Access control in *nix

- Users and groups
  - Any process is characterized by a user identifier (UID) and a list of group identifier (GUID)
  - UID equal to 0 represents root
  - `newgrp` permits to change the primary GUID
- Files and privileges
  - Every file is characterized by a pair UID/GUID (changed by `chown`/`chgrp`)
  - `ls -l` represents 9 privileges; their meaning depends on the file type (file or directory)
  - Also: sticky bit, setuid and setgid
  - Current file systems offer additional privileges (’man lsattr’)

Evaluation of access control in *nix

- Access control has the granularity of file
- No conditional authorizations
- *setuid* violates the least privilege principle
- Root is all-powerful and it is difficult to limit its responsibility
- Only positive authorizations
- Administration is based on root and owner

Access control in Windows NT

- Every subject is represented by a *Security Identifier* (SID)
- (Almost) Every object has a *security descriptor*
  - files, processes, threads, pipes, shared memory areas, registry entries, ...
- The security descriptor contains the owner SID, the group SID, a Discretionary ACL (DACL) and a System ACL (SACL)
- Each element in an ACL is an Access Control Element (ACE), an elementary authorization, with subject, action, type (allow, deny or audit) and several flags
- The action is a 32-bit access mask (only part of the bits have been specified); 16 bits are applicable to every object (they are divided into 3 families) and 16 are specific to the object type

Access tokens and request evaluation

- Each process or thread is associated with an *Access token*
- It contains: user's account SID, user groups SIDs, logon SID, privilege list, owner SID, primary group SID, default DACL
- There is support for impersonation
- Request evaluation:
  - The Access token is compared with the ACEs in the DACL
  - The first match of the ACE's SID with the SID in the Access Token determines the outcome
  - If there is no DACL $\longrightarrow$ open model
  - If there is no match within the DACL $\longrightarrow$ closed model
    I.e., a null DACL is different from an empty DACL
  - The SACL is evaluated in the same way and it specifies the actions that have to be logged

Comments on the access control mechanism

- The mechanism operates at a low level
- It is difficult to introduce abstract mechanisms to manage privileges
- It is not suitable to decentralized administration
- Accessing the DACL, it is possible to completely manage the access policy to the object

The DACL is set on object creation, typically using the DACL in the access token used by the creator

Privileges in Windows

- System privileges represent specific tasks
  - making a backup, debugging a process, increasing quotas, creating accounts, etc.
- Privileges can be considered as authorizations-without-objects
- They are represented in the access token
- Access tokens may be *localized*, i.e., they can depend on the network node in which they are created

Impersonation in Windows

- Impersonation permits to acquire a different “identity” for the execution of a specific task
  - Similar to the *setuid* and *setgid* mechanism
  - A privilege is needed to activate impersonation
  - The process/thread acquires a new Access Token
  - Win2k has introduced a *restricted token*, used to remove access rights

Access control inheritance in Windows

- Securable objects (the ones with a security descriptor) may contain other securable objects
  - E.g., directories/folders may contain files, registry keys may contain subkeys
- Security descriptors are automatically propagated from an object to all the objects contained within it
  - Before Win2k, ACLs of containing objects were copied by default at object creation
  - Since Win2k, inherited ACEs virtually appear at the end of the ACLs of contained objects (dynamic evaluation)
  - For each ACE it is possible to specify if:
    - ∗ The ACE has to be propagated
    - ∗ It has to be propagated recursively or only at the first level
    - ∗ It applies only to descendants

Concise evaluation of Windows access control

- Overall, a powerful mechanism that satisfies many requirements
- It is adequate as an implementation mechanism, it is too complex to be used directly
- There is an opportunity for the design of access control tools that
  - start from a high-level description of access requirements
  - automatically produce a correct representation using DACLs, SACLs, privileges, impersonation, inheritance, etc.

HTTP access control

- HTTP requests may require authorization (answer 401)
- Authentication is passed in the HTTP header, base64 encoded
- Two mechanisms
  - Basic (in HTTP 1.0): username/password in clear
  - Digest (in HTTP 1.1): MD5 of username, password and a nonce

Apache access control

- Access rights are represented by a `.htaccess` file per directory, or in the Apache configuration file within a `<Directory>` section
  - `AllowOverride` is typically set to `None`; it has to be set to `AuthConfig` or `All`
- Access rights are represented by a sequence of directives; *user*-based access control
  - *AuthType*: Basic or Digest
  - *AuthName*: The name of the *realm*; it represents a domain of resources, sharing the same authentication
  - *AuthUserFile/AuthDigestFile*: Location of the password file
  - *AuthGroupFile/AuthDigestGroupFile*: Location of the group file

Positive and negative autorizations in Apache

Authorizations can be positive or negative. `Order` directive

Users can specify an order that define how to interpret positive/negative authorizations. Three choices:

deny,allow negative authorizations are evaluated first and access is allowed by default. A requestor is granted access if it does not have any negative authorizations *or* it has a positive autorization.

allow,deny positive authorizations are evaluated first and access is denied by default. A requestor is denied access if it does not have any positive authorization *or* it has a negative authorization.

mutual-failure : A requestor is granted access if it does not have any negative authorizations *and* it has a positive autorization.

Example of `Order` directive

```
Order Deny,Allow
Deny from all
Allow from .elet.polimi.it
```

Apache access control - host based

- Options for `Allow` and `Deny`
  - `allow/deny from` *host* (DNS or IP mask)
  - `allow/deny from env=` *variable*
- `Require` directive
  - Options
    - ∗ `valid-user`
    - ∗ `user` *user list*
    - ∗ `group` *group list*,

Evaluation of Apache access control

- Strictly related with the hierarchical structure of the file system
  - Access control may operate at a higher level if pages are dynamically created
- It is a modern access control mechanism
  - Flexible resource granularity
  - Conditional authorizations (based on patterns)
  - Multiple policies
  - Positive and negative authorizations
  - Policy combination and restrictions on it

Access control in Java 2

- The focus is on the management of *mobile code*
- The security model of Java 1.0 was based on the construction of a *sandbox*, dedicated to the execution of downloaded code
  - Limited granularity (a single policy for all the code)
  - Programmers could reimplement the access control services, but this is impractical for many situations
- Java 2 introduced a novel Security Architecture, independent from the host operating system
- JAAS (Java Authentication and Authorizaton Service) is the environment containing the Security Architecture

Security Policy in Java

- It consists of a list of permissions (an ACL), global or user-specified
- Each entry describes a piece of Java code and permissions that are granted to it
- `grant` [`signedBy "`*signer_names*`",`] [`codeBase "`*URL*`"`] `{`
  `permission` *permission_class_name "target_name", "action",*
  `[signedBy` *"signer_names"*`];}`
- The piece of code is described by a URL, with support for implication
- Examples:

```
grant codeBase "http://www.example.org/classes/"
 {permission java.io.FilePermission "log" "write";};
grant {permission java.net.SocketPermission
       "localhost:1024-", "listen";};
```

Permissions in Java

- Abstract class `Permission` is specialized in many different ways
- Permissions typically have a *target* and an *action*
- Currently there are no negative permissions (for simplicity)
  - A future addition is possible, as there are no constraints to their introduction
- There are no permissions on single objects, only on classes
  - The rationale is that the policy lives across program runs, whereas objects live only during a program execution
- In Java 1.4, with the integration within JAAS, principals have been introduced for permissions

Access control evaluation in Java

- Permissions are not directly associated with classes
- Class `ProtectionDomain` realizes the link
  - The URL in the policy typically identifies many classes; they are all part of the same protection domain
- Access control is realized at 2 levels: `SecurityManager` and `AccessController`
  - `SecurityManager` can be specialized, whereas `AccessController` is final
  - The main method of `SecurityManager` is `checkPermission`
  - `checkPermission` of `SecurityManager` invokes `checkPermission` of `AccessController`

2-level access control in Java

- The presence of classes `SecurityManager` and `AccessController` satisfies two conflicting requirements
  - Allow a flexible evaluation policy
    - `SecurityManager` is redefined
  - Guarantee that a given permission is verified according to the standard model
    - The services of `AccessController` are directly invoked by the application

Run-time evaluation

- The Java execution environment presents an array of protection domains (classes typically belong to many domains)
- To evaluate permissions, the intersection is considered
- Method `doPrivileged` can be used to overcome this limit
  - it creates a separate execution environment, containing only the protection domain of the invoked class
  - It can be used to realize critical operations (e.g., change password)
  - It is analogous to the *setuid/setgid* mechanism of Unix, but it offers a finer granularity

XACML

- A novel proposal by OASIS
- Similar proposals: EPAL (by IBM), WS-Policy (MS-IBM-BEA)
- Important goals
  - Allow the central definition of access control policies, independent from the specific target
  - Allow the integration between separate mechanisms
  - Identify a canonical architecture for an access control service
  - Offer all the features of a modern access control solution: roles, conditional authorizations, negative authorizations, flexible policy combination, compatibility with multiple user authentication mechanisms, etc.
- An implementation by Sun, version 1.1, is available

XACML: an example (1)

```
<Policy PolicyId="ExamplePolicy"
         RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0
              :rule-combining-algorithm:permit-overrides">
 <Target>
  <Subjects>
   <AnySubject/>
  </Subjects>
```

XACML: an example (2)

```
   <Resources>
    <Resource>
     <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0
                                       :function:anyURI-equal">
      <AttributeValue DataType=
       "http://www.w3.org/2001/XMLSchema#anyURI">
        http://example.com/code/docs/developer-guide.html
      </AttributeValue>
      <ResourceAttributeDesignator DataType=
       "http://www.w3.org/2001/XMLSchema#anyURI"
       AttributeId=
       "urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
      </ResourceMatch>
    </Resource>
   </Resources>
  <Actions>
   <AnyAction/>
  </Actions>
 </Target>
```

XACML: an example (3)

```
<Rule RuleId="ReadRule" Effect="Permit">
 <Target>
  <Subjects>
   <AnySubject/>
  </Subjects>
  <Resources>
   <AnyResource/>
  </Resources>
```

XACML: an example (4)

```
 <Actions>
  <Action>
   <ActionMatch MatchId=
    "urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <AttributeValue DataType=
     "http://www.w3.org/2001/XMLSchema#string">
     read
    </AttributeValue>
    <ActionAttributeDesignator DataType=
     "http://www.w3.org/2001/XMLSchema#string"
     AttributeId=
     "urn:oasis:names:tc:xacml:1.0:action:action-id"/>
   </ActionMatch>
  </Action>
 </Actions>
</Target>
```

XACML: an example (5)

```
  <Condition FunctionId=
   "urn:oasis:names:tc:xacml:1.0:function:string-equal">
   <Apply FunctionId=
    "urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
    <SubjectAttributeDesignator DataType=
     "http://www.w3.org/2001/XMLSchema#string"
     AttributeId="group"/>
   </Apply>
   <AttributeValue DataType=
    "http://www.w3.org/2001/XMLSchema#string">
    developers
   </AttributeValue>
  </Condition>
 </Rule>
</Policy>
```

A concise evaluation of XACML

- Advantages
  - It is in a more advanced status than competing proposals
  - It is extremely powerful
  - The use of XML makes it adequate for the Web infrastructure
- Cons
  - The XML representation cannot be written by hand or interpreted directly by a user (development tools have not yet been implemented)
  - Political environment has an important role in its success
  - Native XML solution may be too expensive in some contexts (e.g., to control access to OS resources)