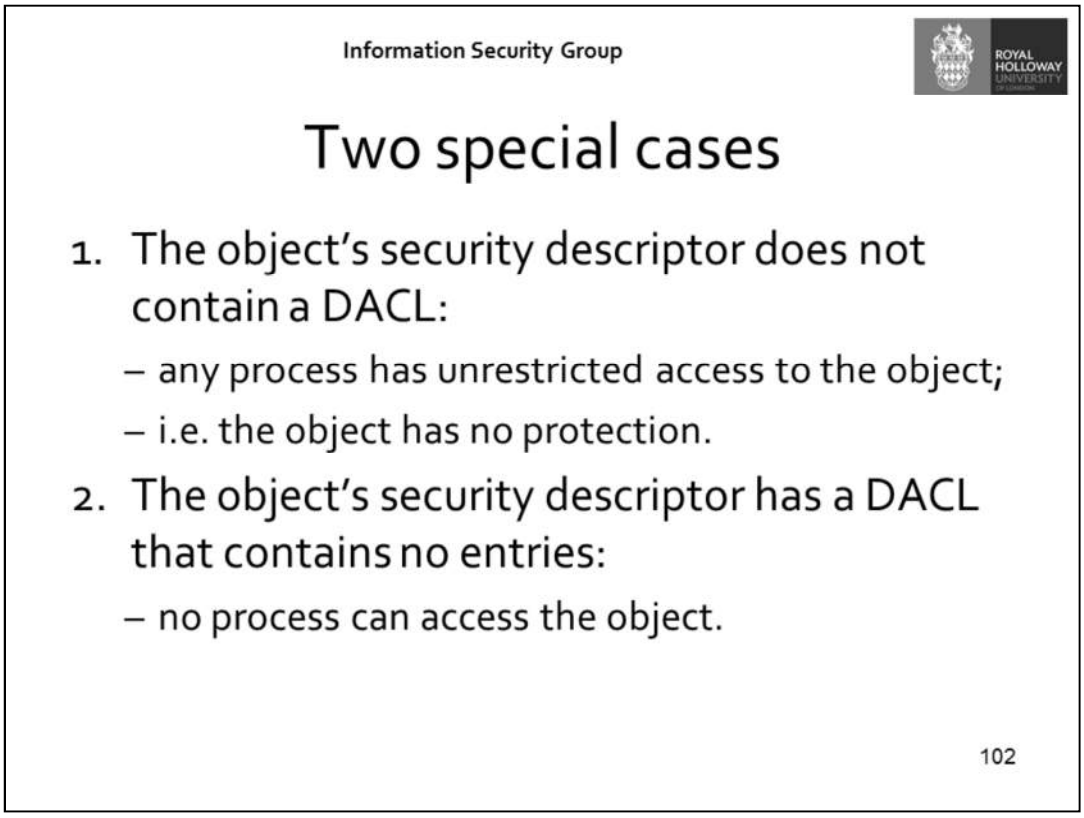



The DACL is used to control which principals can access the object and how such principals can access the object.

The DACL consists of zero or more access control entries (ACEs).

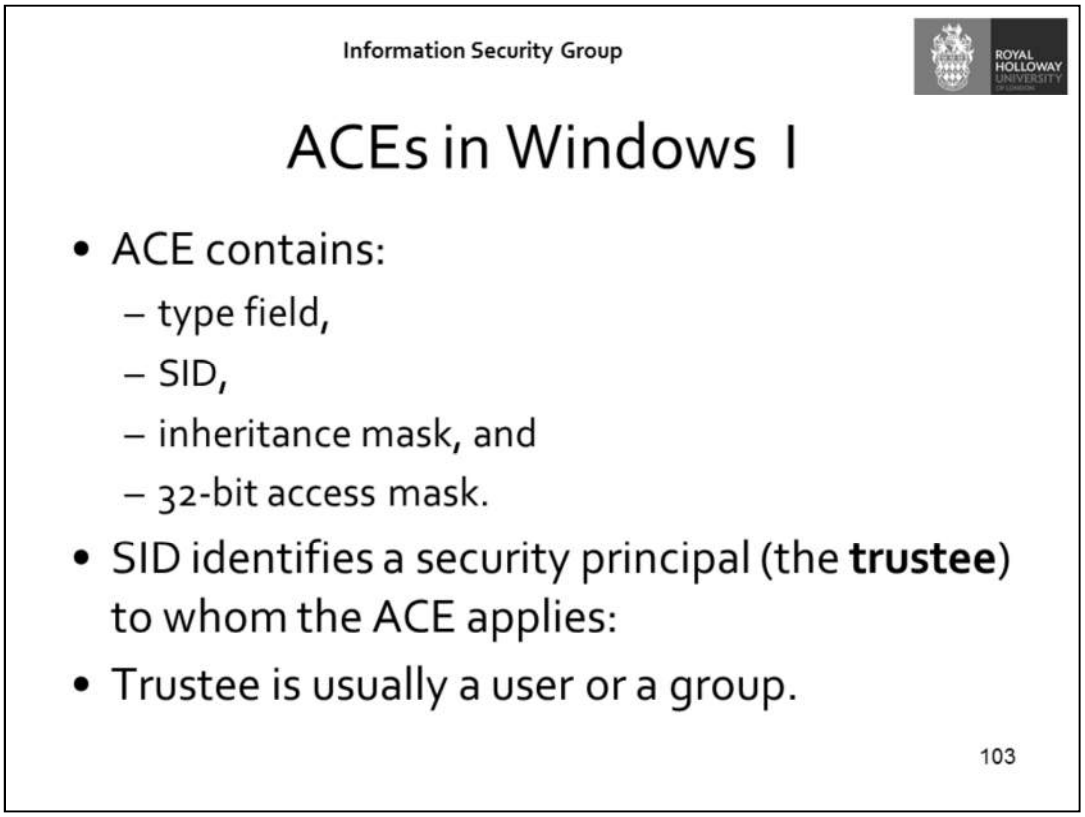
The contents of the DACL in the security descriptor of a new object depend on:

- the inheritance properties of the ACEs in the container object's DACL;
- the Active Directory schema.



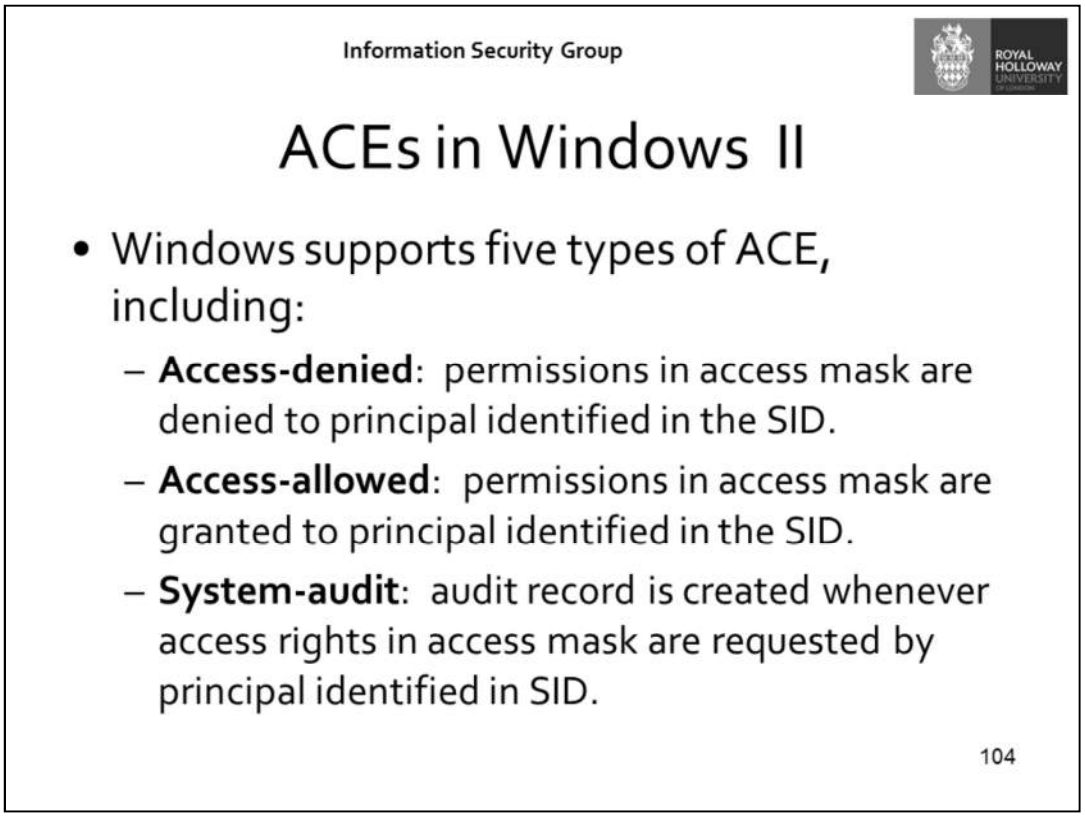

There are two special cases:

1. If the object's security descriptor does not contain a DACL, then every process has unrestricted access to the object, i.e. the object has no protection.
2. If the object's security descriptor has a DACL that contains no entries, then no process can access the object.




An ACE contains a type field, a SID, an inheritance mask and a 32-bit access mask.

The SID identifies a security principal (referred to as the **trustee**) to whom the ACE applies. The trustee is usually a user or a group.

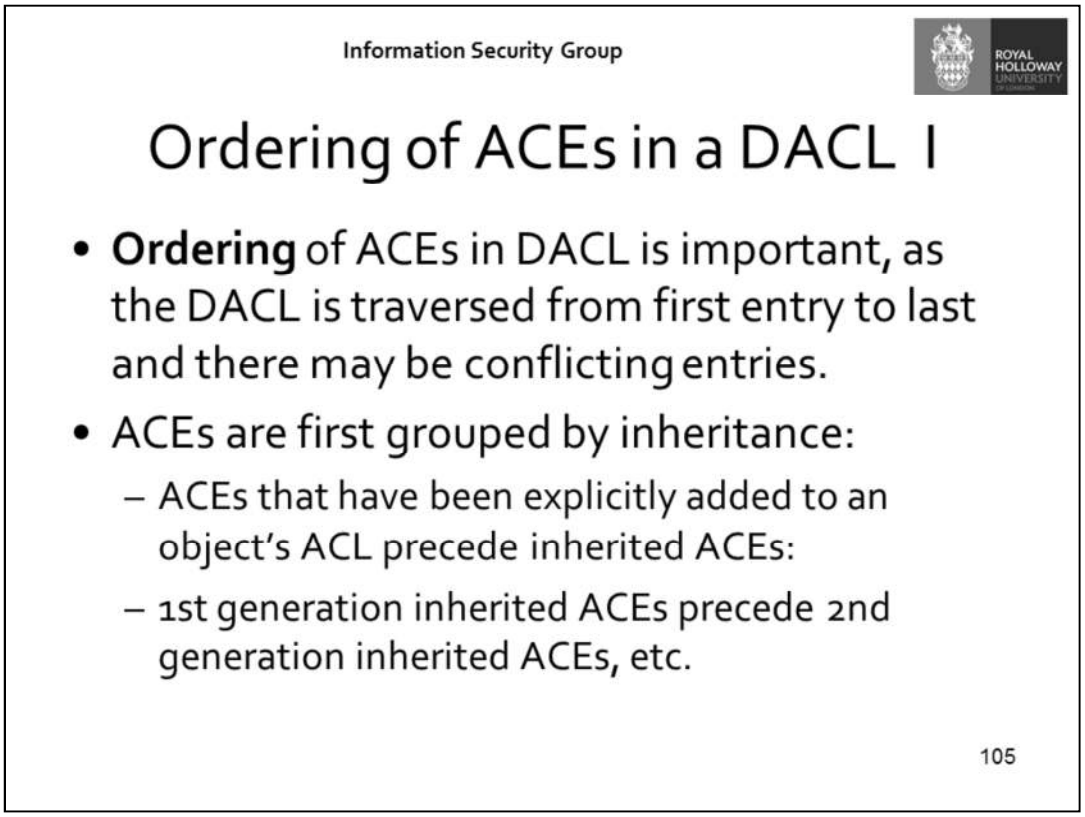



Windows supports five types of ACE, including:

- **Access-denied**:
  - permissions specified in the access mask are denied to the principal identified in the SID.
- **Access-allowed**:
  - permissions specified in the access mask are granted to the principal identified in the SID.
- **System-audit**:
  - an audit record is created whenever the access rights specified in the access mask are requested by the principal identified in the SID.

[Also supports 'system alarm' and 'access allowed compound' ACEs].

[In fact, Windows also supports object-specific ACEs used for Active Directory objects, but we do not cover these here. Such objects support fine grained access control to Active Directory objects].

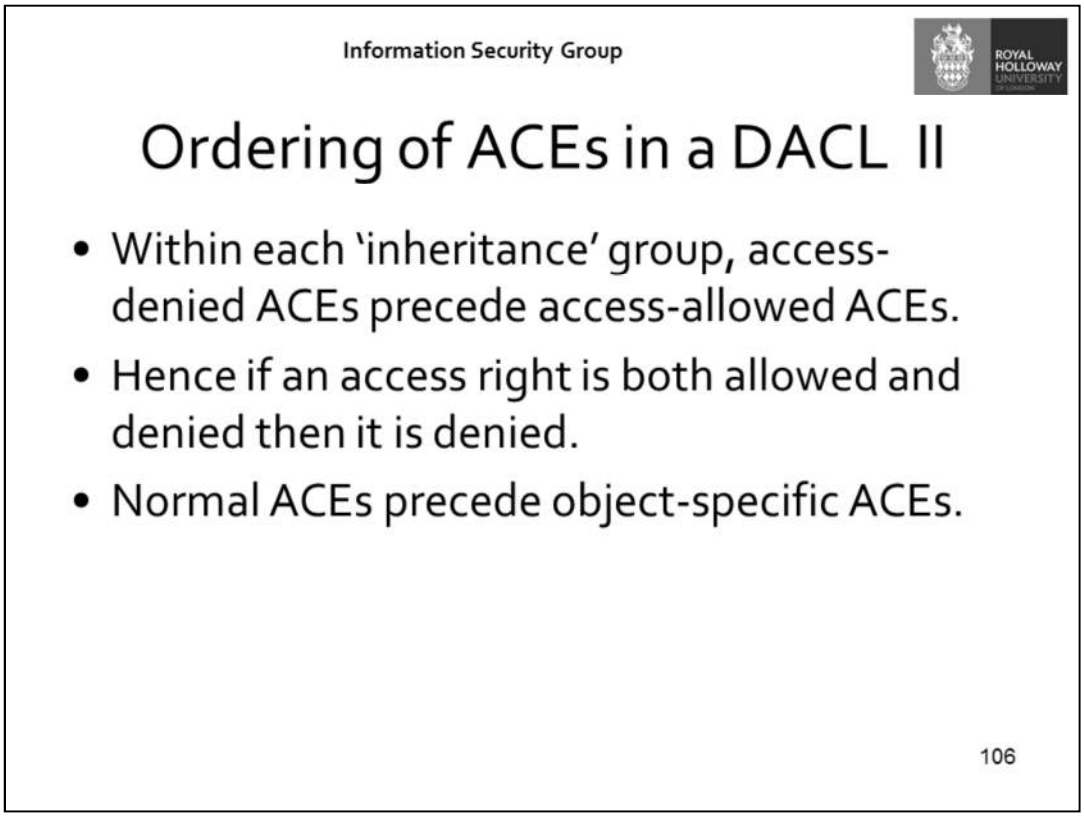



The **ordering** of ACEs in the DACL is important, as the DACL is traversed from first entry to last and there may be conflicting entries. Moreover, once enough ACEs have been examined to enable an access decision, the decision is made and all further ACEs are ignored.

ACEs are first grouped by inheritance:

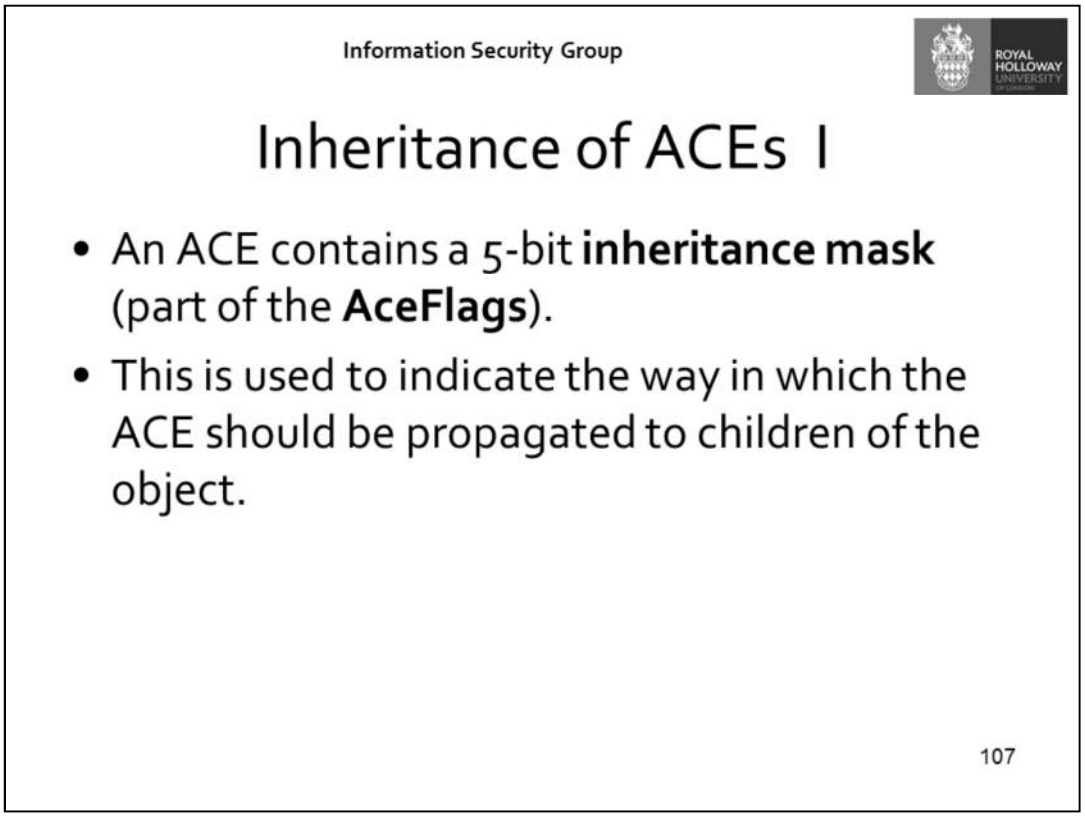
• ACEs that have been explicitly added to an object's ACL precede inherited ACEs, and hence the owner has the most control over access;

• 1st generation inherited ACEs precede 2nd generation inherited ACEs, etc.



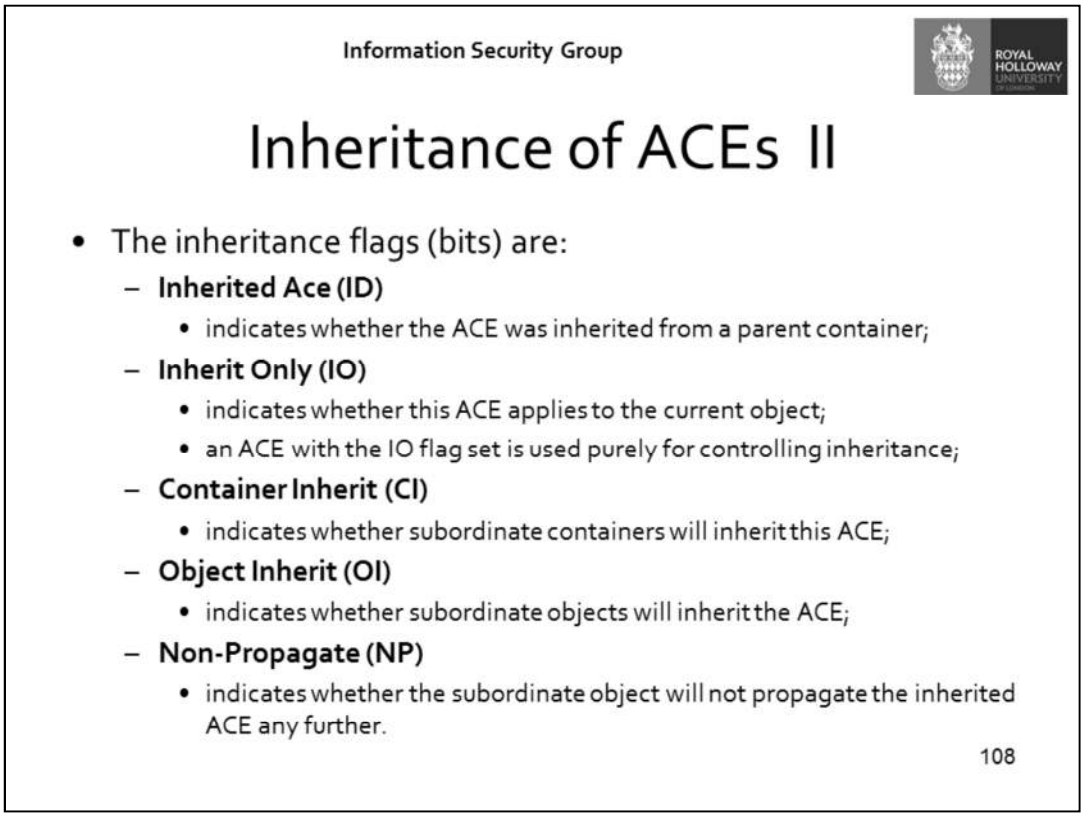

Within each 'inheritance' group, access-denied ACEs precede access-allowed ACEs. Hence if an access right is both allowed and denied then it is denied.

Normal ACEs precede object-specific ACEs.




An ACE contains a 5-bit **inheritance mask** (part of the **AceFlags**).

This is used to indicate the way in which the ACE should be propagated to children of the object.




The inheritance flags (bits) are:

**Inherited Ace (ID)**

indicates whether the ACE was inherited from a parent container;

**Inherit Only (IO)**

indicates whether this ACE applies to the current object;

an ACE with the IO flag set is used purely for controlling inheritance;

**Container Inherit (CI)**

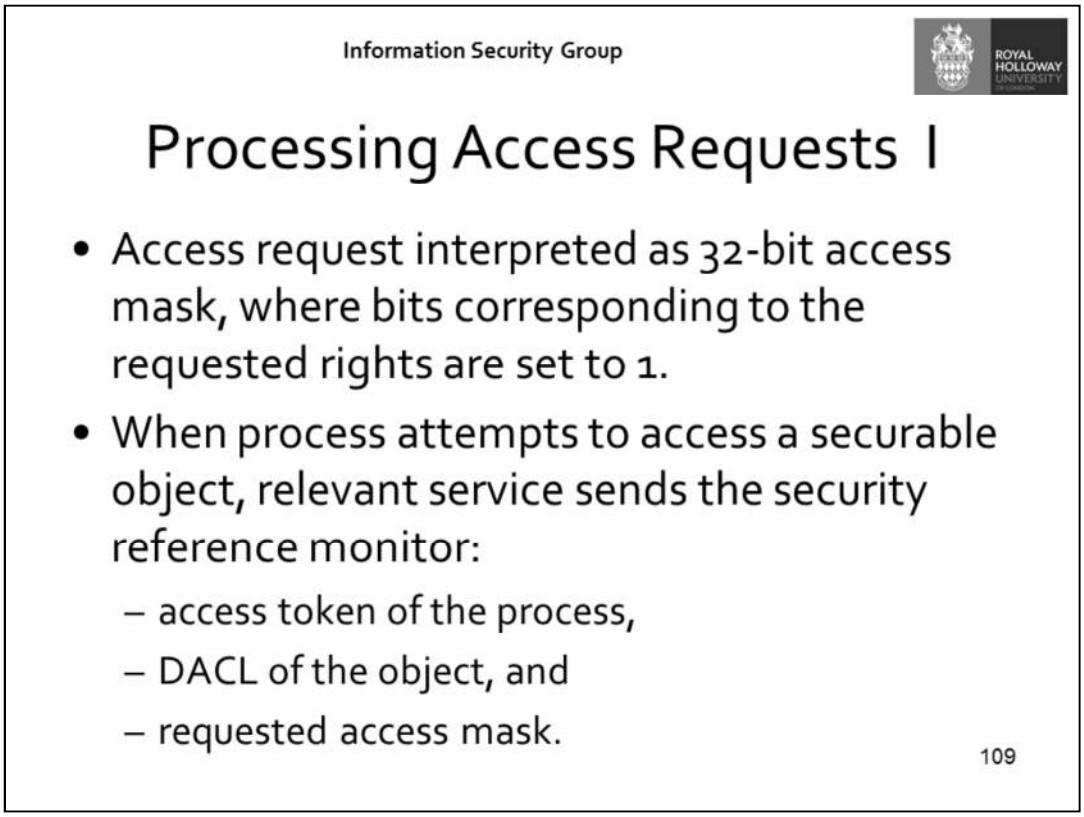
indicates whether subordinate containers will inherit this ACE;

**Object Inherit (OI)**

indicates whether subordinate objects will inherit the ACE;

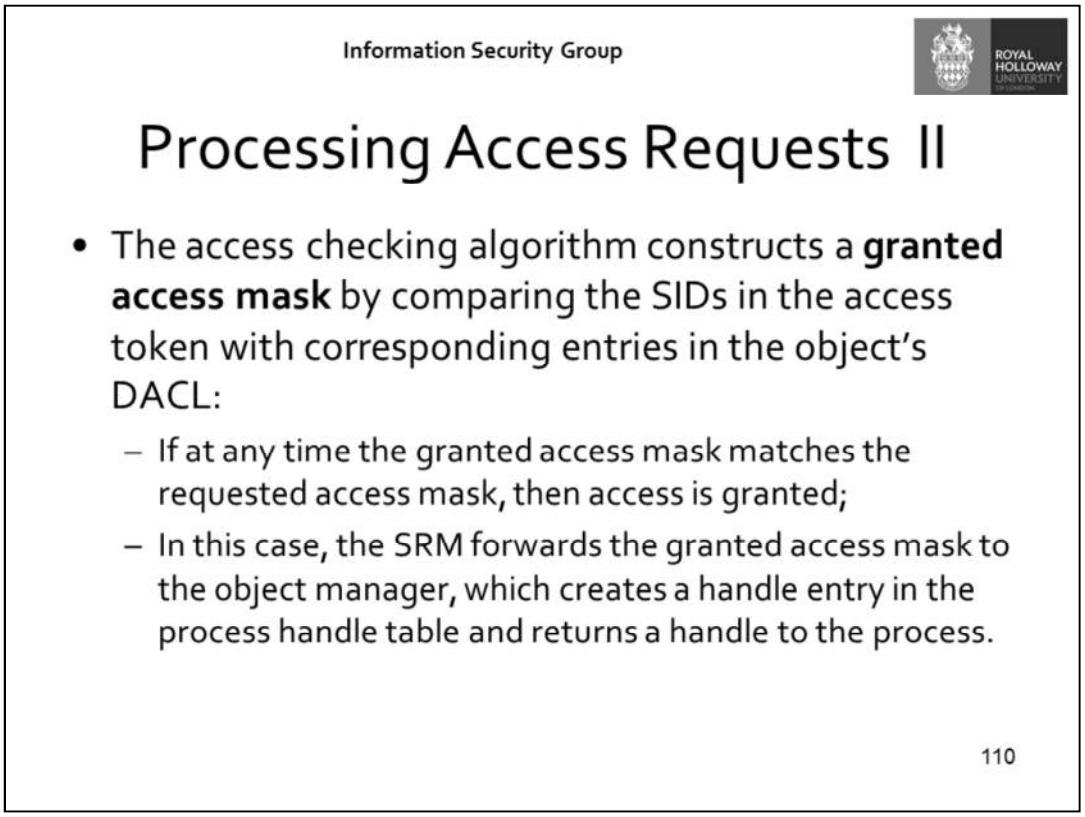
**Non-Propagate (NP)**

indicates whether the subordinate object will not propagate the inherited ACE any further.




An access request for a specific set of access rights is interpreted as a 32-bit access mask, where the bits corresponding to the requested rights are set to 1 (and all other bits are set to 0).

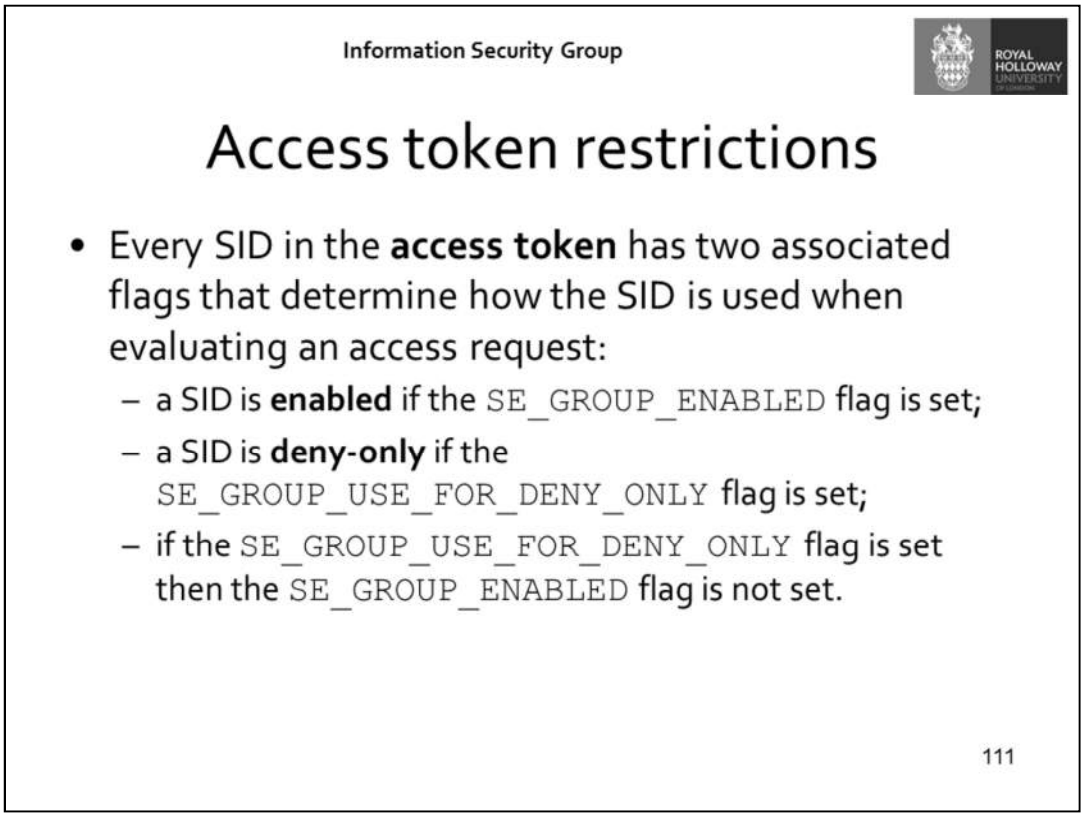
When a process attempts to access a securable object, the relevant service sends the access token of the process, the DACL of the object, and the requested access mask to the security reference monitor.




The SRM performs an access checking algorithm that constructs a granted access mask by comparing the SIDs in the access token with corresponding entries in the object's DACL:

• if at any stage the granted access mask matches the requested access mask, then access is granted;

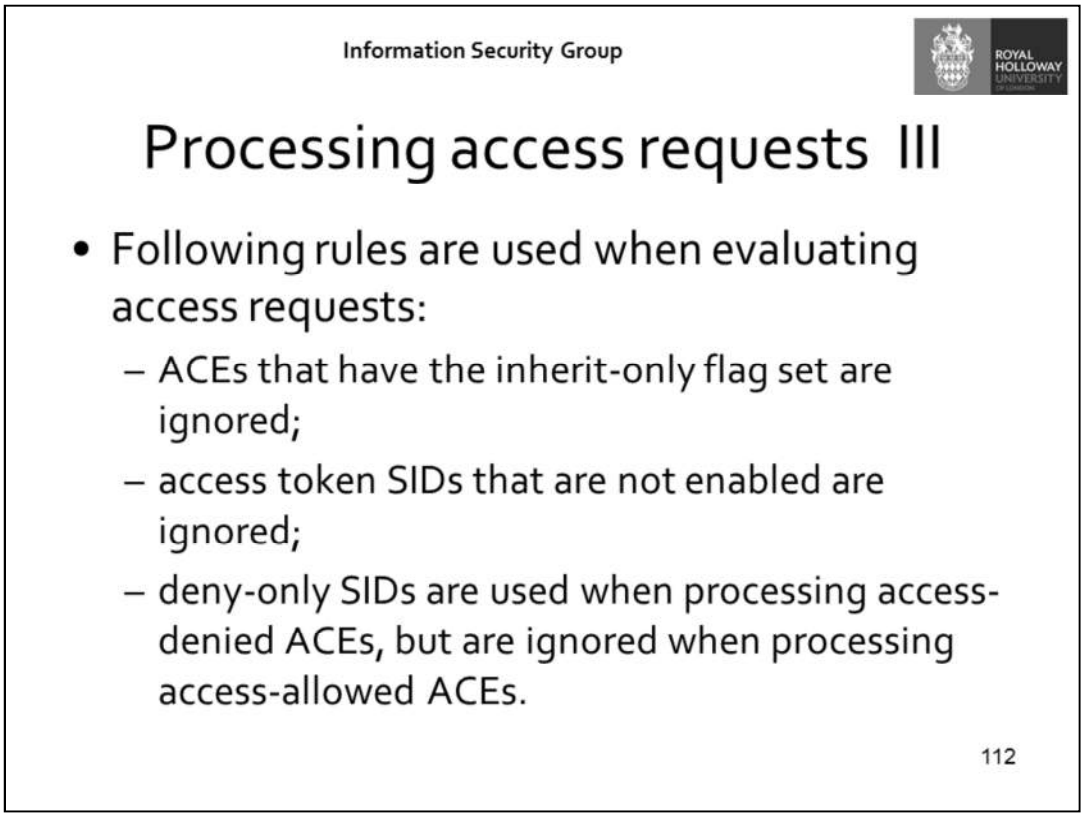
• in this case, the SRM forwards the granted access mask to the object manager, which creates a handle entry in the process handle table and returns a handle to the process.




Every SID in an access token has two associated flags that determine how the SID is used when evaluating an access request:

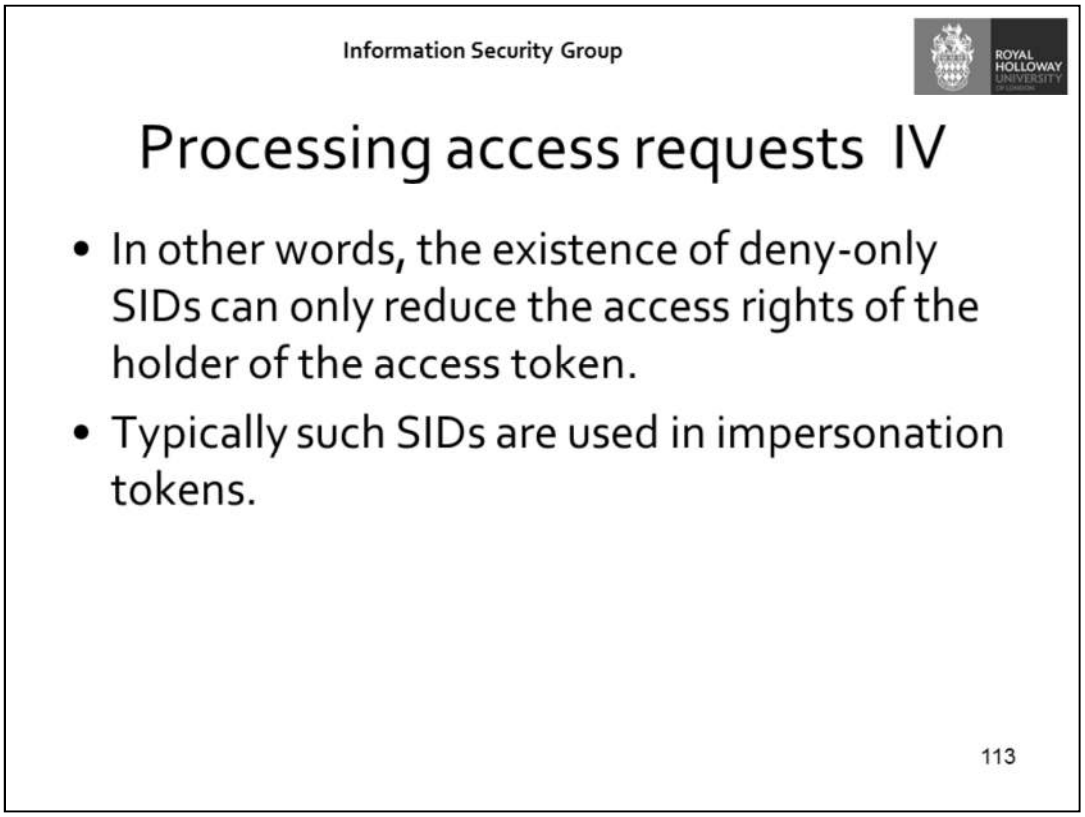
- a SID is **enabled** if the `SE_GROUP_ENABLED` flag is set;
- a SID is **deny-only** if the `SE_GROUP_USE_FOR_DENY_ONLY` flag is set;
- if the `SE_GROUP_USE_FOR_DENY_ONLY` flag is set then the `SE_GROUP_ENABLED` flag is not set.

The deny-only flag means that the SID cannot be used to gain positive access rights.



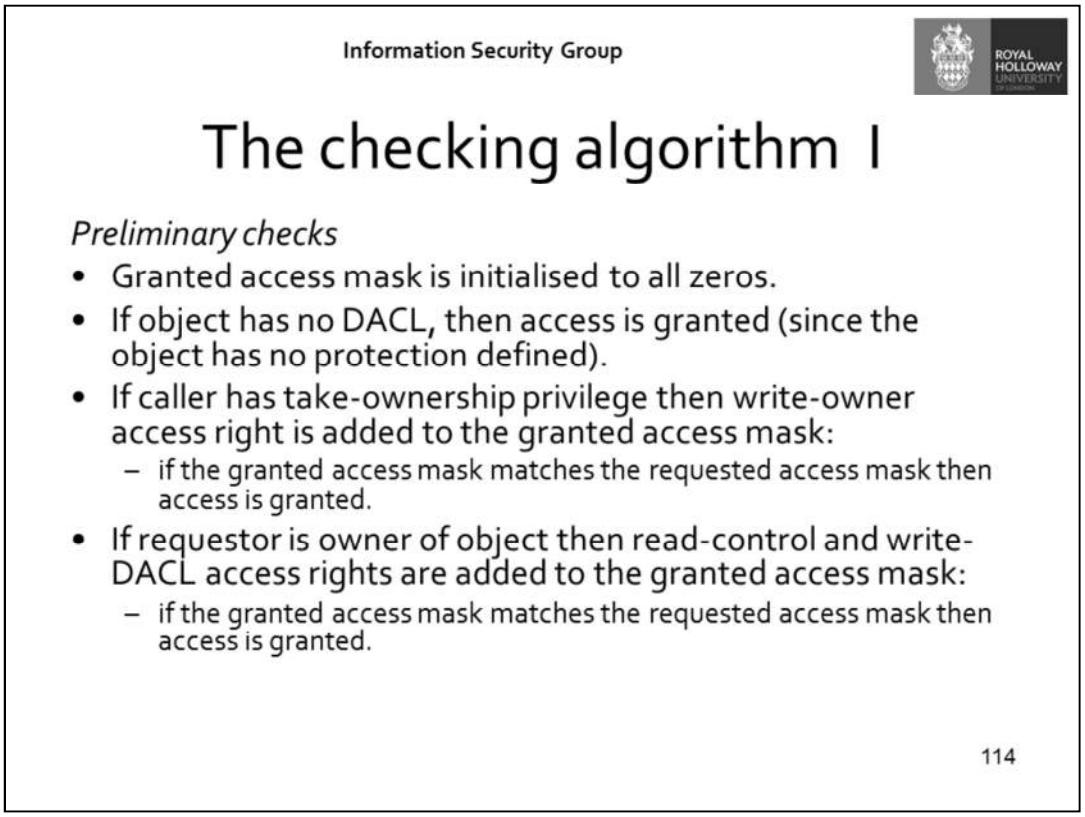

The following rules are used when evaluating an access request against a DACL (note this is a simplified version of the process):

- ACEs that have the inherit-only flag set are ignored;
- access token SIDs that are not enabled are ignored;
- deny-only SIDs are used when processing access-denied ACEs, but are ignored when processing access-allowed ACEs.




In other words, the existence of deny-only SIDs can only reduce the access rights of the holder of the access token.

Typically such SIDs are used in impersonation tokens.




Before starting, the granted access mask is set to all zeros. (Adding rights to the granted access mask involves setting the relevant bit(s) to one).

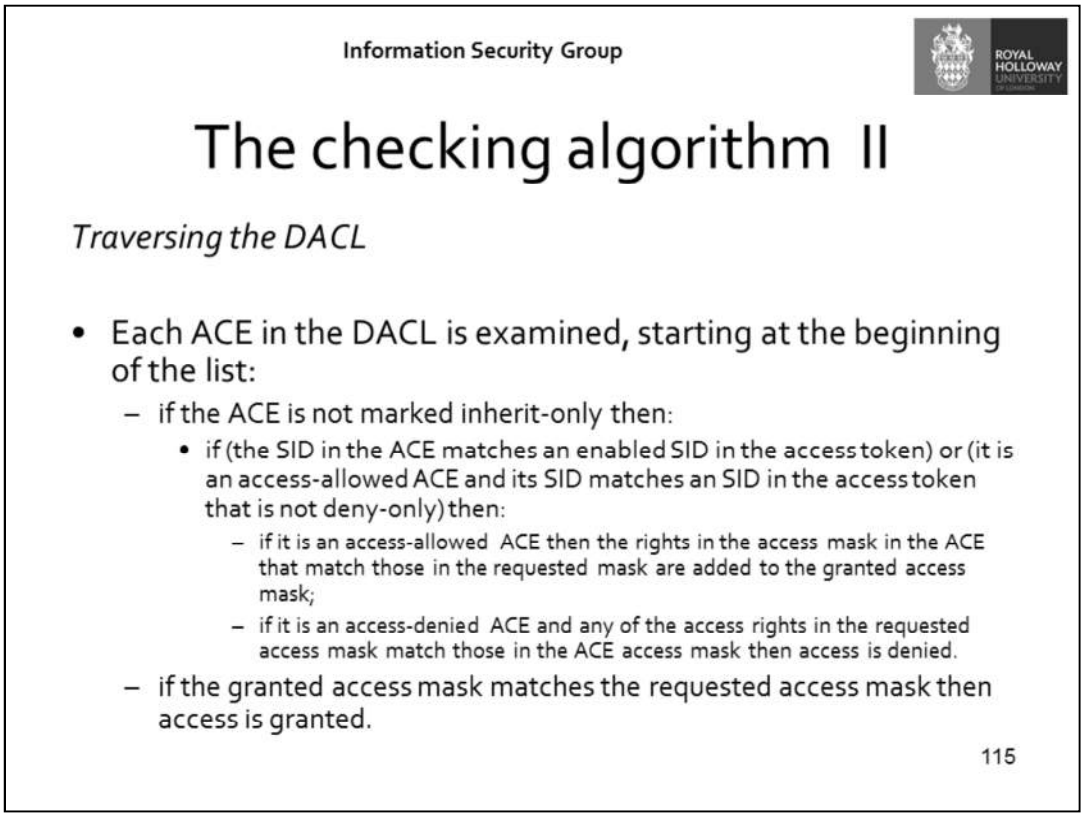
If the object has no DACL, then access is granted (since the object has no protection defined).

If the caller has the take-ownership privilege then the write-owner access right is added to the granted access mask:

• if the granted access mask matches the requested access mask then access is granted.

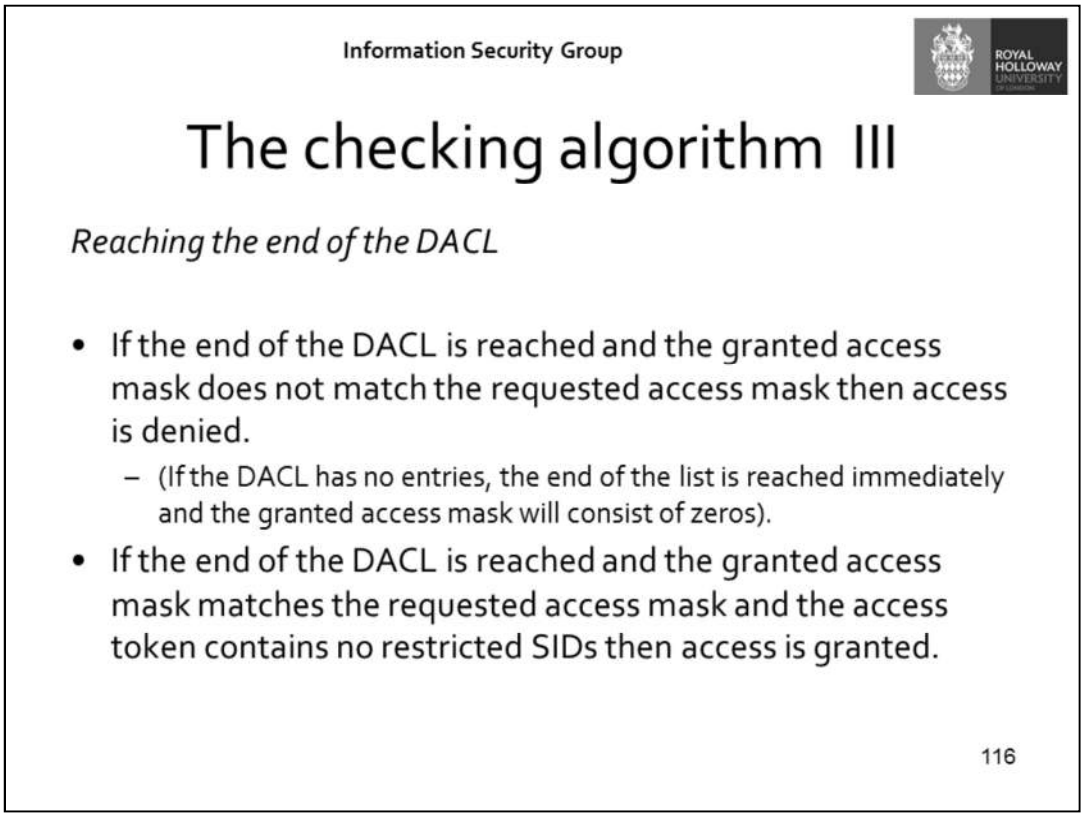
If the requestor is the owner of the object then the read-control and write-DACL access rights are added to the granted access mask:

• if the granted access mask matches the requested access mask then access is granted.




Each ACE in the DACL is examined, starting at the beginning of the list:

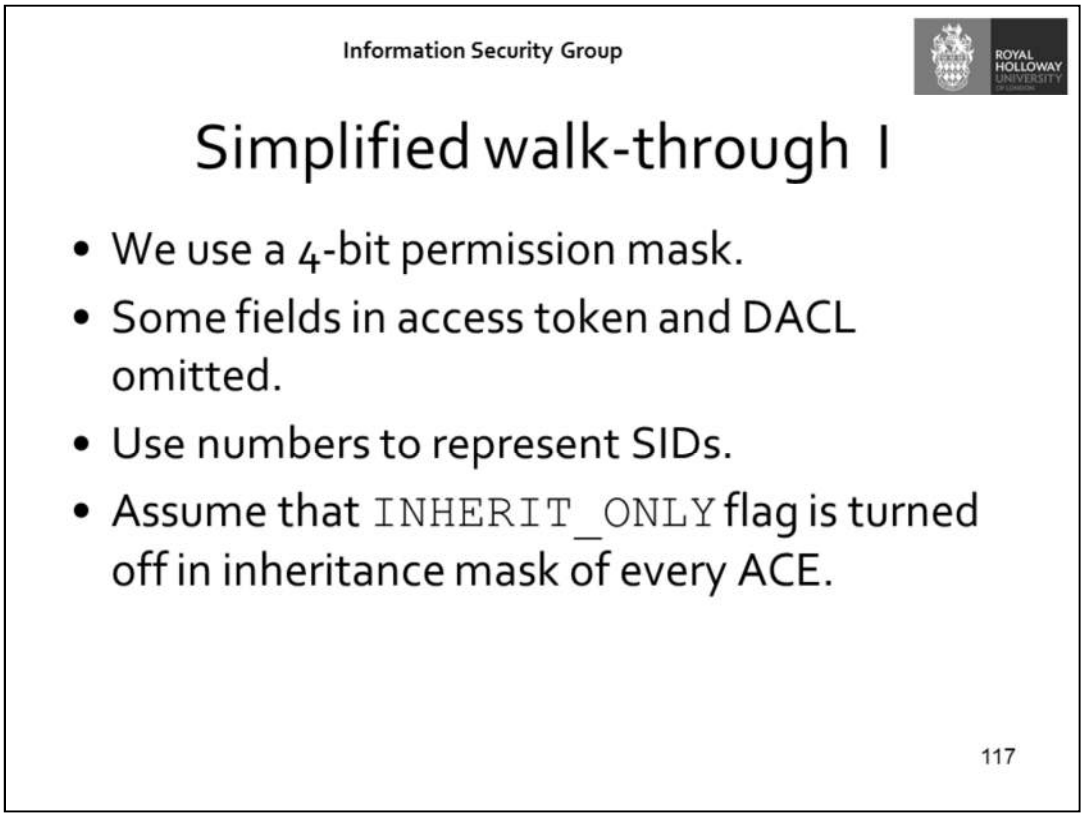
- if the ACE is not marked inherit-only then:
    - if (the SID in the ACE matches an enabled SID in the access token) or (it is an access-allowed ACE and its SID matches an SID in the access token that is not deny-only) then:
        - if it is an access-allowed ACE then the rights in the access mask in the ACE that match those in the requested mask are added to the granted access mask;
        - if it is an access-denied ACE and any of the access rights in the requested access mask match those in the ACE access mask then access is denied.
- if the granted access mask matches the requested access mask then access is granted.




If the end of the DACL is reached and the granted access mask does not match the requested access mask then access is denied.

• (If the DACL has no entries, the end of the list is reached immediately and the granted access mask will consist of zeros).

If the end of the DACL is reached and the granted access mask matches the requested access mask and the access token contains no restricted SIDs then access is granted.

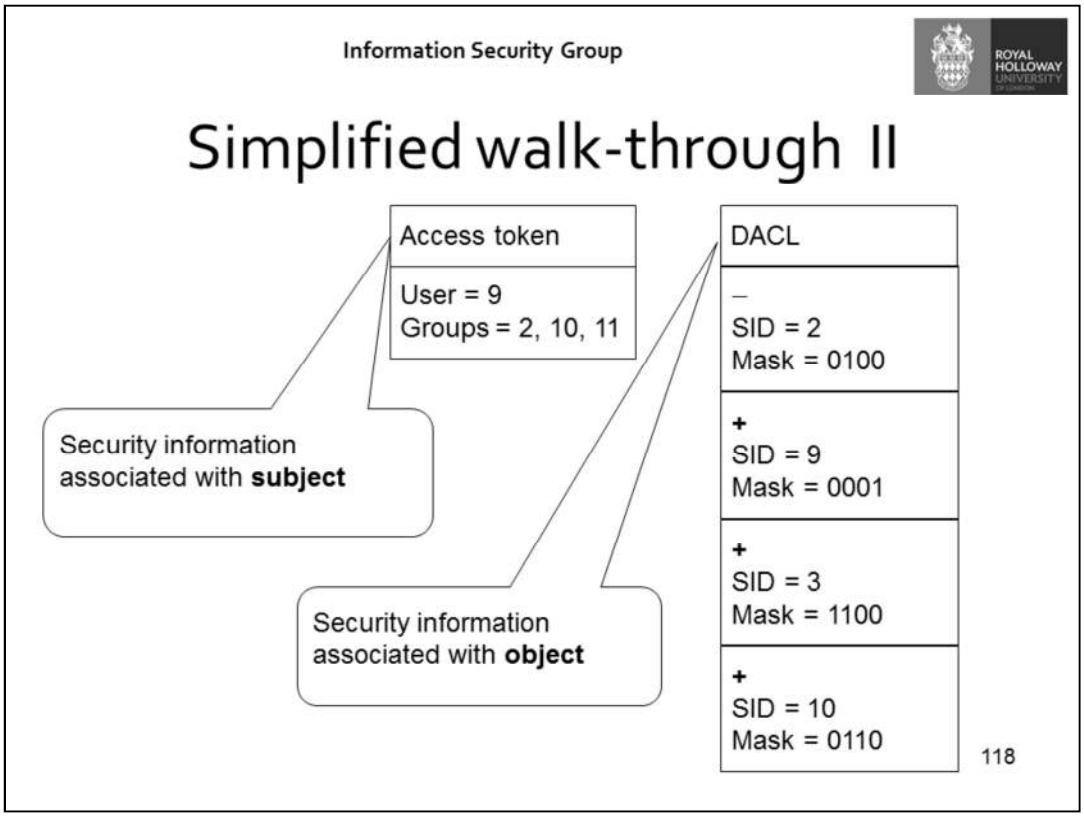



In our simplified example we use a 4-bit permission mask to make the description simpler.

Some of the fields in the access token and DACL are omitted.

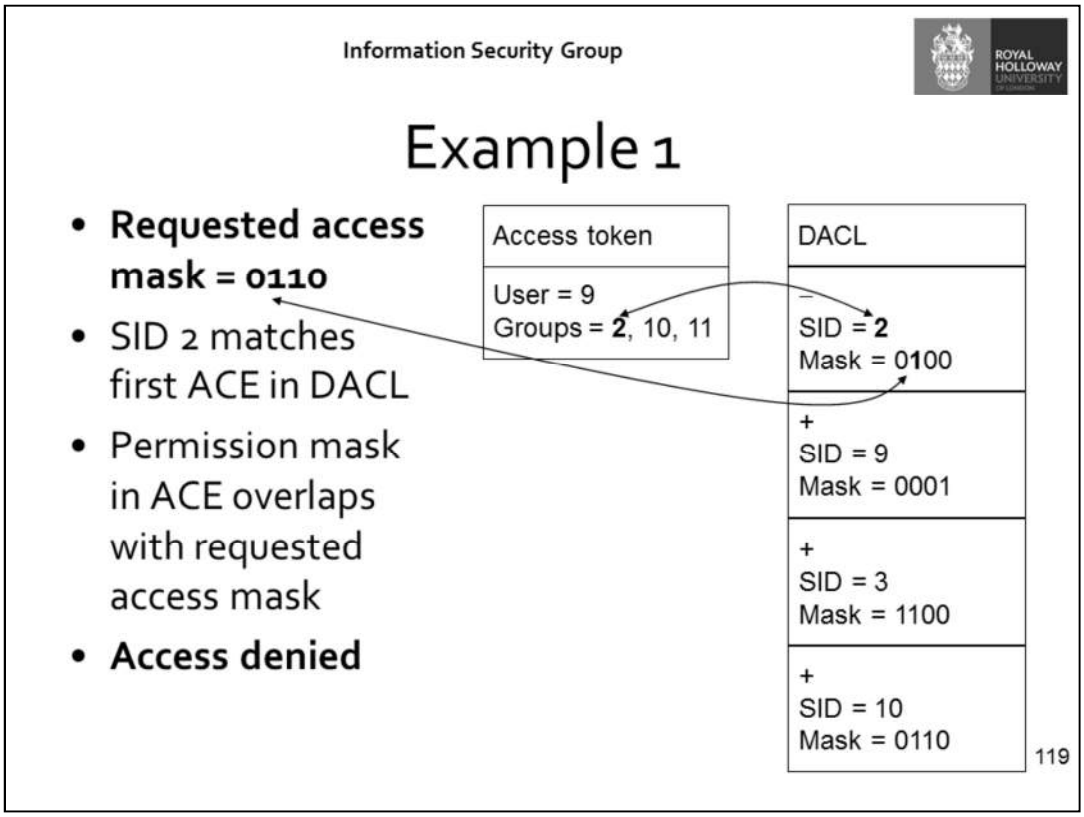
We use numbers to represent SIDs.

We assume that the `INHERIT_ONLY` flag is turned off in the inheritance mask of every ACE.



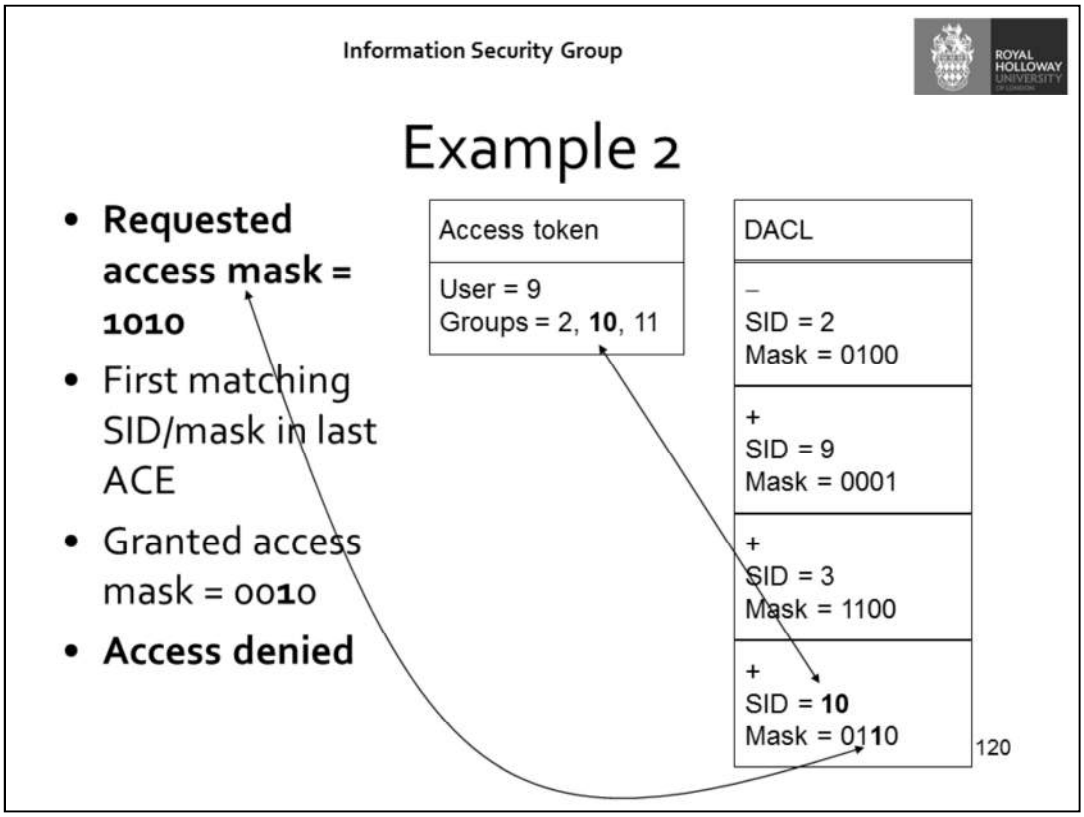

In our simple examples we assume that the requesting process has an associated access token containing user SID 9, and three group SIDs (2, 10 and 11). We also suppose that the object to which access is requested has a DACL with four ACEs.

Note that the first ACE is an 'access denied' ACE, i.e. it indicates a SID which is **not permitted** to have access (this is signified by the minus sign at the top of the box). The other three ACEs are 'access granted' ACEs, as signified by the plus sign.




We first consider the case where the requested access mark is 0110.

The SID in the first ACE is 2. Hence this ACE may apply to this request, since 2 is one of the SIDs in the access token. The second bit in the mask in this ACE is set. The second bit in the requested access mask is also set, and hence **this ACE applies**. Since this ACE is an **access denied** ACE, access is denied, and processing of the ACL stops at this point.




We second consider the case where the requested access mark is 1010. The granted access mask is initially set to **0000**.

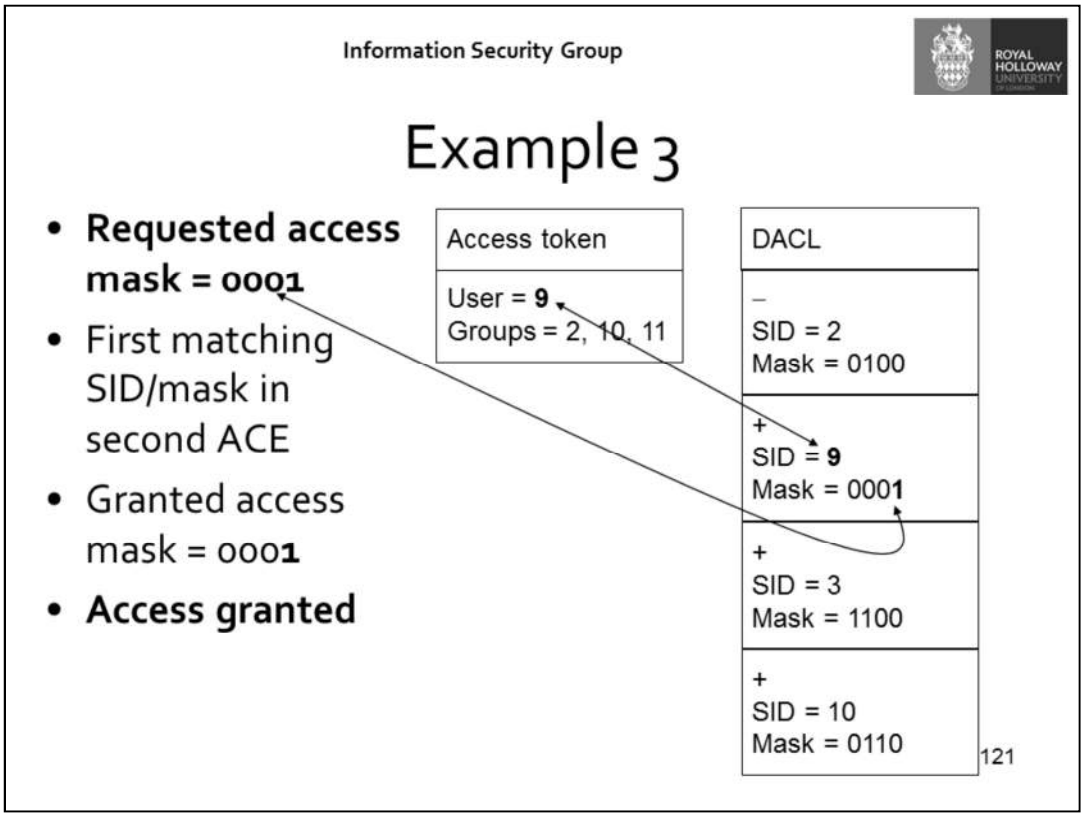
The SID in the first ACE is 2. Hence this ACE may apply to this request, since 2 is one of the SIDs in the access token. However, there is no match between any of the bits in the mask in the ACE and the requested access mask. Hence this ACE **does not apply**.

The SID in the second ACE is 9. Hence this ACE may apply to this request, since 9 is one of the SIDs in the access token. However, there is no match between any of the bits in the mask in the ACE and the requested access mask. Hence this ACE **does not apply**.

The SID in the third ACE is 3. Hence this ACE **does not apply** to this request, since 3 is not one of the SIDs in the access token.

The SID in the fourth ACE is 10. Hence this ACE may apply to this request, since 10 is one of the SIDs in the access token. There is also a match between one of the bits in the mask in the ACE and the requested access mask, namely the third bit. This matching bit is now added to the granted access mask, i.e. the granted access mask becomes **0010**.

There are no more ACEs, and hence at this point the granted access mask (0010) is compared to the requested access mask (1010). They are not equal and hence **access is denied**.

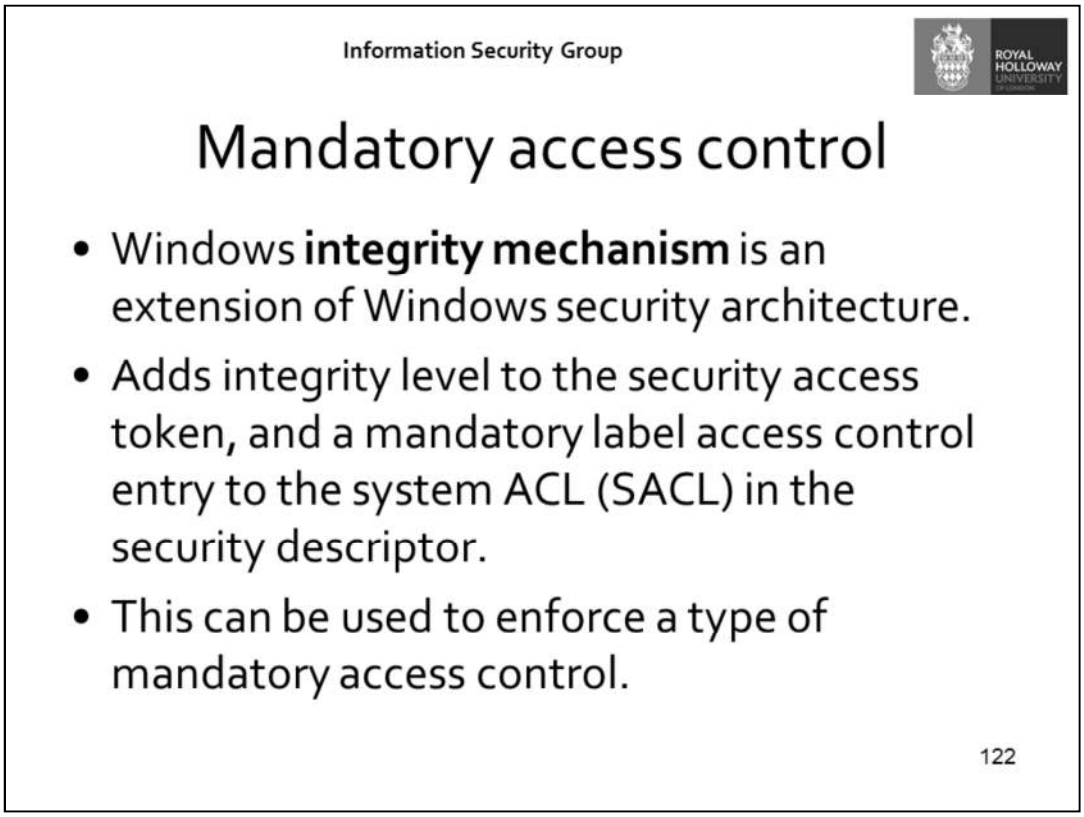



Finally we consider the case where the requested access mark is **0001**. The granted access mask is initially set to **0000**.

The SID in the first ACE is 2. Hence this ACE may apply to this request, since 2 is one of the SIDs in the access token. However, there is no match between any of the bits in the mask in the ACE and the requested access mask. Hence this ACE **does not apply**.

The SID in the second ACE is 9. Hence this ACE may apply to this request, since 9 is one of the SIDs in the access token. There is also a match between one of the bits in the mask in the ACE and the requested access mask, namely the fourth bit. This matching bit is now added to the granted access mask, i.e. the granted access mask becomes **0001**.

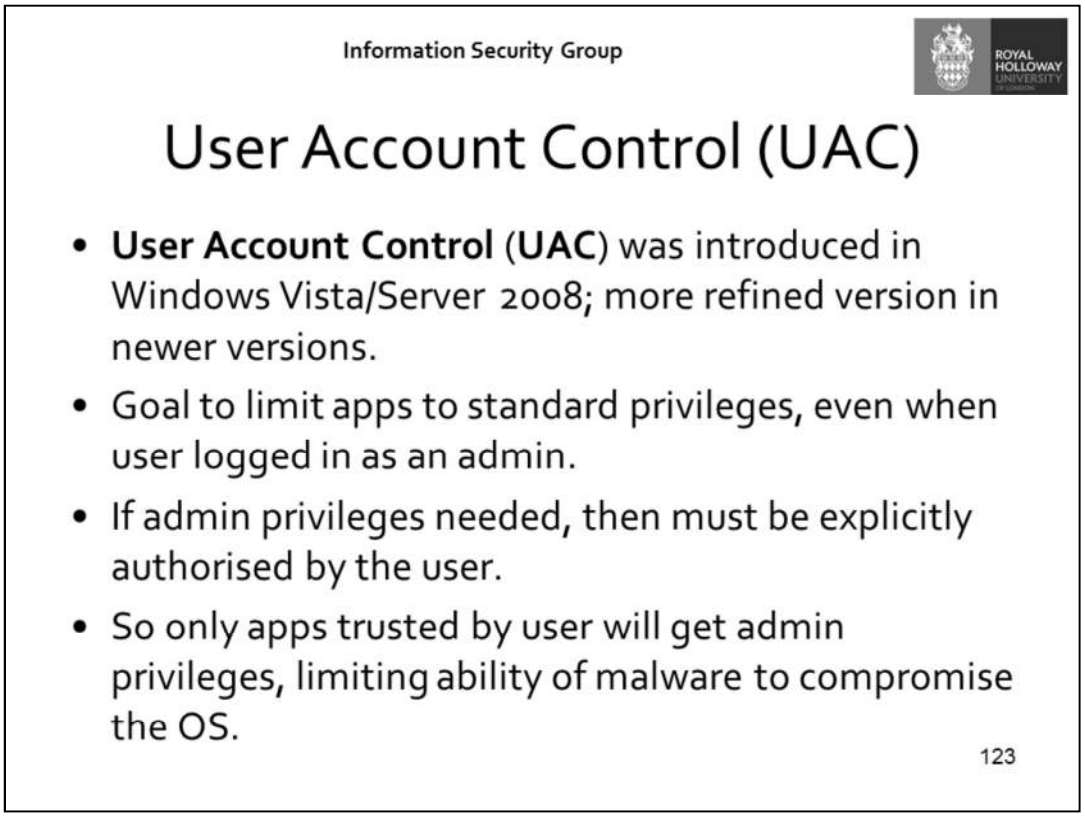
Every time the SRM finishes processing an ACE in the DACL, the granted access mask (0001) is compared to the requested access mask (0001). They are equal and hence **access is granted**.



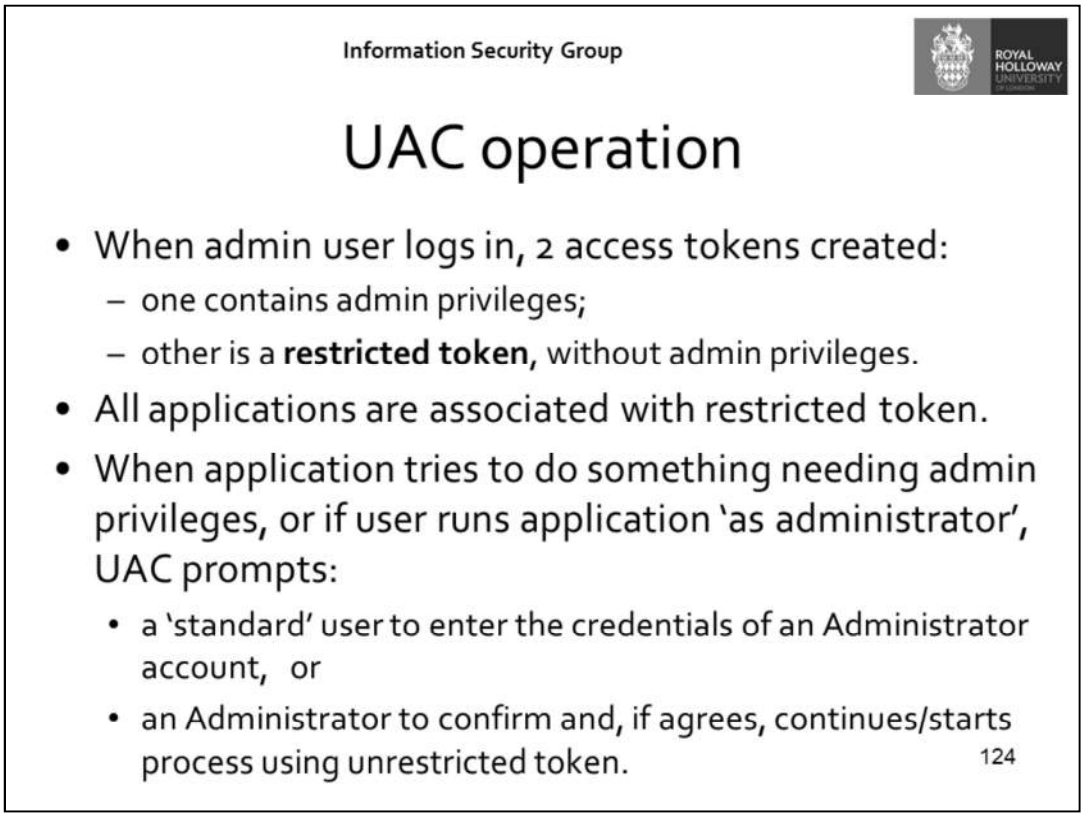

The Windows **integrity mechanism** is an extension of the Windows security architecture.

It adds an integrity level to the security access token, and a mandatory label access control entry to the system ACL (SACL) in the security descriptor.

This can be used to enforce a type of mandatory access control.



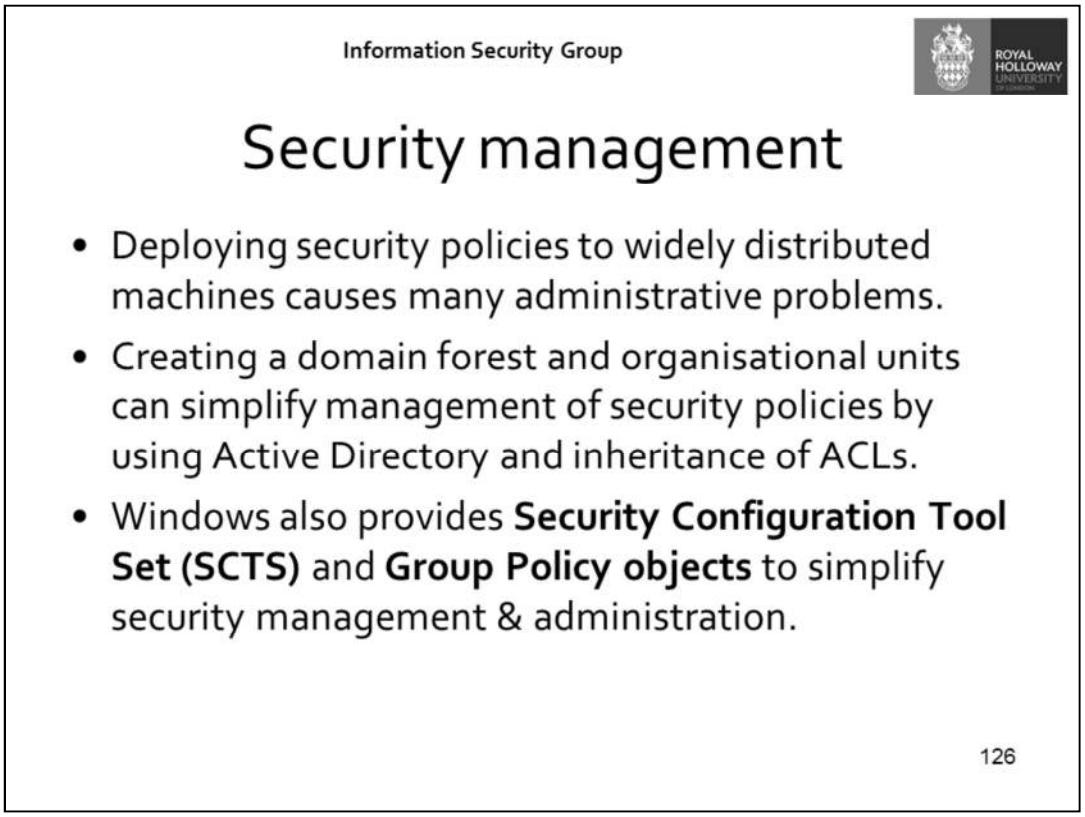

**User Account Control** (**UAC**) was introduced in Windows Vista and Windows Server 2008; a more refined version is present in subsequent versions of Windows. The goal of UAC is to improve security by limiting applications to standard user privileges even when a user is logged in as an administrator. If admin privileges are required then this must be explicitly authorised by the user. As a result, only applications trusted by the user will enjoy administrative privileges, and this will limit the ability of malware to compromise the OS.




When a user in the Administrators group logs in, two separate access tokens are created. The first token contains all the privileges typically given to an administrator, and the second is a **restricted token**, somewhat similar to that which a non-administrator (standard) user would receive. All applications run by the user, including the Windows desktop, are associated with the restricted token. When an application tries to do something requiring administrator privileges, or when a user indicates it wishes to run an application `as administrator', UAC will prompt either:

- a 'standard' user to enter the credentials of an Administrator account, or
- an Administrator for confirmation and, if consent is given, continue or start the process using an unrestricted token.

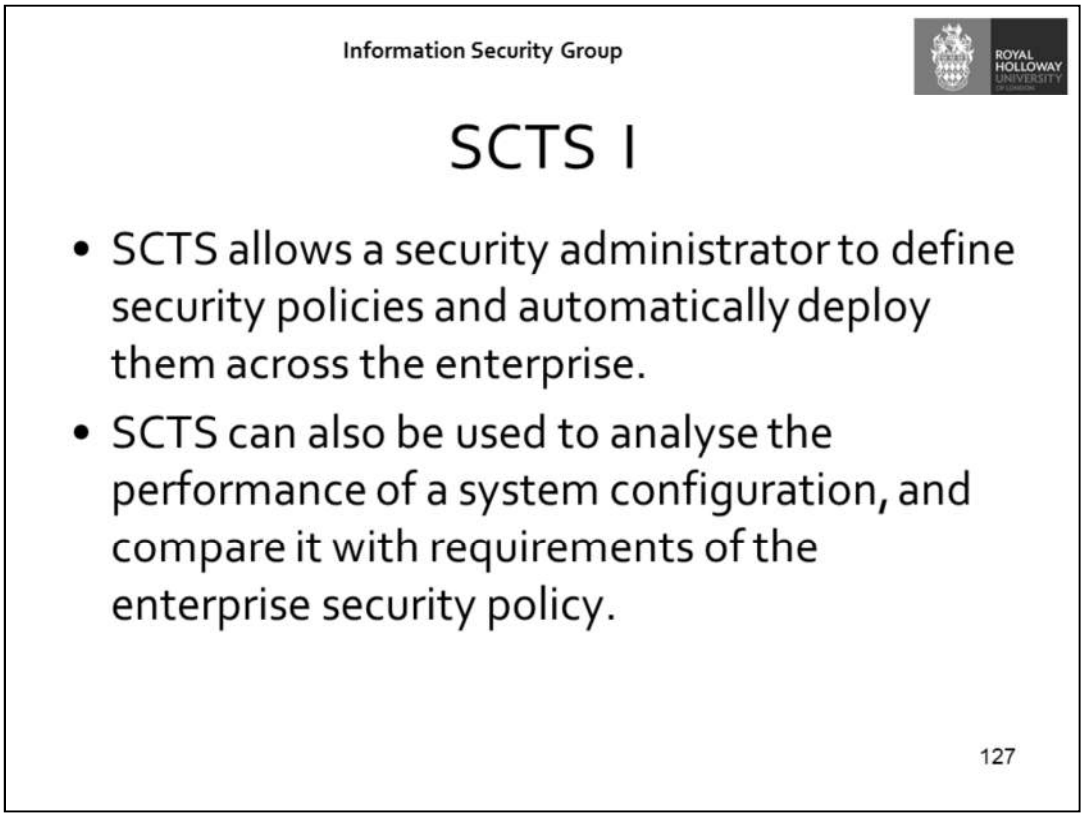

We conclude by briefly introducing some of the functions Windows provides to support security management across a domain.




The deployment of security policies to widely distributed machines causes many administrative problems in modern enterprises.

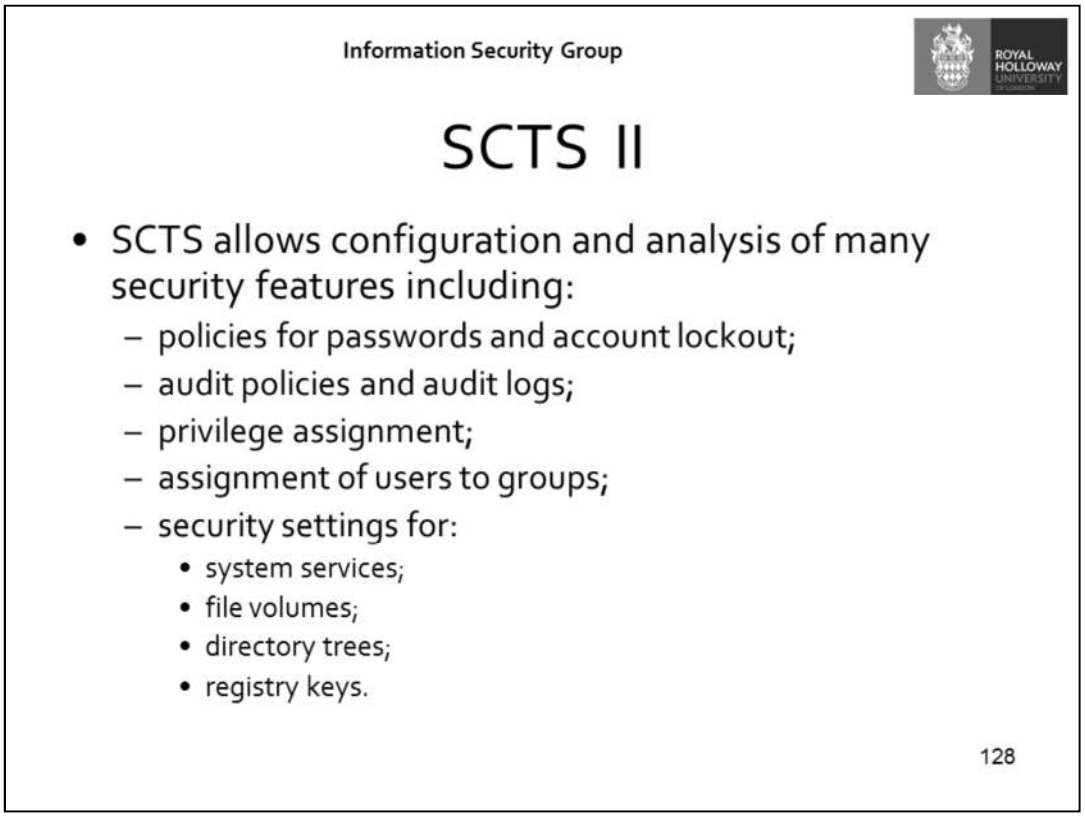
Creating a domain forest and suitable organisational units can simplify the management of security policies by exploiting the features of Active Directory and the inheritance of ACLs.

Windows also provides the **Security Configuration Tool Set** (SCTS) and **Group Policy objects** to simplify the management and administration of security.



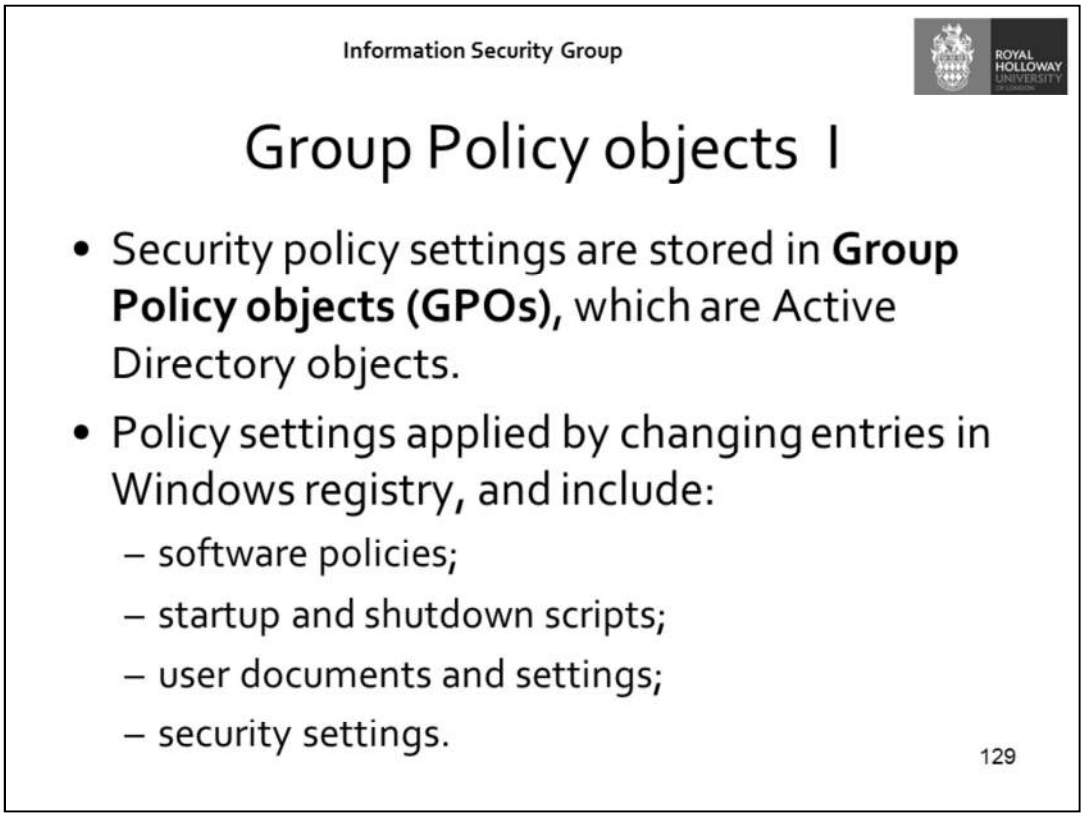

SCTS allows a security administrator to define one or more security policies and automatically deploy them across the enterprise.

SCTS can also be used to analyse the performance of a particular system configuration and compare it with the requirements of the enterprise security policy.




SCTS allows the configuration and analysis of many security features including:

- policies for passwords and account lockout;
- audit policies and audit logs;
- privilege assignment;
- assignment of users to groups;
- security settings for:
    - system services;
    - file volumes;
    - directory trees;
    - registry keys.

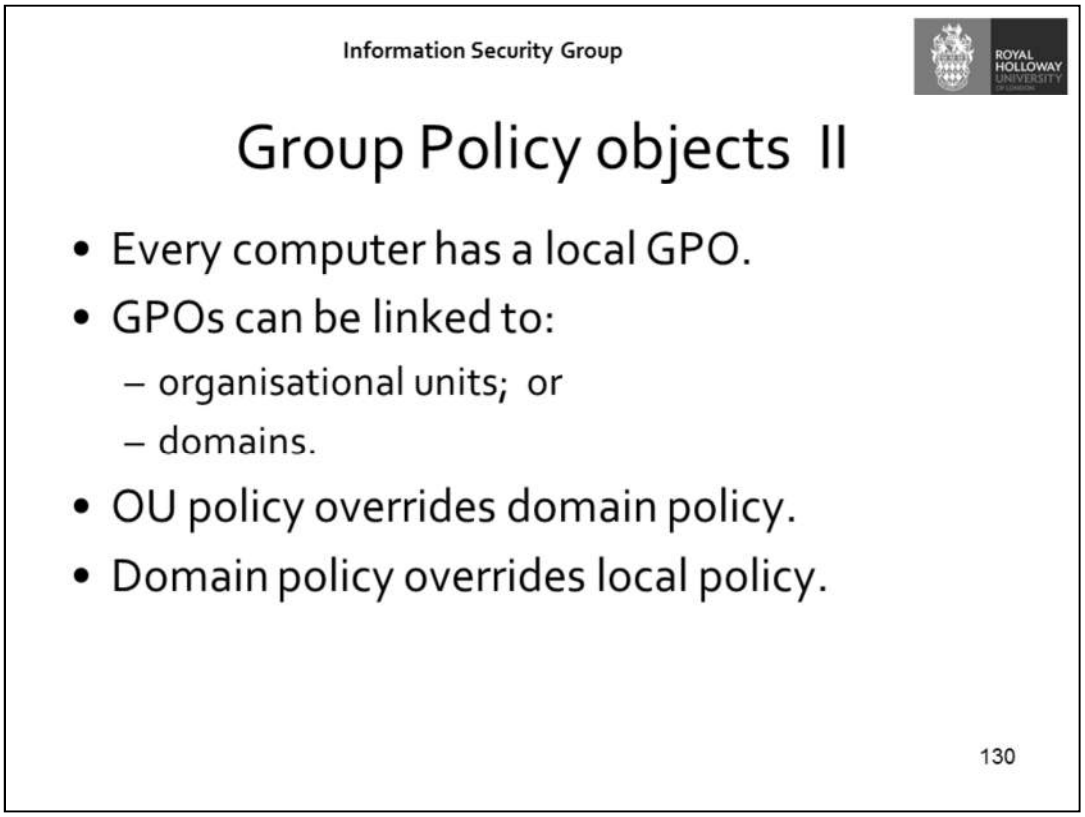



Security policy settings are stored in Group Policy objects (GPOs):

- GPOs are Active Directory objects.

Policy settings are applied by changing entries in the Windows registry and include:

- software policies;
- startup and shutdown scripts;
- user documents and settings;
- security settings.



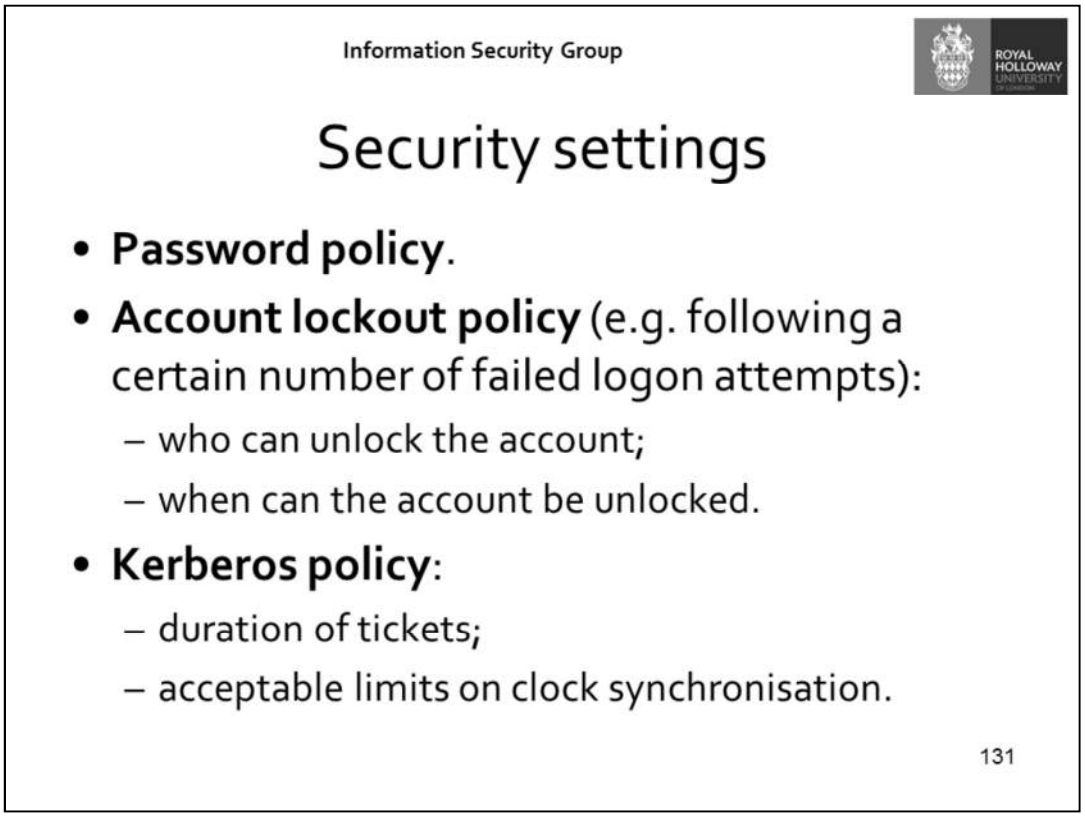

Every computer has a local GPO.

GPOs can also be linked to:

- organisational units; or to
- domains.

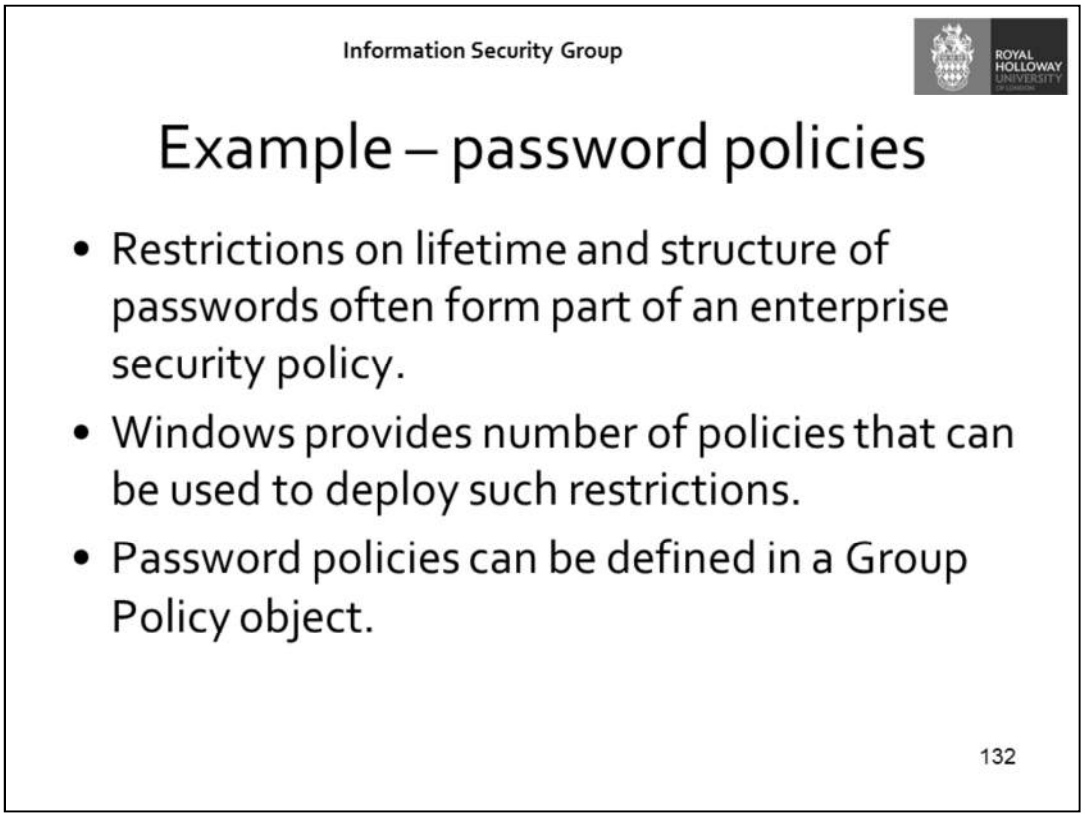
An OU policy overrides any domain policy.

A domain policy overrides any local policy.



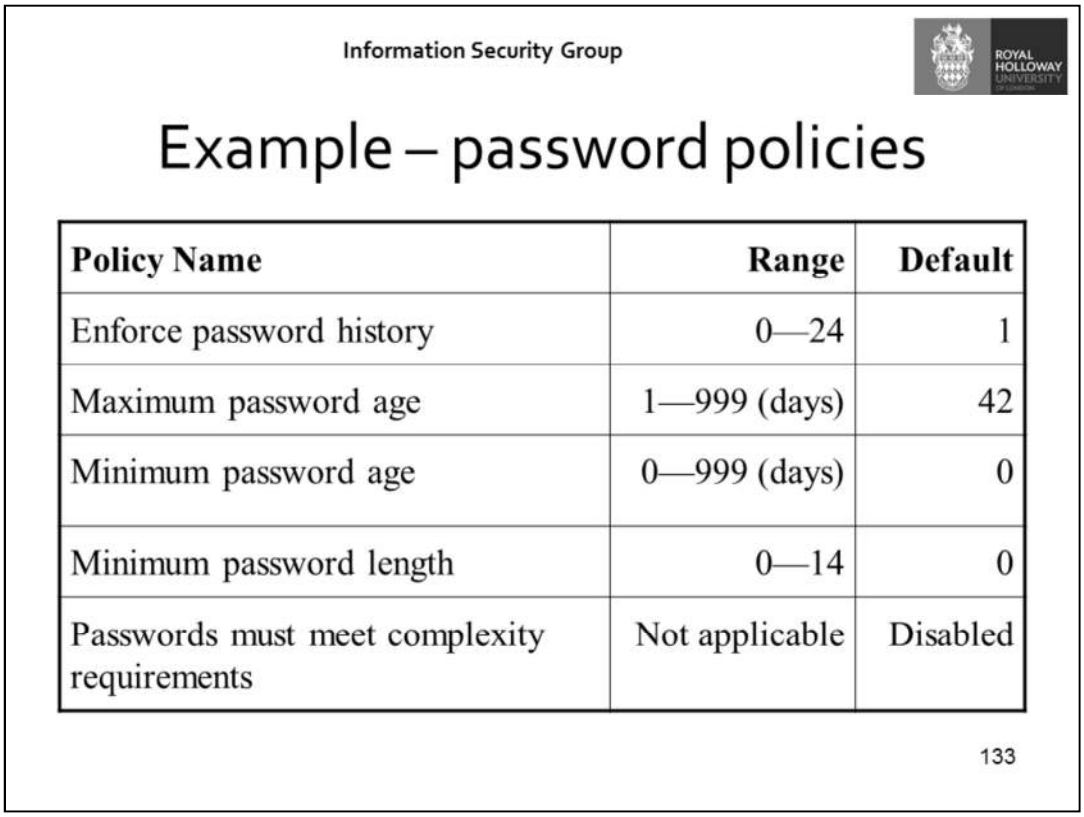

The following policies can be set using security settings:

- The Password policy (discussed on next slide).
- The Account lockout policy (which applies, for example, following a certain number of failed logon attempts), covering:
  - who can unlock the account;
  - when can the account be unlocked.
- The Kerberos policy, covering:
  - the duration of tickets;
  - acceptable limits on clock synchronisation.

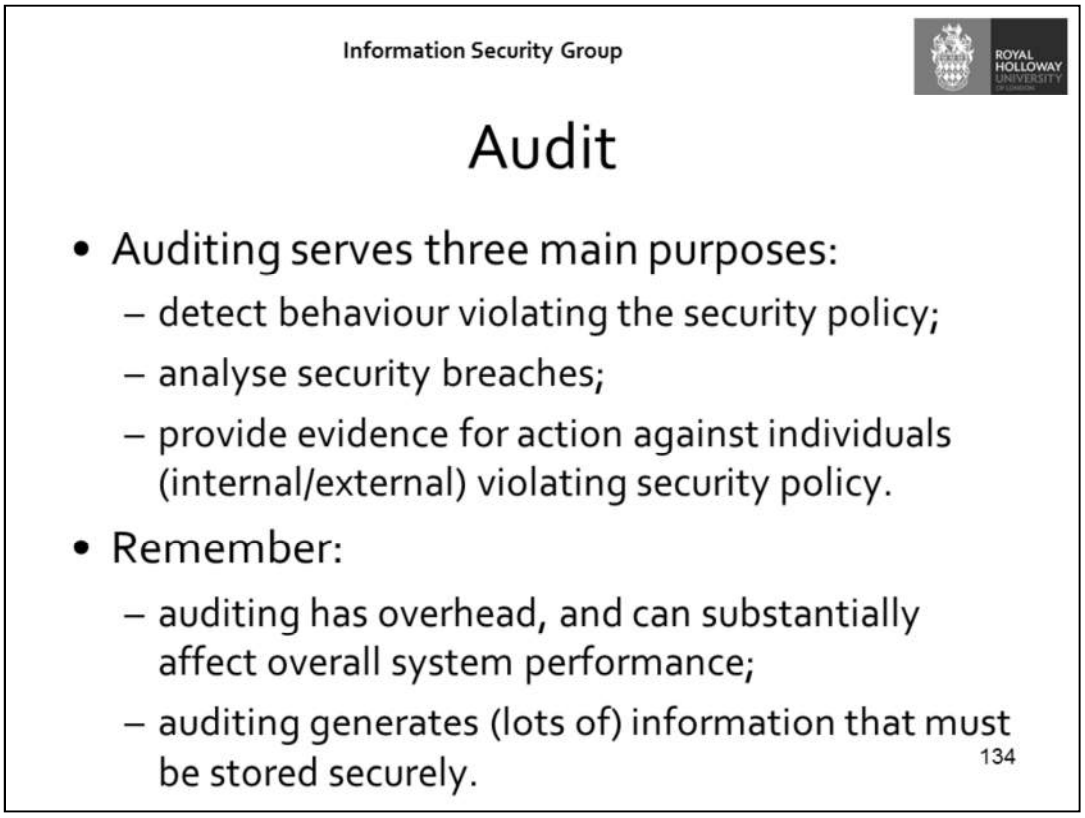



Restrictions on the lifetime and structure of passwords often form an important part of an enterprise security policy. Windows provides a number of policies that can be used to define and deploy such restrictions.

Password policies can be defined in a Group Policy object.




Examples of predefined password policy settings are shown.

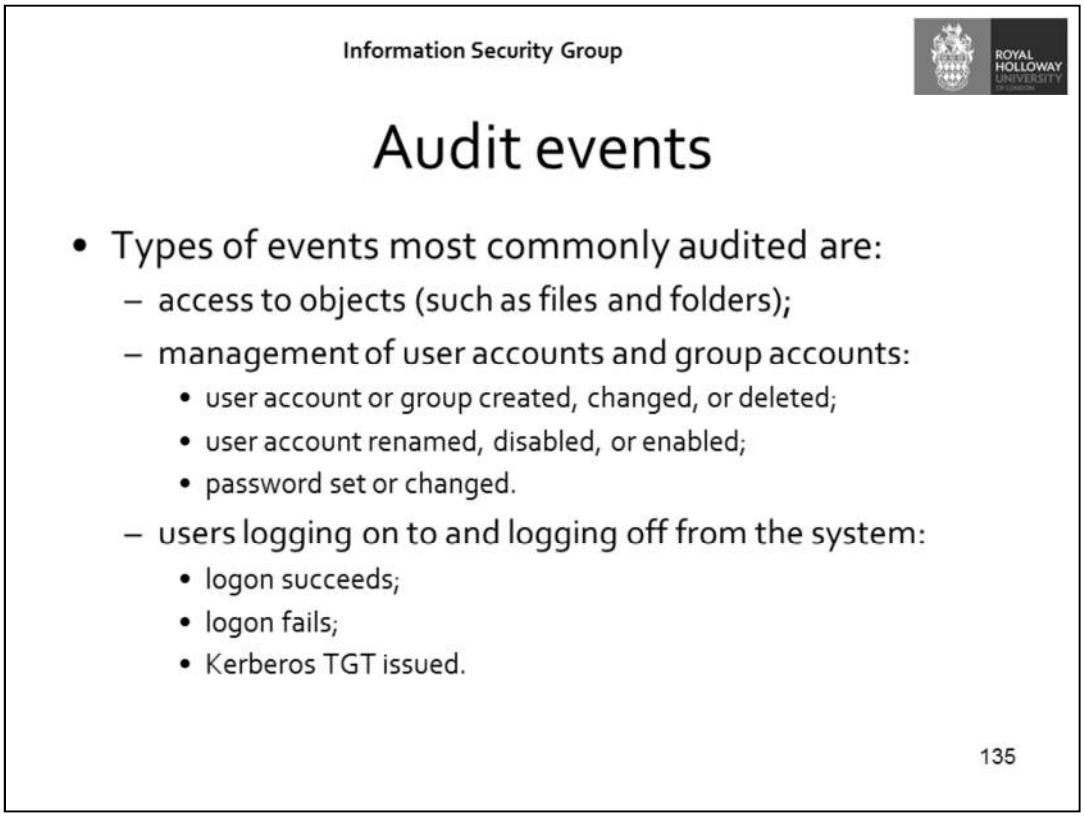



Auditing serves three main purposes:

- to detect behaviour that violates the security policy;
- to analyse security breaches;
- to provide evidence for possible actions against individuals (internally or externally) who violate security policy.

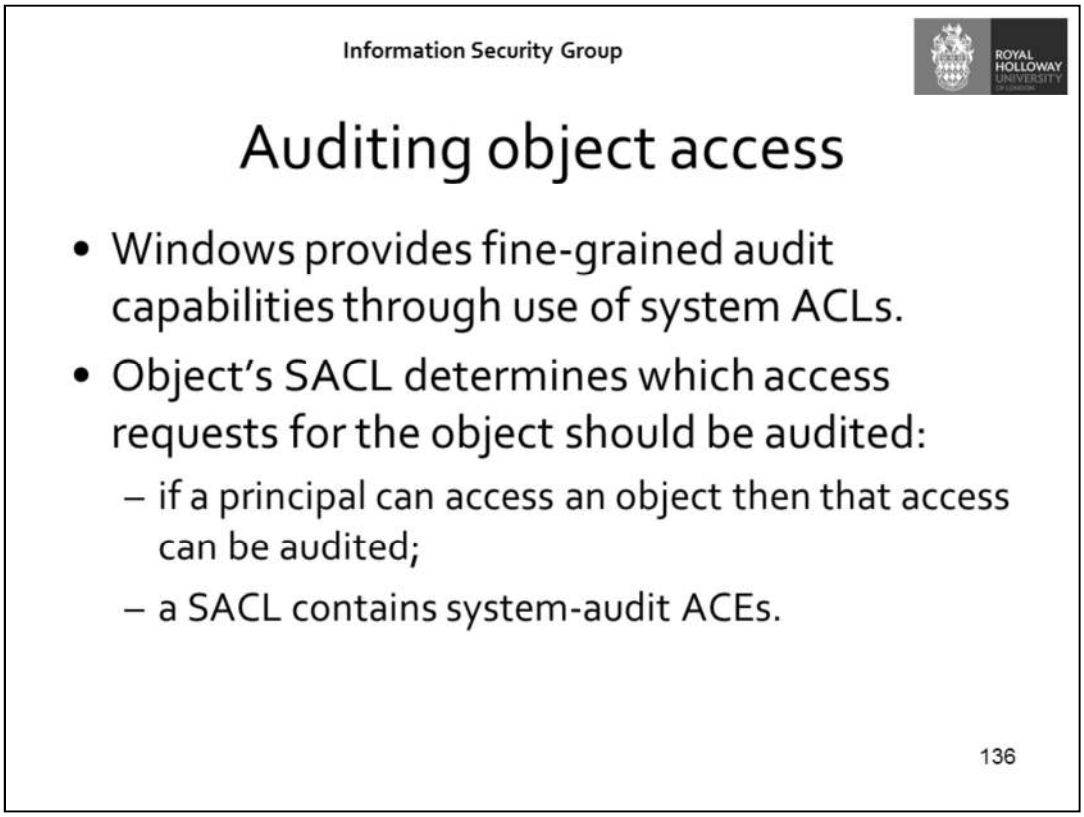
Remember:

- auditing always incurs overheads and can substantially affect overall system performance;
- auditing generates (potentially large quantities of) information that needs to be stored securely.




The types of events which are most commonly audited are:

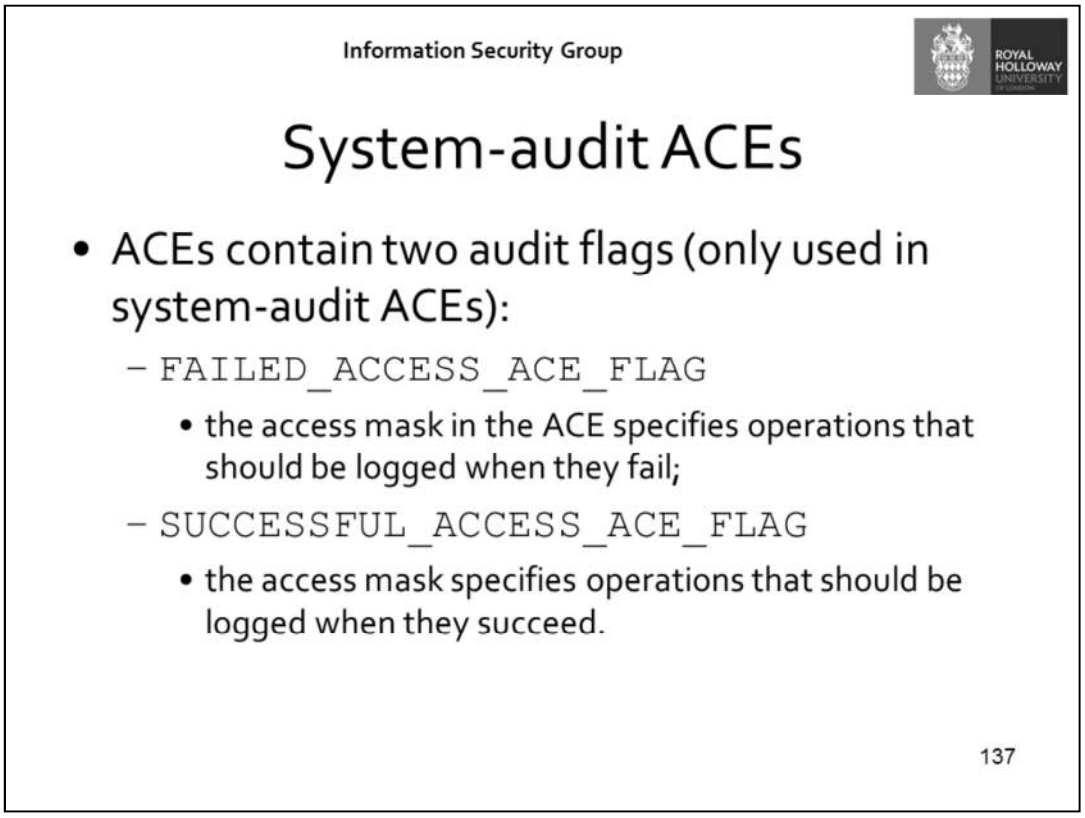
- access to objects (such as files and folders);
- management of user accounts and group accounts:
    - user account or group is created, changed, or deleted;
    - user account is renamed, disabled, or enabled;
    - password is set or changed.
- users logging on to and logging off from the system:
    - logon succeeds;
    - logon fails;
    - Kerberos TGT issued.




Windows provides fine-grained audit capabilities through the use of system ACLs.

An object's SACL determines which access requests for the object should be audited:

- if a principal can access an object then that access can be audited;
- a SACL contains system-audit ACEs.




ACEs contain two audit flags (that are only used in system-audit ACEs):

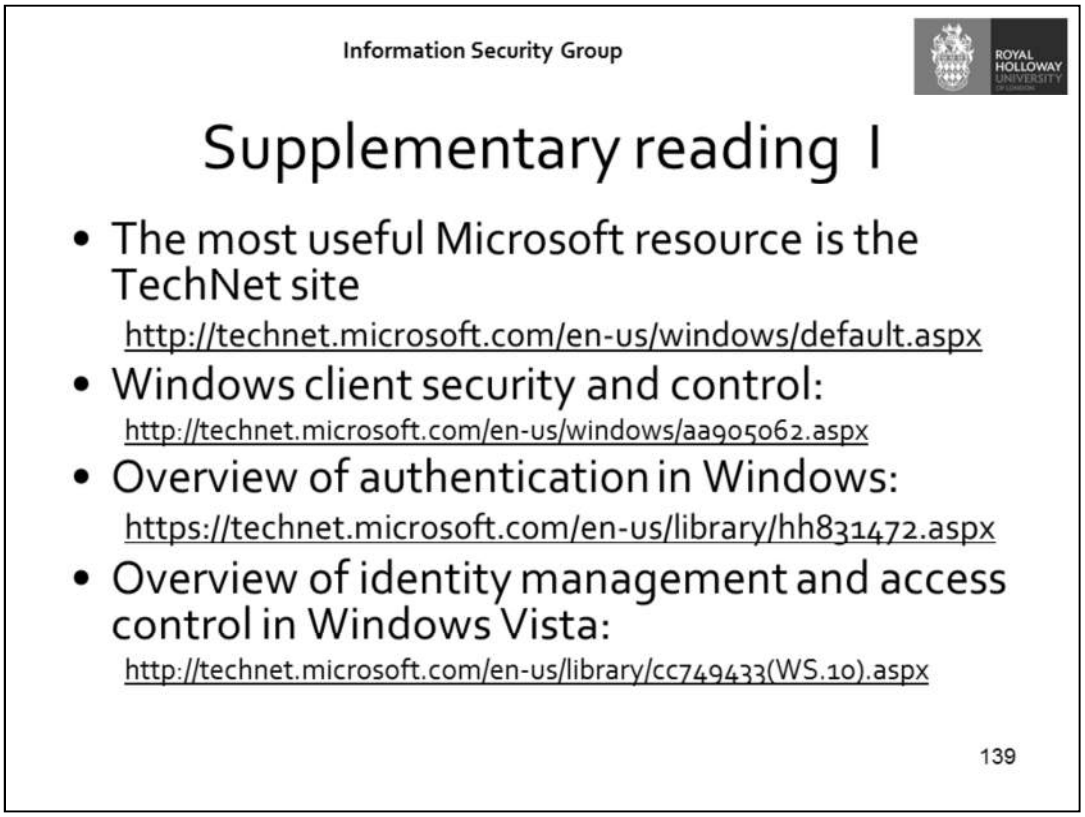
`FAILED_ACCESS_ACE_FLAG`

the access mask in the ACE specifies the operations that should be logged when they fail;

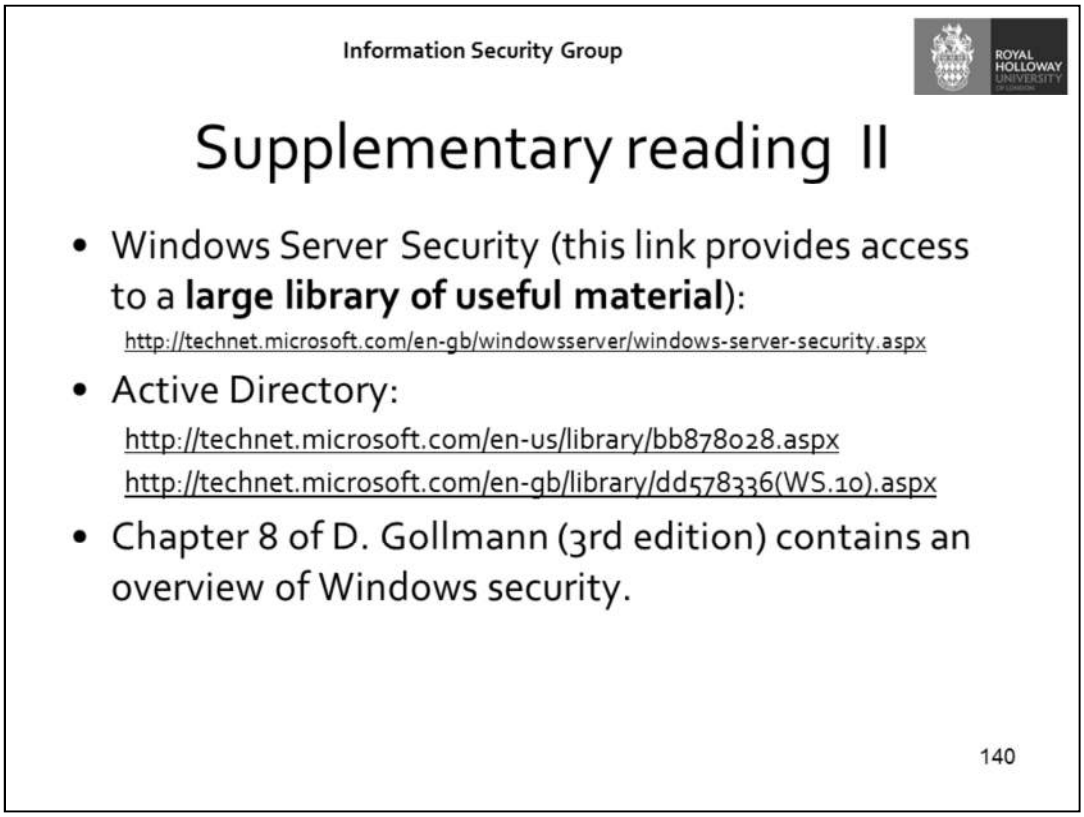
`SUCCESSFUL_ACCESS_ACE_FLAG`

the access mask specifies the operations that should be logged when they succeed.


There are a wide variety of helpful resources on the topic of Windows security.




The most useful Microsoft resource is the TechNet site
http://technet.microsoft.com/en-us/windows/default.aspx
Windows client security and control:
http://technet.microsoft.com/en-us/windows/aa905062.aspx
Overview of authentication in Windows:
https://technet.microsoft.com/en-us/library/hh831472.aspx
Overview of identity management and access control in Windows Vista:
http://technet.microsoft.com/en-us/library/cc749433(WS.10).aspx




Windows Server Security (this link provides access to a **large library of useful material**):

http://technet.microsoft.com/en-gb/windowsserver/windows-server-security.aspx

Active Directory:

http://technet.microsoft.com/en-us/library/bb878028.aspx

http://technet.microsoft.com/en-gb/library/dd578336(WS.10).aspx

Chapter 8 of D. Gollmann's *Computer Security* (3rd edition) contains an overview of Windows security.