1495 UNIVERSITY OF ABERDEEN | CELEBRATING 525 YEARS 1495 – 2020

ABERDEEN 2040

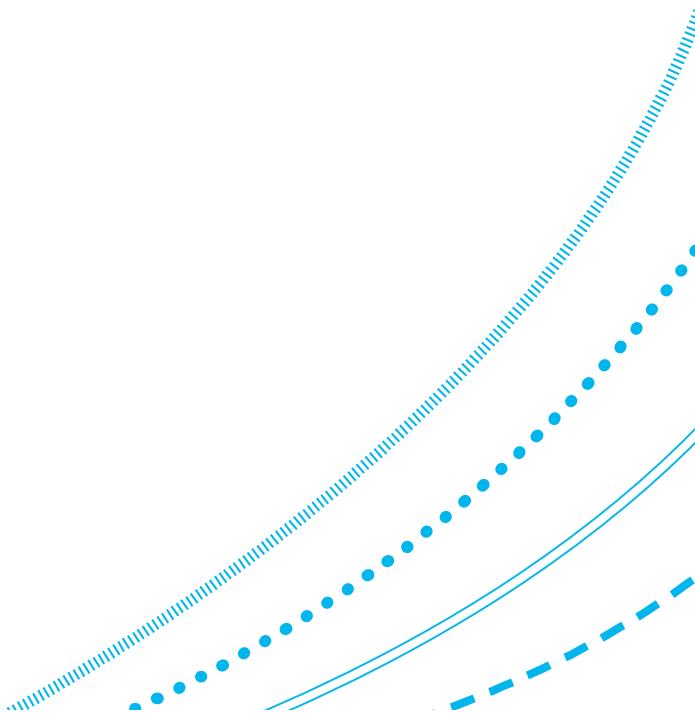
Network Security Technology

Enterprise access control, identity management and OS-level access

September 2025

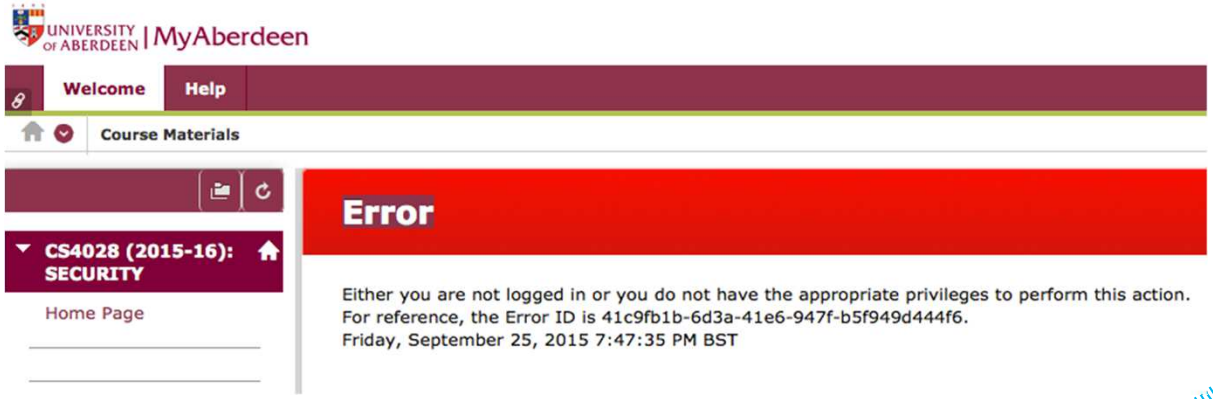
Outline of lecture

1. Introduction.
2. Enterprise access control.
3. Identity management.
4. OS-level access.
5. Indirection mechanism: groups and roles.

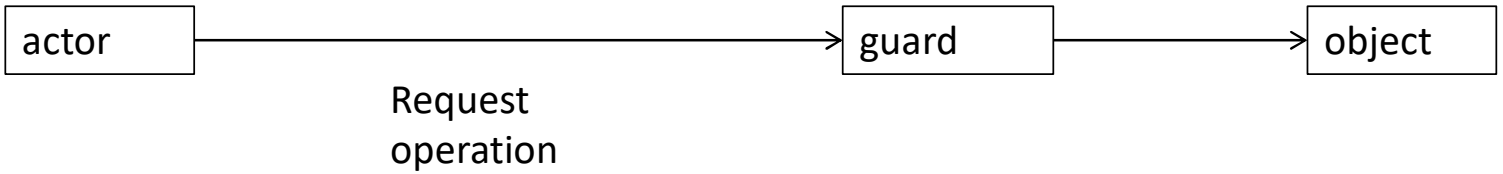


Access control in information security

1. Access control is about ensuring that **just the right agents** have access to **just the right resources** (e.g., data, information, devices) under **just the right conditions** (in particular, at the right time).
2. Access control is an important and general problem of information security. However, a lot of this lecture will explain it in the context of computer security. We will see different examples later.



A basic model of access control



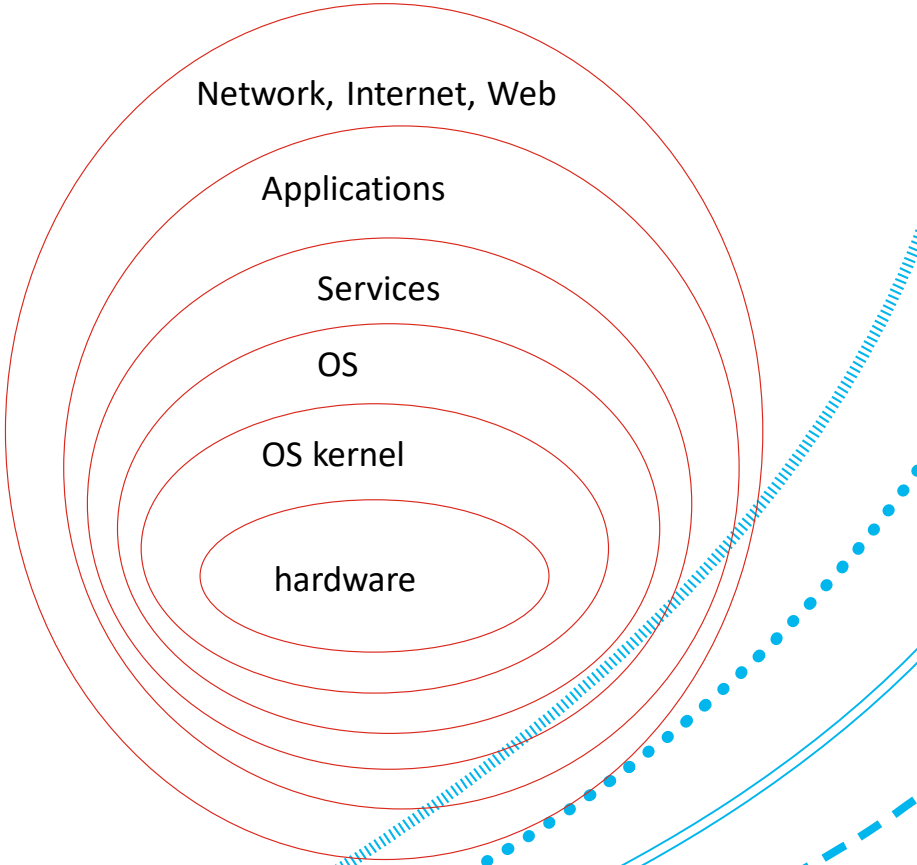
1. In the model:
    1. An actor makes a **request** for an access operation on a resource object.
    2. A guard (reference monitor) **grants** or **denies** access.
2. In order for this to work:
    1. All requests must go through the guard, and
    2. The guard must have integrity.

Protection, policy and mechanism

1. Consider a **protection domain**: a place where various entities have (potential) access to shared resources.
2. This leads to **protection goals**: requirements for what should be protected from access by what.
3. This leads to **security policies** for protection: some kind of formulation that states, or makes statements about processes intended to achieve, the protection goals.
    1. In particular, there are **access control policies** that say who should have access to what, and under which conditions.
4. **Security/protection mechanism:** put in place to deal with comply with policy, or ameliorate risk.
    1. Prevention, detection, accountability, recovery …
    2. In particular, there are **access control mechanisms**

Onion model of computer protection mechanism

1. **Protection** is a general computer security issue, not just data protection.
    1. Users run applications
    2. Applications use services (including middleware, e.g. a Database Management Service DBMS, object reference broker (ORB), or a browser)
    3. They run on top of the OS
    4. OS may have a kernel (microkernel, hypervisor) that mediates access to processor and memory.
    5. Hardware: physical storage and manipulation of data.
    6. We might also add the network, internet and web layers.

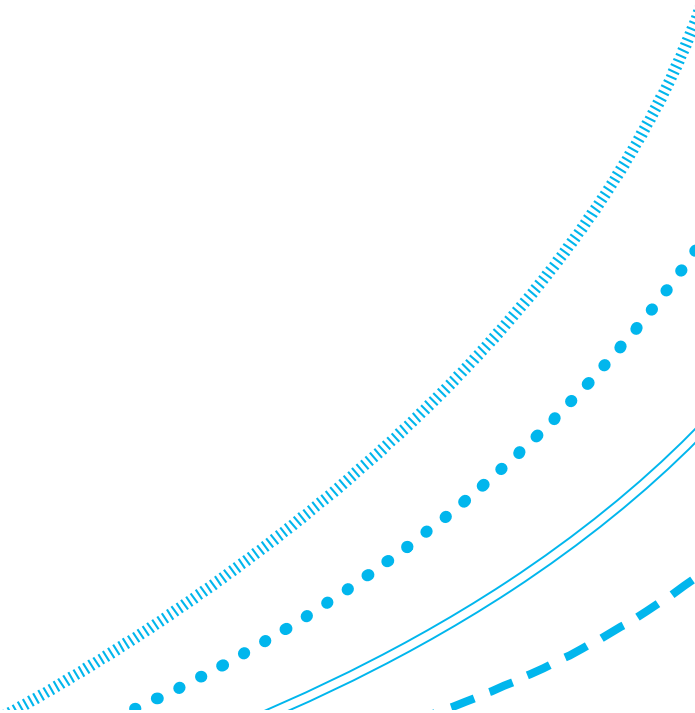


Authentication and authorization with identities

1. Two main step in performing access control:
    1. **Authentication**: process of verifying attributes claimed by an entity/actor
    2. **Authorization**: deciding whether (and how) to grant access.
2. We focus on authorization today. We think about the functionality of the guard, but not so much about low-level implementation details.

Authorization

1. This is itself commonly divided into 2 steps:
    1. The setting (and changing) of access controls
        1. E.g. setting **permissions** to perform access operations
        2. Sometimes called *authorization* (confusingly!)
    2. The use by the guard of the access control information.
        1. E.g. consulting some data structure recording permissions.



Accountability

`` *The property of a system or system resource that ensures that the actions of a system entity may be traced uniquely to that entity, which can then be held responsible for its actions*."

http://tools.ietf.org/html/rfc4949 Internet Security Glossary

1. Accounting mechanisms are used to provide for accountability.

Non-repudiation and the AAA framework

1. Accountability is linked to non-repudiation: it is not possible (or difficult) for agents to deny (repudiate) their actions.
2. The AAA framework requires an access control system to include mechanisms for:
    1. Authentication,
    2. Authorization and
    3. Accounting.

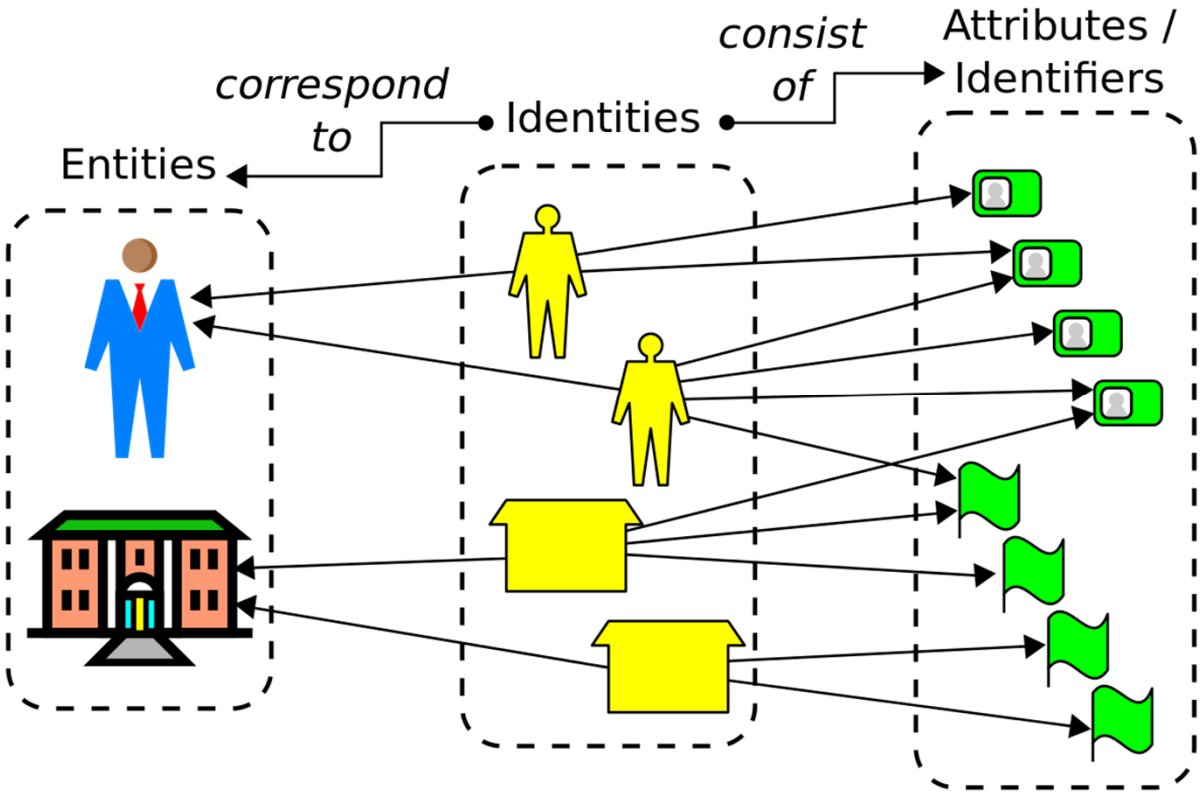
Identity and access management

1. In an enterprise setting, one finds **Identity and Access Management** (IAM) solutions, sometimes just called **identity management** (IdM) **.**
2. Function is management of:
    1. Identities (creation, maintenance, deletion),
    2. Authentication,
    3. Authorization,
    4. Accounting,
    5. Interaction with Identity Federations.
3. Management is centralized. Admins can set policy at a single controlling server, not at each host separately.

Identity

1. An **entity** could be real (e.g. a person) or virtual (e.g. a department).
2. **Identities** are entity names, including user names (from some namespace of possible names).
    1. An entity could have more than one identity, but every identity is associated with just one entity.
3. An identity can have many **attributes**.
    1. Many identities can have the same attribute.
    2. Department, role, clearance level, ownership of resource.

Identity concept



Authentication

1. Increasingly in the enterprise, users have a single set of credentials (password, smartcard, biometric) etc. that they use to authenticate. This is **single sign-on**.
2. Authentication is managed centrally, i.e., not at each server running each service.

Authorization

1. Policies are also managed centrally, i.e. not by individual services. There is a central reference monitor.
2. (There can be a hierarchical system of reference monitors).

Identity federation

1. An enterprise may support use of services from other organizations. It may allow an entity to use its credentials to authenticate for those services.
2. The second organization has services that can work using identities from the first enterprise.
3. The authentication is done by the enterprise to which the entity belongs. Protocols allow the second organization to be assured that this has been done adequately.
4. This is an example of **identity federation**.

Computer access control: actors

1. **Subject**: An active entity within an IT system. It can make requests and be granted access to objects or make statements affecting access control.
2. Examples of subjects include the following:
    1. A user ID,
    2. A cryptographic key belonging to a user,
    3. Processes, threads, services,
    4. Resources.
3. Subjects can operate on behalf of others, particularly *principals*.
4. **Principal**: A special kind of subject. The point is that a particular access request by a subject may be initiated by another, the principal. Example:
    1. A process (subject) may attempt to access a file (object) on behalf of a UserID

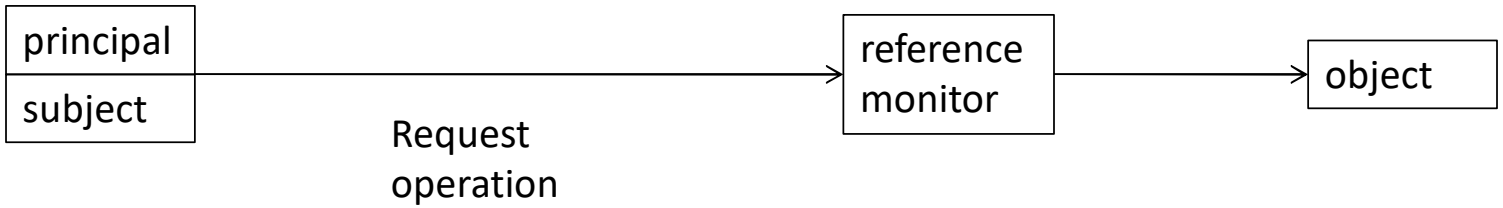
Policy language

1. Policy gets formalized for execution on a machine
2. Whether explicitly or not, we have a **formal policy language** in which this gets written.
3. “Whereof one cannot speak, thereof one must be silent” – Wittgenstein, Tractatus
4. Policy language will embody exactly and only the semantic concepts that are used for deciding access.

Subject and object

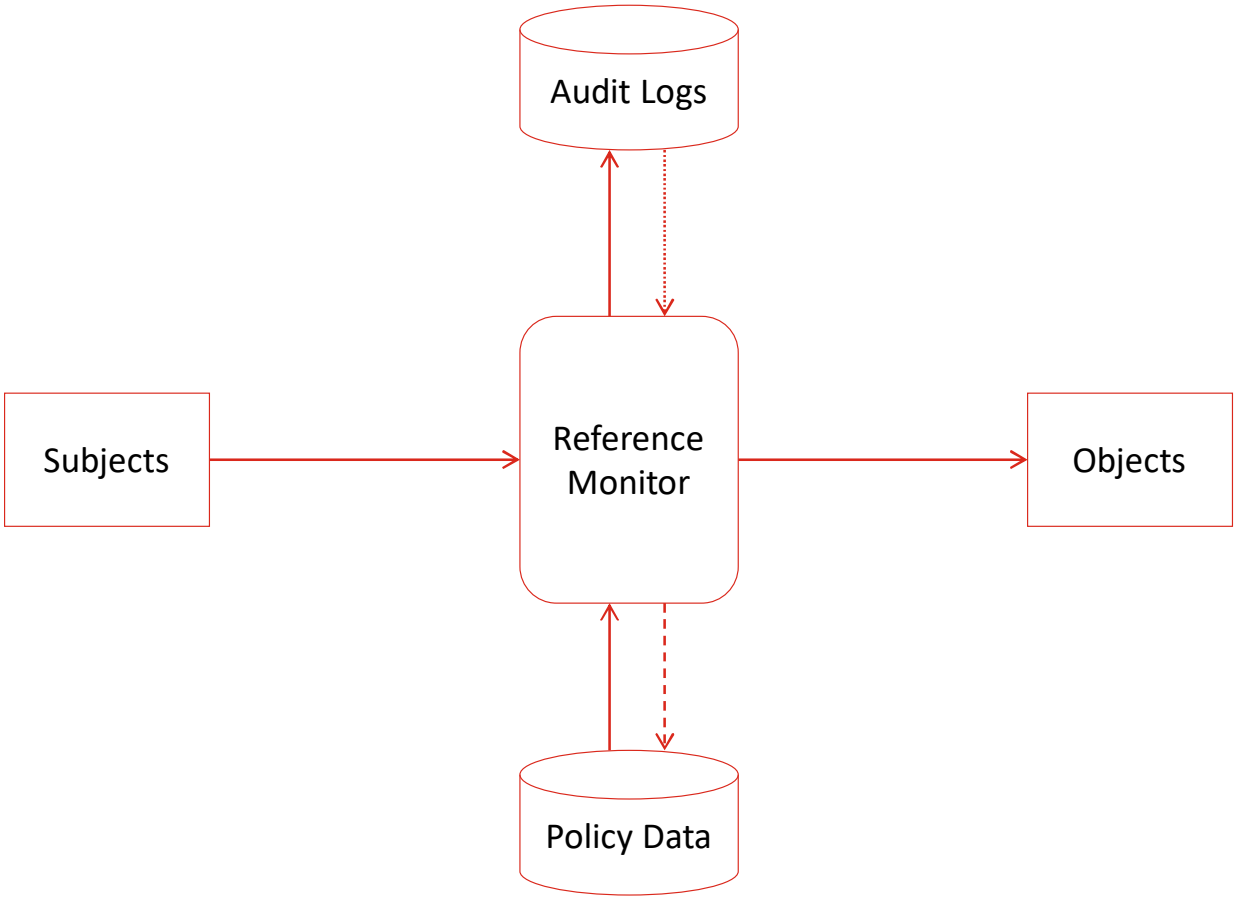
1. **Object**: the entity being accessed.
2. Active/passive distinction between subjects and objects like in grammar of many natural languages. Examples include the following:
    1. ``*s* does operation *a* to *o*''
    2. ``*s* requests operation *a* on *o*''.
3. Some computer access operations:
    1. Read, Execute, Create, Write, Append,
    2. Move, Delete, Change owner, Change access permissions,
    3. Different meanings and families in different systems.

Access control with subjects & principals



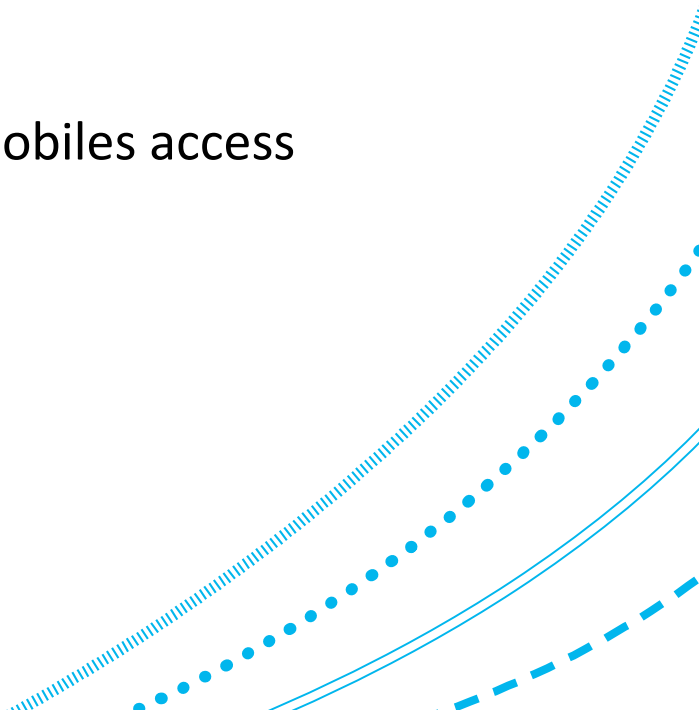
1. In computer security, the guard is often referred to as a **reference monitor**.
2. In the model:
   1. a subject makes a request for an access operation on an object, on behalf of a principal.
   2. A reference monitor grants or denies the request.
3. In the slides below, we enrich this model with features to provide for:
   1. Accountability, and
   2. A separate data repository for maintaining rules (representing policy).

Reference monitor



Multi-user

1. Access control is more interesting in multi-user systems.
    1. Even if not, there will usually still be multiple subjects, so still need it.
2. Single-user devices like mobiles and iPads seem like a bit of a throwback.
    1. Cause serious security usability headaches.
3. Most systems multi-user at some level (e.g. since even mobiles access resources across a network.)



Setting policy

1. The word *policy* gets used in two ways here:
    1. (Goals) The intended objectives that the organization has for access.
    2. (Current rules) The collection of all automated access rights in the system.
2. This is because the reference monitor just implements the technical policy.
3. Anyway, who sets the rules?

Multi-user: who sets policy

1. Two well-known, traditional policy schemes:
    1. DAC
    2. MAC.
2. E.g., From the **Orange Book** (US, DoD, 1985), a.k.a. The Trusted Computer System Evaluation Criteria (TCSEC).

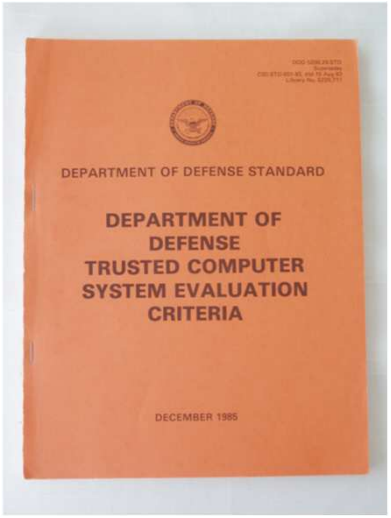


Figure: public domain

DAC

1. **Discretionary access control (DAC)**: Define an *owner* for each resource. Let the owner decide who should be allowed access. Control is at the discretion of the owner:
    1. This assumes a notion of ownership of resource (often by subjects or principals).
    2. Also need to be able to identify other individuals
    3. Often only allow individual who need-to-know.
2. Example:
    1. Child keeps diary with lock. Child may grant access to parent to read, but no-one else (other than his/her-self).

MAC

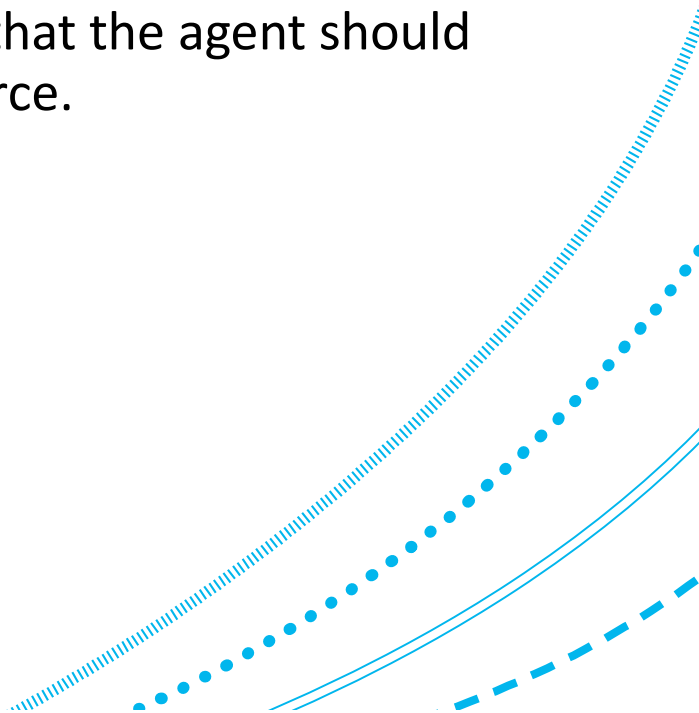
1. **Mandatory access control (MAC):** Have a system-wide policy about who has access. Set centrally by an administrator:
    1. Uses rules regarding which principals should access which objects.
    2. Often based upon artificial attributes of the individual (e.g. *clearance*) and resource (e.g. *classification*).
    3. Access is only granted to individuals with sufficient clearance to match the classification of the object.
    4. Sometimes called **rule-based access control**.

MAC examples

1. In some army, all army officers (but not lower ranks) can read all material marked (classified) Secret:
    1. Many MAC schemes use these kind of levels.
2. In some countries, there are records of drivers:
    1. Each record is `owned' by the corresponding driver.
    2. The law allows a court read access to a driver's record without his/her permission.
    3. The owner of the record has no say.

Three rights

1. Access control is about ensuring that just the right agents have access to just the right resources (e.g., data, information, devices) under the right conditions (including the right time).
2. A **privilege** or **permission** is a record (in the policy data) that the agent should be allowed to perform the particular access to the resource.

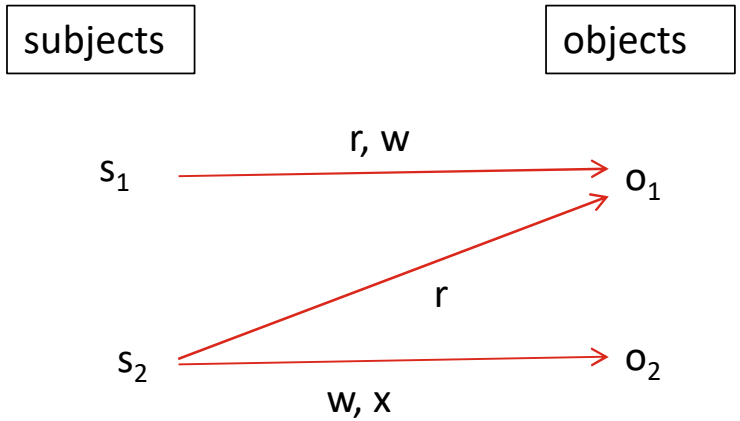


A common simplification: two rights, and right now

1. Access requests come in from subjects at some instant of time. At least in computer security we expect the request to be processed quickly (now). Access rights generally change more slowly.
2. The reference monitor consults some policy data structure that tells it if the request should be granted now:
    1. The data structure usually doesn't need to store **when**.
    2. The access data is just updated for future requests.
3. However, there is a general problem of computer security called **Time-of-check to time-of-use** (**TOCTTOU**). For how long should access be granted?

Access relations and graph

1. Can visualize the relation of `being able to perform operation' graphically.



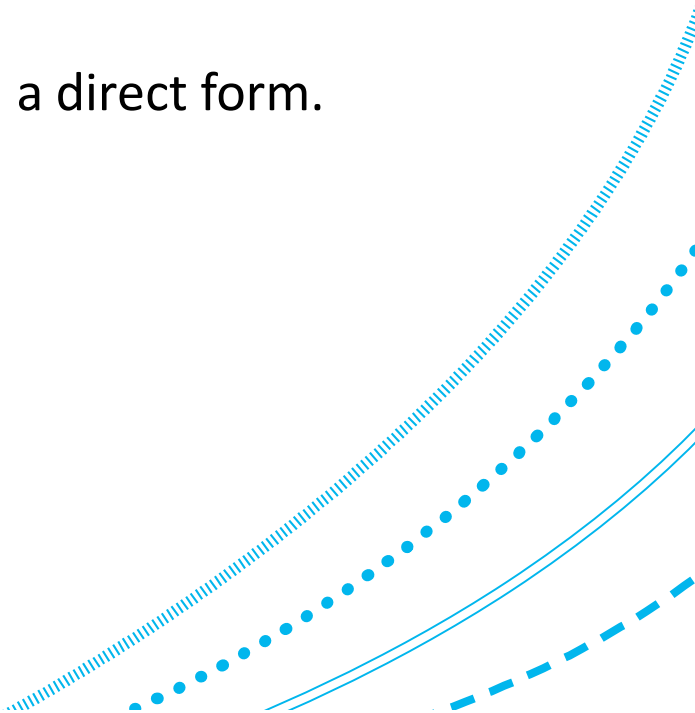
Access matrix example

subject\object	bill.doc	edit.exe	fun.com
Alice	-	{x}	{x, r}
Bill	{r, w}	{x}	{x, r, w}

1. bill.doc can be read and written by Bill. Alice has no access.
2. edit.exe can be executed by both Bill and Alice, but that's all they can do.
3. fun.com can be executed and read by both users, but only Bill can write to it.
4. Each cell contains exactly the permissions held by the corresponding subject on the corresponding object.

Access control matrix problem

1. Good that it gives very fine-grained control.
2. But permissions often change frequently.
3. Could be a big data structure.
4. In practice, usually difficult to implement and maintain in a direct form.



Indirect matrix implementations

1. Two basic ways:
    1. **Capabilities**:
        1. For every subject, *s*, keep a list of its access rights (what it can do to what).
            1. Basically, the row *s* of the matrix.
    2. **Access Control Lists (ACLs)**:
        1. For every object, *o*, keep a list of the access rights to it (who can do what to it).
            1. Basically, the column *o* of the matrix.
2. ACLs much more widely used explicitly, especially at OS-level.
3. But many systems have capability-like aspects.

Capabilities

1. Advantages:
    1. Capabilities can be stored `with' principals
    2. Subjects could pass on capabilities to other subjects.
        1. A capability can often be thought of as a ticket/token for access.
2. Disadvantages:
    1. Difficult to know who has access to a particular object
    2. Difficult to revoke permissions.
    3. Maintenance of integrity of capabilities: where are they stored and how are they communicated?
    4. If an attacker can steal or copy capabilities, then they can potentially defeat our access controls.

Capabilities (cont'd)

- Capabilities sometimes enter into an access control system by accident rather than by design. For example:
  - an application or service might run over SSL/TLS (described later);
  - this might use certified private keys from a public-key cryptography scheme for authentication;
  - authentication with appropriate keys may allow for certain operations;
  - keys and certificates might be shared.

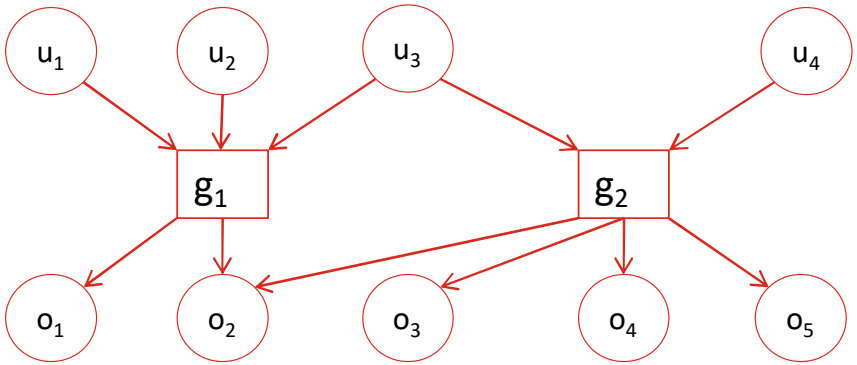
Access control list (ACL)

1. Very common in real systems, including operating systems.
2. Example ACLs:
    1. `bill.doc:  Bill: r,w;`
    2. `edit.exe:  Alice: x,r  ;  Bill: e;`
    3. `fun.com:  Alice: x,r  ;  Bill: x,r,w`

ACL (cont'd)

1. Advantages:
    1. Good when focus on managing access to particular objects is a central problem.
    2. That is, the relevant questions are of the form: ``For object A, who (or what) should have access?''
2. Disadvantages:
    1. Difficult to find all the permissions of a particular user
    2. Difficult to revoke their permissions
3. Needs abstractions to help with management.
    1. E.g. groups and roles below (and later).

Groups



1. A **group** is a list of principals.
2. This allows us to manage them collectively.
3. We can then define access rights for all members of the group, e.g. `Any member of group *G* can do operation *a* to object *o*'.
4. Group members have some of the same privileges.
5. Privileges for group members may change frequently, as may membership of the group.

Roles

1. A **role** is a fixed (or slow-changing) set of access permissions; subjects may then be assigned to roles.
2. Subjects may have more than one role.
3. More than one subject may have the same role. Examples:
    1. The officer on watch on the ship HMS Tadpole
    2. Acting manager of a bank
    3. Course organizer of CS1599
    4. Lecturer
4. **Role-based access control** (RBAC) is becoming very popular because it can match access control to organizational structure.

Groups and roles

1. Groups and roles are not thought of as identical. Although they have structural similarities, they are managed quite differently:
    1. A **group** is a collection of users/principals. The members of a group share privileges, which may change quite frequently.
        1. We assign users to (sets of) groups and we give permissions to groups.
    2. A **role** is a set of permissions (associated with a functional requirement within an organization) which one or more principals may assume for a given period of time according to a given access and authentication procedure. The permissions change infrequently, if at all.
        1. That is, we the role is (at any moment of time), essentially, defined in terms of its permissions; we then have a mapping of users to (sets of) roles.