Systems Administration (ASIST)

Practical Theoretical Class 02

Users and groups.
Centralized administration.

Based on: A. Moreira, 2018/2019, ASIST Theoretical-practical classes

Users, groups and permissions

The concept of user is common to all multiuser operating systems, it is fundamental to distinguish different actors before the system. Each user has associated with them a set of **attributes** that constitute what is commonly referred to as a **user account**.

User accounts are stored by the operating system in appropriate databases. Different operating systems use different types of user accounts with different attribute types, so this is a difficult aspect to integrate.

Among the attributes that are common to all operating systems is the ***login name*** through which the user identifies himself with the system.

To each user the administrator can assign permissions (what he is allowed to do). The concept of *group account* makes this task easier since the permissions assigned to a group have an effect on all its members.

Permissions can be set as user rights associated with the user or group account, or as an *Access Control List* (**ACL**) associated with an object.

Unix systems – Users

On Unix systems the fundamental user database is the text file **/etc/passwd**. Other databases may be used in addition to this.

```
root:x:0:0:root:/root:/bin/bash
nobody:x:65534:65534:Nobody:/:/bin/sh
www:x:98:98:Gestor WWW:/users/home/www:/usr/bin/tcsh
i977805:x:2176:102:Joaquim Cardoso Morais:/users/home/i977805:/bin/csh
artur:$1$s2b9pyc6$46nIOl8G1fGrymmvsJejG/:1203:98:Artur:/users/home/artur:/bin/bash
```

The image above illustrates an example of the contents of this file, in this case with 5 accounts, starting with the root user (system administrator, has all permissions). Each line of the file corresponds to a user account and contains 7 attributes separated by the colon symbol:

- *Login-name* – unique name that identifies the user in the system.
- Password *hash*. The value **x** indicates that the *hash* is in **/etc/shadow** file.
- UID (*User IDentifier*) – unique number of 16 bits that identifies the user.
- Primary GID (*Group IDentifier*) – 16 bits number which indicates this user's primary group GID.
- Long name or description of the user.
- Home folder (*home*). Current folder after *login*.
- Initial program. Tipically a *shell* executed after *login* (if not filled, applies to default *shell*).

Unix systems – Users authentication

The password *hash* stored in the **/etc/passwd** file is used for authentication, when the user provides the *password* the system applies the *hash* function and compares the result with what is in **/etc/passwd**; being equal the authentication is valid.

The operating system requires the **/etc/passwd** file to be readable by all users, so it is concluded that it is not the ideal place to store user password *hashes*.

Note that a *hash* function is not reversible, so it is not possible to get the *password* from the *hash*, but having access to that information opens the possibility of brute force attacks attempting multiple random passwords and seeing if by chance after applying the *hash* function to any of them the output equals the stored value in **/etc/passwd**.

To avoid this vulnerability the password *hashes* in **/etc/passwd** can be replaced by the **x** symbol meaning that the *hash* is in the **/etc/shadow** helper file, and users cannot read this helper file.

Note that placing the *hash* in the **/etc/shadow** file does not prevent brute force attacks, it only makes them difficult as it is not possible for a user to compare the generated *hash* with the one stored in the file.

Unix systems – User groups

On Unix systems, user group accounts are stored in the text file **/etc/group**, other databases may be used in addition to this.

The format is identical to the file **/etc/passwd**, in the figure an example is shown. The attributes of each group are the group name, the next field (**x**) is not currently used, followed by the *Group IDentifier* (**GID**). Both group name and GID must be unique.

```
root:x:0:
bin:x:1:
nogroup:x:65534:
sshd:x:77:
users:x:100:
inf:x:102:
profs:x:98:
dom_users:x:1003:artur,i977805
```

The first line of the example defines the **root** group. Although group names and user names are completely independent, by default a group with the same name as that user's primary group is created for each user.

The last attribute is a comma-separated list of usernames that are members of the group (such as secondary group) in addition to their primary group.

A user can belong to a group in two ways: because it is their primary group (defined in the /etc/passwd file) or because they are in the group's member list (in the /etc/group file)).

Linux – User and group management

The management of local users and groups (defined in **/etc/passwd**, **/etc/group** and **/etc/shadow** files) must be performed through a set of standard Linux commands that exists in most distributions: **useradd** ; **usermod** ; **userdel** ; **groupadd** ; **groupmod** ; **groupdel** ; **chfn** ; **chsh** ; **passwd**; **pwconv**.

These commands have the advantage of having a proven solidity and perform some validations and ancillary operations.

Na alternative to use with caution is direct file manipulation, ie using a texto editor to change the **/etc/passwd** and **/etc/group** files. In this case there is a need to avoid any kind of inconsistency or overlap of identifiers. These files are read sequentially, if there is a duplicate identifier, only the first one will be valid.

Note that the **root** user (UID = 0) and **root** group (GID = 0) are critical to system operation, without them the system will not even be able to boot.

Windows systems – Users and groups

In Windows Server there are local user accounts and local groups only if the server is not integrated into any Windows domain, ie operating as a *Standalone Server*.

On a *standalone* Windows system, user accounts are stored in an operating system database called *Security Account Manager* (**SAM**). When a Windows system is integrated into a domain, it no longer uses local SAM and uses the domain's *Active Directory* database.

The number of attributes associated with Windows user accounts and groups is quite large and can even be expanded for *Active Directory*. One difference that stands out from Unix now is that unlike that one, on Windows systems **one group can be a member of another group**. Another difference is that on Windows systems user and group accounts are stored together and therefore it is not possible to have a username equal to a group name.

When Windows Server is installed, several user and group accounts that are critical to system operation are automatically created. The name of the user who has full administration rights is ***Administrator*** and belongs to the ***Administrators*** group (if English version of Windows).

User and group names

On **Windows** systems usernames and groups can have up to 64 characters. Some characters are prohibited, but spaces and periods may be used, Microsoft advises against using spaces for usernames. Usernames and groups can have upper and lower case letters but they ***are not case sensitive***.

On **Linux** systems the restrictions are bigger, not due to the operating system itself, but due to a wide range of open source software with which it is necessary to maintain compatibility. The recommendations are as follows:

- Not contain uppercase letters.
- Name must start with a lowercase letter or an *underscore*.
- The following characters can also be numeric digits or the minus sign. Additionally the last character can also be the dollar sign. Other characters including space are not allowed.
- Name must not exceed 31 characters in length.

Centralized administration

In a typical work environment, multiple authorized users should have access to a set of servers and other resources. It is obviously not feasible to maintain a separate user and group database locally for each.

The solution is to have a single central database of accounts that is accessible to all devices across the network using appropriate application protocols. Therefore, any change to the central account database has immediate effect on all equipment using it.

It's much easier, for users, to use just a single set of credentials and not a different one for each resource they access.

Centralizing accounts also gives users a **centralized authentication system**. Although credentials are the same across all features/services, user must authenticate to each.

A ***single sign-on*** **authentication system** (**SSO**) is even more convenient for users. As the name implies, the user only needs to login once, after which he has access to all resources. This is the kind of authentication we have on Microsoft Windows domains.

Microsoft Windows Domains

Windows domains came up with Windows NT Server, it's a ***single sign-on*** centralized authentication system..

Domain operation is centralized on a Windows Server machine with the ***Domain Controller*** (**DC**) role. DC contains database with domain user, group, and other constituent accounts. Since Windows 2000 Server, DC databases are *Active Directory* (**AD**) databases using *Lightweight Directory Access Protocol* (**LDAP**) standard.

Prior to the introduction of AD, each domain had a DC designated **PDC** (*Primary Domain Controller*) that contained the domain database. Only one PDC can exist in a domain, but other DCs can contain read-only copies of the database.

With the introduction of AD the concept of PDC disappears although it is still emulated for compatibility with older systems. An AD domain can contain multiple DCs, databases and control functions required(*Active Directory Domain Services* – **AD DS**) are distributed across the various existing DCs, some of them will have special functions (***operations masters***).

Active Directory logical structure - Tree

Active Directory (**AD**) is not limited to domains, there is a logical infrastructure that integrates domains.

AD domain names match qualified DNS domain names. One logical superstructure of domains that can be created is the ***Domain Tree***.

A tree corresponds to a **set of contiguous DNS domains**, ie a DNS domain and eventually DNS subdomains. The AD domain corresponding to the base (closest to the root) DNS domain that belongs to the tree is called ***tree root domain***.

An AD domain always belong to a tree, if it is the only domain tree is also the ***tree root domain***.

When a new domain is created, by *promoting* a Windows Server from ***Standalone Server*** to ***Domain Controller***, the new domain is either integrated into an existing tree if the DNS name matches a subdomain of another that already belongs to a tree, or else a new tree will be created with the new domain as root.

Active Directory logical structure - Forest

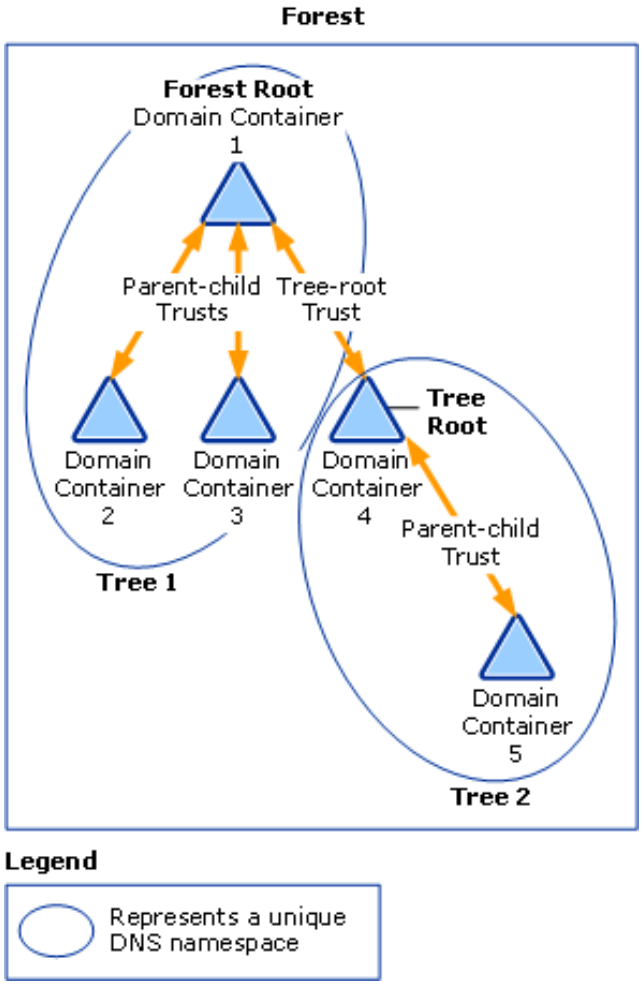
A forest is a set of trees, and therefore a set of domains, and **unlike the domains of a tree, there may be no relationship between the DNS names of the trees in a forest.**

When a new domain is created it must be integrated into an existing forest or a new forest must be created. The first domain to be added to a forest becomes the ***forest root domain*** and the forest is identified by the DNS name of that domain.

A forest establishes a trust relationship between all its domains.

Simply put this means that a user authenticated to one of the forest domains will have access to resources from other forest domains.

The forest boundary as a whole constitutes a *security boundary*. The forest can also be viewed as the highest level of AD logical organization.



Fonte: https://technet.microsoft.com

Active Directory Federation Services (AD FS)

This is a ***single sign-on*** (**SSO**) authentication system that allows you to exceed forest-defined trust limits.

Federation should not, however, be interpreted as being a logical AD level above forests.

It was designed to allow the integration of authentication between different organizations (***federation partners***) that will constitute a **federation**.

These services are intended for direct use by applications such as web services and use the internet securely to access specific authentication servers called ***AD FS servers***.

Users of organizations that are part of the same federation (***federation partners***) need to authenticate only once, after which they have access to applications (possibly web) made available by ***federation partners*** without the need for additional authentication.

Active Directory Domain Controllers

The logical structure of Active Directory is supported by a set of Windows Server machines called *Domain Controllers* (**DC**), which makes *Active Directory Domain Services* available (**AD DS**).

Each AD domain must have at least one DC, Microsoft recommends a minimum of two to ensure fault tolerance.

A Windows Server machine may find itself in one of 3 situations:

- After the initial installation, it is not integrated in any domain and is called **Standalone Server**.
- It can be integrated into a domain without being a Domain Controller, in this case it is called ***Domain Member***, just like workstations.
- Can be promoted to Domain ***Controller***, eventually creating a new domain, a new tree, and a new forest.

Depending on the context in which it is promoted to *Domain Controller* may also perform special functions in *Active Directory*, namely ***Global Catalog Server*** and ***Operations Master*** roles.

In each forest there is a *global catalog*, it is a database containing replicas of the most important parts of each AD database of each domain that integrates the forest. The *global catalog* is critical to enabling forest-level operations.

Active Directory Domain Controllers

The first domain controller to be created in a forest becomes the *global catalog server* containing this database, later other *domain controllers* in the forest can also become *global catalog servers.*

The different operations master functions must be performed by a single *domain controller* (*Single-Master Operations*) within the framework level to which they apply. Therefore:

- In a forest: in addition to the global catalog servers, there must be exactly one DC performing the ***Schema Master*** role and one DC performing the ***Domain Naming Master*** role.

- In a domain: there must be exactly one DC performing the ***Primary Domain Controller*** (***PDC***) ***Emulator*** role, one DC exercising the ***Infrastructure Master*** role, and one DC performing the ***Relative ID (RID) Master role.***

When multiple DCs exist in the same domain *Active Directory* synchronization is guaranteed (*Multimaster replication*) and fault tolerance is provided.

You can also set a DC to be *read-only* (**RODC**). Operations such as a search can be performed by the RODC, operations involving changes are not possible. They can be installed in remote locations with poor security and DC connection issues.

Active Directory physical structure - Sites

The logical infrastructure of domains, trees, and forests has no reflection of the underlying physical reality. To store physical location information the concept of ***Site Object*** is introduced in the forest context. So from a physical point of view **the forest is divided into sites**.

This only happens because the forest is the top of the logical hierarchy, there is no direct relationship between Sites (physical concept) and domains, trees and forests (logical concepts).

Therefore: **a domain may be spread across multiple sites**, also: **a site can be part of multiple domains.**

The relevance of splitting the forest into sites relates to optimizing replication mechanisms between *domain controllers* within the forest.

The definition of the Sites is based on the existing DC network settings. Each ***Site Object*** is associated with one or more ***Subnet Objects***, a subnet object is a representation of a level 3 (IPv4 or IPv6) network. Other objects such as servers and workstations are associated with Sites, with this information it is possible to determine for a workstation which is the DC of the desired domain that is physically closest, possibly in the same Site.

Active Directory – Organizational Units (OU)

Logically, a domain can be divided into a tree structure of ***Organizational Units*****(OU).**

The purpose of the OU is to facilitate organization and administration. An OU may contain domain objects such as users, groups, computers, and even other OUs.

OUs should reflect the hierarchical or logical structure of the organization, each OU can for example represent a department or section, it is important that there is some logical relationship between the objects that are placed in the same OU.

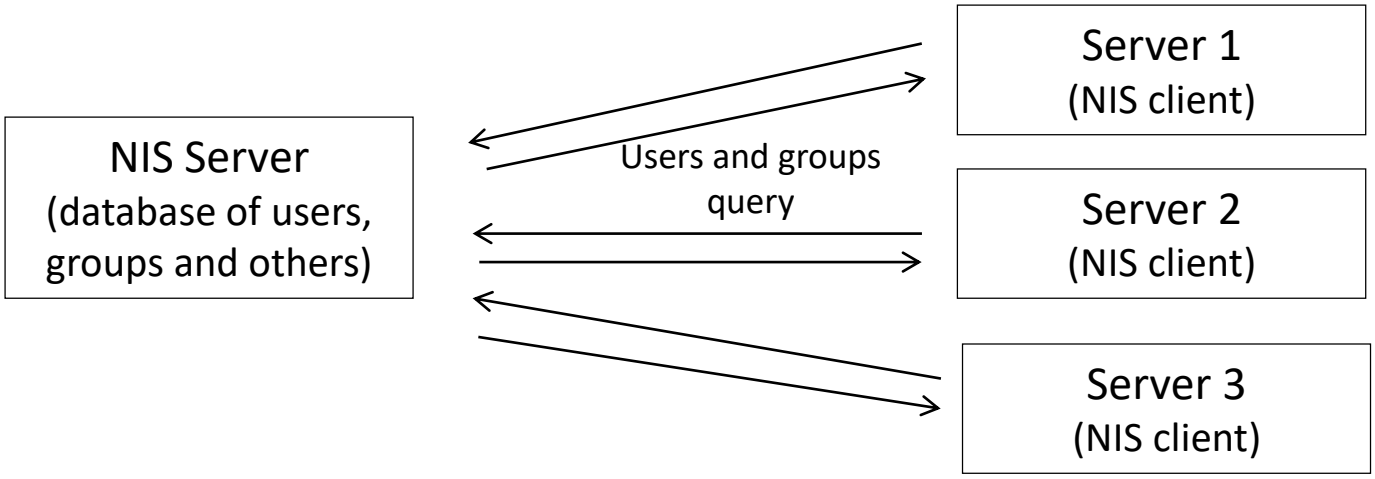
From an administrative point of view, two uses stand out:

- A ***Group Policy*** can be applied to one OU having effect on all objects contained in the OU.
- Users and groups can be given permissions on an OU. This way you can for example create an administrator of an OU with permissions to create users and groups in that OU only.

Unix - diversification of system repositories

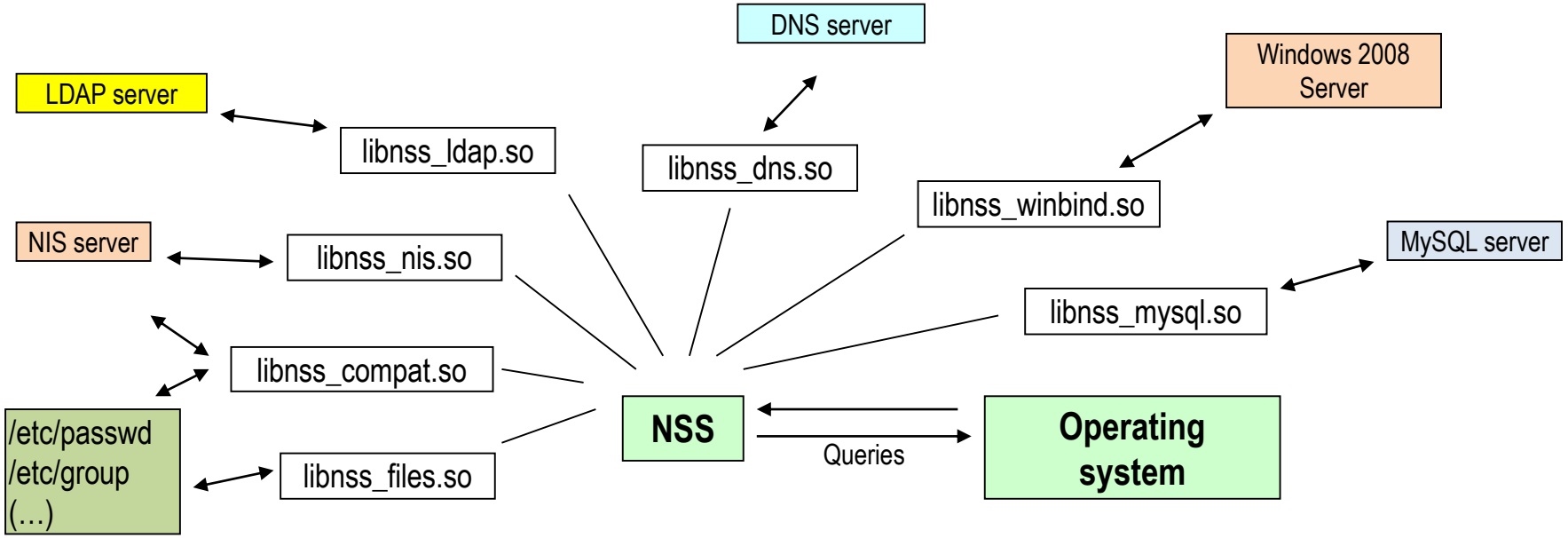
On Unix systems early on there was a need to diversify and centralize repositories of system configuration information (among others user and group databases), this need led to the development of alternative systems, one of the most important being the *Network Information Service* (**NIS**), also known as the *Yellow Pages*.

Centralizing these repositories has huge advantages from the point of view of managing a server pool. If all servers use the same central repository the administrator can manage all information in one place, users and groups will be the same on any of the servers in the pool..



Name Service Switch (NSS)

NSS is a modular system that allows you to integrate various types of system information repositories into Linux systems. Adding a new module (dynamic library) adds support for a new repository type.



Not all modules are intended for user and group databases, for example *libnss_dns* is commonly used for hostnames.

The *libnss_compat* module implements a pre-NSS system compatibility mode by querying local files and additionally using NIS servers.

Name Service Switch configuration

NSS configuration consists of defining the order in which the various available repositories (modules) will be used. For security and reliability reasons, preference should be given to local repositories. The texto file **/etc/nsswitch.conf** contains the configuration.

For each type of information defines the search sequence. Example:

```
bash-3.00$ cat /etc/nsswitch.conf
#
passwd:     files ldap winbind nis
shadow:     files nis
group:      files ldap winbind nis

hosts:      files nis dns winbind
```

In the example the user settings (***passwd***) will be searched in the following order:

1° Local files(***files***)

2° LDAP servers (***ldap***)

3° Windows servers (***winbind***)

4° NIS servers (***nis***)

The first one found will be used, ending the search

Name Service Switch configuration

NSS configuration itself is quite simple, however, many of the NSS modules used require specific configuration.

For example, the *libnss_ldap* module that lets you get information from database servers through *Lightweight Directory Access Protocol* (**LDAP**) usually uses the configuration file **/etc/ldap.conf**. In this file, among other things, will have to be specified:

- IP address or DNS name of the LDAP server
- DN (*Distinguish Name*) from the root of the LDAP database
- LDAP protocol version

When integrating remote user and group repositories we should be aware that the above mentioned Linux set of user and group management commands only works with local accounts stored in the **/etc/passwd**, **/etc/shadow** and **/etc/group** files.

In conjunction with the NSS *Pluggable Authentication Module* (**PAM**) is also commonly used, the two are not inseparable either can work without the other. Among other things, PAM handles the user login process. When users are in remote repositories PAM has to resort to them.

Linux Pluggable Authentication Module (PAM)

Linux-PAM is used by all current Linux distributions. It is a modular library used by applications and services that process user inputs and outputs in the system.

PAM handles 4 types of tasks (***management groups***):

- **account** – allows to perform validations before authentication (Examples: access time, source address) and after authentication (Examples: user allowed access, password expired).
- **auth** – this is authentication itself, usually via *username/password*.
- **password** – user changing their password.
- **session** – registration and preparation of the environment. Records user input after successful authentication, prepares desktop for user. It can be invoked again at *logout*, for example to log user out.

There are a variety of modules to implement these tasks, many of them specifically designed to interface with remote repositories. Some modules can be used for various types of tasks, others are specific to certain tasks.

Linux-PAM and Linux authentication

The traditional form of user authentication in UNIX is to compare the *hash* of the password that was provided by the user with the *hash* that is stored in the user's account (/etc/passwd or /etc/shadow).

With the diversification of user account repositories, the way authentication is done depends on the type of repository. The PAM system solves the problem, as NSS is typically modular for each NSS module there is a PAM module for the same repository type. For example for LDAP there is **libnss-ldap** and **pam-ldap**.

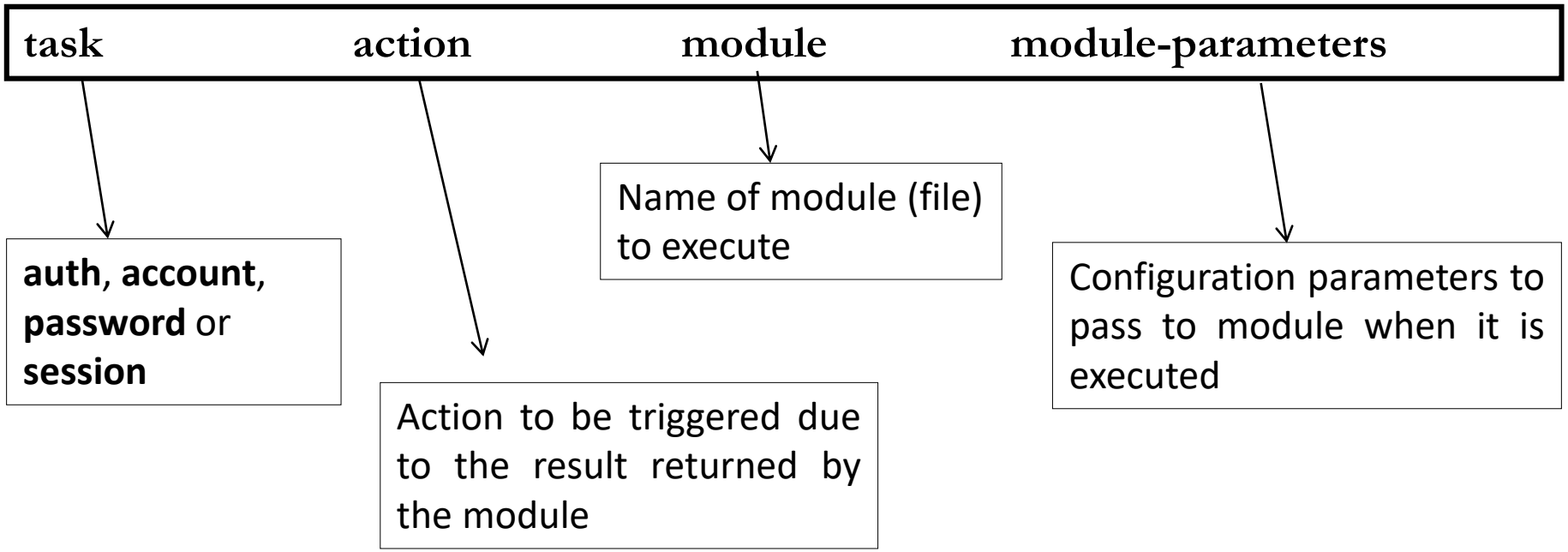
With PAM the forms of authentication can be easily diversified and are not limited to the hash types supported in /etc/passwd and /etc/shadow files.

Direct provision of the *username/password* pair to the authentication system is known as **PAP** (*Password Authentication Protocol*), but should only be used through secure communication channels.

Challenge Handshake Authentication Protocol (**CHAP**), often also called password protected authentication, never transmits the readable password. CHAP cannot be implemented through traditional Unix *hashes*, but can be implemented using appropriate PAM modules.

Linux-PAM - configuration

Configuring the PAM system consists of defining the sequence of modules to be used for each type of task, this is achieved with text files, usually resident in the **/etc/pam.d/** folder, where each line represents the invocation of a module in context of a task in the form:



PAM modules can return multiple results, zero value (**PAM_SUCCESS**) indicates total success, other values indicate various error types. The action to be performed is defined according to this value returned by the module.

Linux-PAM – configuration - action

task	action	module	module-parameters

required – on success continues processing of the following modules, on failure also continues processing on the following modules, but the end result will be failure.

requisite – on success continues processing of the following modules, on failure immediately terminates resulting in failure.

sufficient – on success and no previous required failures, returns success and ends chain. In case of failure it records an *optional* fault and continues processing the chain.

optional – on success continues chain, on failure registers an *optional* failure and continues processing chain.

Linux-PAM – configuration - action

The action to take can be specified on a case by case basis for each possible value returned by the module:

task	action	module	module-parameters

[value1=action1 value2=action2 value3=action3 …]

Where **value** represents the returned result, for example **success** or **default** and **action** the action to take in this case, the actions may be among others:

ok – marks a success and continues to process the following modules

done – marks a success and finishes processing with success

die – signals a failure and ends processing with failure

{*NN*} – marks a success, ignores the following *NN* modules (lines) and continues processing on *NN+1* module (line). ({*NN*} is an integer greater than zero)

The following equivalences can be referred to between ways of specifying the action:

required	⇔	[success=ok new_authtok_reqd=ok ignore=ignore default=bad]
requisite	⇔	[success=ok new_authtok_reqd=ok ignore=ignore default=die]
sufficient	⇔	[success=done new_authtok_reqd=done default=ignore]
optional	⇔	[success=ok new_authtok_reqd=ok default=ignore]

Linux-PAM – some common modules

pam_unix.so	Can be used for any task, implements traditional pre-PAM UNIX functionality.
pam_deny.so	Can be used on any task, always returns failure. Used for testing or ending a sequence.
pam_env.so	**Auth** and **session** module that allows manipulating environment variables according to a configuration file, usually "/etc/security/pam_env.conf".
pam_mail.so	**Auth** and **account** module that checks mail at login.
pam_ldap.so	LDAP repository interface module, supports **auth**, **account** and **password** functionality.
pam_issue.so	**Auth** module that allows displaying a message before user authentication.
pam_motd.so	**Session** module that displays a message upon system login (***message of the day***).
pam_nologin.so	**Auth** module that always returns success for the administrator and returns success for other users unless the file **/etc/nologin** exists.
pam_listfile.so	Can be used on either chain, allows success or failure to be returned due to a list resident in a file, for example with a list of authorized users.
pam_cracklib.so	**Password** module that checks the strength of the new password that the user intends to use. In addition to checking it in the system dictionary you can also do some additional checks.
pam_faildelay.so	**Auth** module that allows you to define the time that elapses since the user fails authentication until the error message is displayed.

Linux-PAM – use by applications

PAM configuration is independent for each application, each application is identified by a **service name** provided to the API by the application. In the **/etc/pam.d/** folder there must be a file with the service name provided, this will be the configuration file used.

Example for SSH service that identifies API with **sshd** name, file **/etc/pam.d/sshd**:

```
@include common-auth
account  required     pam_nologin.so
account  required     pam_access.so
@include common-account
session    required     pam_loginuid.so
session    optional     pam_keyinit.so force revoke
@include common-session
session    optional     pam_motd.so  motd=/run/motd.dynamic
session    optional     pam_motd.so noupdate
session    optional     pam_mail.so standard noenv
session    required     pam_limits.so
session    required     pam_env.so user_readenv=1 envfile=/etc/default/locale
@include common-password
```

As can be seen most of the settings is in included files that are shared by several services.

Linux-PAM – example /etc/pam.d/common-auth

```
# /etc/pam.d/common-auth - authentication settings common to all services


auth    [success=2 default=ignore]      pam_unix.so nullok_secure
auth    [success=1 default=ignore]      pam_ldap.so use_first_pass
auth    requisite                       pam_deny.so
auth    required                        pam_permit.so
auth    optional                        pam_cap.so
```

This file is intended to be included in the **auth** definitions of the various services through the directive **@include common-auth.**

We can see that a user will be successfully authenticated if authentication is successful via passwd / shadow (**pam_unix.so**) files or authentication is successful via **bind** (login) in the LDAP repository.(**pam_ldap.so**). Failing both will reach the **pam_deny.so** module and authentication will fail.

The configuration information of the **pam_ldap.so** module is the same as that used by the NSS **libnss_ldap** module, ie the file **/etc/ldap.conf.**