SSSD

Introducció:

A Linux es pot utilitzar el dimoni SSSD per autenticar l'usuari contra servidors LDAP remots (combinats amb servidors Kerberos o no) a més d'identificar-los. Mitjançant SSSD tot el procés d'autenticació/identificació de l'usuari es produeix al client d'una forma homogènia i coherent, independentment de la distribució Linux allà instal.lada.

> NOTA: A més d'autenticar-se contra servidors LDAP "pelats" o combinats amb Kerberos, el dimoni SSSD també permet autenticar usuari contra sistemes integrals empresarials com ara FreeIPA o l'Active Directory (els quals internament es basen en LDAP/Kerberos). En canvi, no pot treballar (encara) amb SGBDs o servidors RADIUS com a font d'autenticació/identificació

Un gran avantatge de fer servir SSSD en comptes dels mòduls PAM/NSS "standalone" individualment és que aquest dimoni guarda una catxé (a la carpeta "/var/lib/sss/db") que permet l'accés "offline" al/s sistema/es client/s (sense haver de tenir cap compte local diferent) en el cas de que el servidor LDAP/Kerberos remot no estigui disponible en aquell moment o, en el cas que sí, disminuir la seva càrrega.

> NOTA: El dimoni SSSD també ofereix una interfície D-Bus per tal de poder interactuar amb el sistema mitjançant aquest mètode i així obtenir, per exemple, informació extra sobre l'usuari (tal com veurem).

El dimoni SSSD proporciona els seus propis mòduls PAM i NSS per connectar amb servidors LDAP/Kerberos i LDAP, respectivament, els quals són diferents dels mòduls "standalone" pam_ldap/pam_krb5 i nss_ldap ja coneguts. Això és perque SSSD ofereix funcionalitats extra com ara un framework conjunt i llibreries homogènies, un mecanisme de catxé de credencials -similar al que oferia el dimoni *ncsd*, ja obsolet-, una interfície D-Bus, etc. Per a què els programes d'inici de sessió (*login*, *gdm*, *su*, *sudo*, *ssh*, *etc*) facin servir els mòduls PAM/NSS propis de SSSD haurem de realitzar un conjunt de configuracions al sistema que explicarem tot seguit.

> NOTA: A Ubuntu cal instal.lar el paquet "sssd" (és a dir, el dimoni SSSD no ve preinstal.lat "de fàbrica"). Afortunadament, en instal.lar aquest paquet ja s'instal.laran automàticament com a dependència la resta de paquets que poguem necessitar, com ara els connectors amb els diferents providers: "sssd-ldap", "sssd-krb5", "sssd-ipa", etc. A Fedora tots aquests paquets ja vénen instal.lats "de fàbrica".

Configuració de SSSD:

SSSD reads the configuration files in this order: first it reads the primary "**/etc/sssd/sssd.conf**" file and then, other *.conf files in "/etc/sssd/conf.d" (in alphabetical order). If the same parameter appears in multiple configuration files, SSSD uses the last read parameter. This allows you to use the default "/etc/sssd/sssd.conf" file on all clients and add additional settings in further configuration files to extend the functionality individually on a per-client basis. Anyway, all of these files must be owned by root:root and have 600 permissions

Authentication and identity providers are configured as "domains" in the SSSD configuration file. You can configure multiple domains for SSSD but at least one domain must be configured, otherwise SSSD will not start. A single domain can be used as:

*An authentication provider (for authentication requests)
*An identity provider (for user information)
*An access control provider (for authorization requests). We won't study them here.
*A combination of these providers (if all the corresponding operations are performed within a single server)

> NOTA: There's too what is called a "proxy" provider. This provider works as an intermediary relay between SSSD and resources that SSSD would otherwise not be able to use. When using a proxy provider, SSSD connects to the proxy service, and the proxy loads the specified libraries. Using a proxy provider, you can configure SSSD to use alternative authentication methods, such as a fingerprint scanner, entre otras cosas.

In "sssd.conf" file there must be at least a general *[sssd]* section and one section for each defined domain. Below *[sssd]* section we should keep in mind this line (more information in *man sssd.conf*):

-Line "*domains=*" : Specifies a comma-separated list of defined domains in order you want them to be queried. So if a domain doesn't appear in this list, won't be queried although it is defined in "sssd.conf" file

Sections corresponding to defined domains must be titled like this *[domain/xxx]* where "xxx" can be any string. Below a domain section we should keep in mind these lines (more in *man sssd.conf*):

-Line "*auth_provider=*" : Sets the authentication provider used for the domain. Possible values are: "ldap", "krb5", "ipa", "ad" or "proxy", among others. Depending on the choosen provider, there will be necessary write more specific lines (see *man sssd-ldap , man sssd-krb5, man sssd-ipa* or *man sssd-ad* for details).

-Line "*id_provider=*" : Sets the identity provider used for the domain. Possible values are: "ldap", "files", "ipa", "ad" or "proxy". Depending on the choosen provider, there will be necessary write more specific lines (see *man sssd-ldap , man sssd-files, man sssd-ipa* or *man sssd-ad* for details).

-Line "access*_provider=*" : Sets the access provider used for the domain, which is used to evaluate which users are granted access to the system. Even if a user successfully authenticates, if they don't meet the criteria provided by the access provider, their access will be denied. Possible values are: "ldap", "krb5", "simple", "ipa", "ad", "proxy", "deny" (always denies access) or "permit" (by default, always permits access). Depending on the choosen provider, there will be necessary write more specific lines (see *man sssd-ldap , man sssd-krb5, man sssd-simple, man sssd-ipa* or *man sssd-ad* for details).

NOTA: The "simple" access provider allows or denies access based on a list of user names or groups. It enables you to restrict access to specific machines. For example, on company laptops, you can use the simple access provider to restrict access to only a specific user or a specific group. Other users or groups will not be allowed to log in even if they authenticate successfully against the configured authentication provider.

```
[domain/domain_name]
...
access_provider = simple
simple_allow_users = user1, user2
#simple_allow_groups = group1
```

If the access provider you are using is the "ldap" one, you can also specify an LDAP access control filter that a user must match in order to be allowed access to the system.

```
[domain/domain_name]
...
access_provider = ldap
ldap_access_filter = (memberOf=cn=allowedusers,ou=Groups,dc=example,dc=com)
#ldap_access_order = filter, host, authorized_service
```

NOTA: La directiva *ldap_access_order pot tenir eixos valors (es poden indicar per ordre d'aplicació):*
**"filter" : S'usa el filtre indicat a ldap_access_filter com a criteri. Valor per defecte*
**"host" : S'usa el valor de l'atribut "host" en el directori com a criter*i
**"authorized_service" : S'usa el valor de l'atribut "authorizedService" en el directori com a criteri*
**"expire" : S'usa el valor de la directiva ldap_account_expire_policy com a criteri*

-Line "*chpass_provider=*" : Sets the provider which should handle change password operations for the domain. Possible values are: "ldap", "krb5", "ipa", "ad", "proxy" or "none" (this explicitly disables the ability to change passwords). If this line not specified, then the value specified in "auth_provider" line is used by default.

-Line "*sudo_provider=*" : Sets the provider which should validate *sudo* operations for the domain. Possible values are: "ldap", "ipa", "ad" or "none" (this explicitly disables *sudo*). If this line not specified, then the value specified in "id_provider" line is used by default. See man *sssd-sudo* for details

Note SSSD does not cache user credentials by default. So when processing authentication requests, by default SSSD always contacts the identity provider; if the provider is unavailable, user authentication fails. To ensure that credentials are stored locally after successfully authenticating the users so they can keep authenticating even when the identity provider is unavailable, you should enable local credential caching adding the line *cache_credentials=true* below the corresponding domain section. By default, the credential cache never expires: if you want sssd to remove cached credentials, *account_cache_expiration=* line will cause them to expire after the number of days specified.

> NOTA: When a user attempts a *sudo* operation, we already know that SSSD contacts the identity/sudo provider defined in the domain to obtain the required information about the current *sudo* configuration. If *cache_credentials* line is set to *true*, it will store that *sudo* information in a cache too, so that users can perform *sudo* operations even when the LDAP/IPA/AD server is offline.

Optional *[pam]* and/or *[nss]* sections can appear in "sssd.conf" file, too. They could be used to configure some aspects of the SSSD's PAM or NSS modules' behaviour, respectively. For instance, if you wanted to configure a time limit (in days) for how long SSSD allows offline authentication if the identity provider is unavailable, you could use the "*offline_credentials_expiration=*" line below the *[pam]* section to specify that time limit. Other interesting lines to specify below *[pam]* section are "*offline_failed_login_attempts=*", "*offline_failed_login_delay=" or "reconnection_retries=*", among others.

Alike, below *[nss]* section some interesting lines which deserves some attention are "*filter_groups=*" and "*filter_users=*" (which exclude certain users -by default, only "root"- from being fetched from the NSS backend; this is particularly useful for system accounts.; this option can also be set per-domain or include fully-qualified names to filter only users from the particular domain), "*entry_cache_timeout=*" (which specifies how many seconds -by default 5400- should sssd consider entries valid before asking the NSS backend again), "*entry_cache_nowait_percentage=*", "*override_homedir=*", "*override_shell=*", "*allowed_shells=*", etc.

*Example 1 (a LDAP server as both authentication and identity provider):

A suitable "sssd.conf" file could be like this:

```
[sssd]
domains=pepito
[domain/pepito]
auth_provider=ldap
id_provider=ldap
ldap_uri=ldaps://ldapserver.example.com
ldap_search_base=dc=example,dc=com
```

> NOTA: SSSD always uses an encrypted channel for authentication, which ensures that passwords are never sent over the network unencrypted but forces the LDAP server to run over TLS. If this LDAP server used a self-signed certificate, you must add the line "*ldap_tls_reqcert=allow*" in the "sssd.conf" file so that client doesn't reject it. If this LDAP used a ca-signed certificate instead, you should add the line "*ldap_tls_cacert = /etc/pki/tls/certs/ca-bundle.crt*" instead (pointing to the right CA's bundle file in each case). Anyway, if you wanted identity lookups (such as commands based on the *id* or *getent* utilities) were also encrypted (or if LDAP server only runs over TLS), you should add the "*ldap_id_use_start_tls = true*" line.

> NOTA: If LDAP server's configuration requires that queries to LDAP directory have to be authenticated, you should add the "*ldap_default_bind_dn= cn=admin*" and "*ldap_default_authtok= 12345*" lines to specify which user ("cn=admin" in this example) and password ("12345" in this example) should be used to do these queries, respectively.

*Example 2 (a KDC server as authentication provider and a LDAP server as identity provider):

A suitable sssd.conf file could be like this:

[sssd]
domains=pepito
[domain/pepito]
auth_provider=krb5
krb5_server=kdc.example.com
krb5_realm=EXAMPLE.COM
id_provider=ldap
ldap_uri= ...
ldap_search_base= ...

NOTA: If the Change Password service is not running on the KDC specified in "krb5_server" or "krb5_backup_server" lines, you can use the "krb5_passwd" line to specify the server where the service is running or a chpass_provider instead.

Ús de SSSD:

Now SSSD is configured, you should tell the system's applications to use it.

1.-In order to use authentication providers via PAM, you could manually edit "/etc/pam.d/system-auth" and/or "/etc/pam.d/password-auth" files (in Fedora) or "/etc/pam.d/common-*" files (in Ubuntu) to add several lines like "*auth sufficient pam_sss.so ...*" or similar, all written in their right places. However, in order to minimize human errors manually editing PAM's configuration files, Fedora provides the *authselect* command, which does these changes for us. You just have to execute this command: ***sudo authselect select sssd with-mkhomedir with-sudo***

What former *authselect* command does is to select a "profile" (specifically, the "*sssd*" profile) with some arguments "*with-...*". Running *authselect list* you can see available profiles in your system; another profile you will have installed by default is the "winbind" one, which modifies "/etc/smb/smb.conf" file to be able to connect to Windows AD servers,. If you want to know which is the current selected profile, you can run the *authselect current* command. To get more information about possible case uses and options of the "sssd" profile, execute *authselect show sssd* To know all the possible arguments of authselect command, execute *authselect help*.

Els paràmetres "*with-...*" que la comanda *authselect select sssd ...* admet es tradueixen en opcions personalitzades concretes que s'escriuran adientment dins els arxius /etc/pam.d/* que toqui (per saber quines són aquestes opcions en concret, es pot fer un *diff* entre la versió dels fitxers /etc/pam.d/* prèvia i posterior a executar *authselect*). Concretament, el paràmetre *with-mkhomedir* enables automatic creation of home directories for users on their first login, el paràmetre *with-sudo* allows *sudo* to use SSSD as a source for sudo rules in addition of "/etc/sudoers" (it's not sufficient specifying only the sudo_provider in sssd.conf) . Però n'hi ha d'altres: el paràmetre *with-ecryptfs* enables automatic per-user ecryptfs, el paràmetre *with-faillock* enables account locking in case of too many consecutive authentication failures,, el paràmetre *with-pamaccess* enables checking access.conf during account authorization, el paràmetre *without-nullok* don't add "nullok" parameter to pam_unix.so lines, etc.

Anyway, either you modify PAM's configuration files manually or via *authselect* utility, after that you should restart the SSSD daemon (*systemctl restart sssd.service*) to let changes take effect.

NOTA: As a system administrator, you can modify one of the default profiles (like the "sssd" one) to change its behaviour when applying it. To do so, first you should modify the desired file/s ("system-auth", "password-auth" or "user-nsswitch.conf") located below the "/etc/authselect" directory. Then, you should run the *authselect select sssd* command again to transfer changes made in that "/etc/authselect/*" files to effective files (that is, to "/etc/pam.d/*" ones and "/etc/nsswitch.conf"). Alternatively, another way to do the same would be:

1.Select an authselect profile (for instance, *authselect select sssd*)
2. Edit the /etc/authselect/* files
3. Apply the changes executing: *authselect apply-changes*

NOTA: If adjusting a ready-made profile by adding one of the *authselect select* command-line options described above or even modifying this ready-made profile by changing the "/etc/authselect/*" files is not enough for your use case, you can create your own custom profile. To do so, see *man authselect-profiles* and this article https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/html/using_authselect_on_a_red_hat_enterprise_linux_host/using-authselect#creating-own-authselect-profile

1BIS.-In order to use identity providers via NSS, you could manually edit "/etc/nsswitch.conf" file to leave it like this...:

passwd: sss files
shadow: sss files
group: sss files
netgroup: sss files
services: sss files

...but *authselect* can take care of this, too. In fact, running former *sudo authselect select sssd ...* command, this step was already done. Anyway, either you modify "/etc/nswitch.conf" file manually or via *authselect* utility, after that you should restart the SSSD daemon (*systemctl restart sssd.service*) to let changes take effect.

2.-S'han d'activar i posar en marxa els "sockets" PAM i NSS de SSSD i el propi servei SSSD així:

sudo systemctl enable oddjobd && sudo systemctl start oddjobd
sudo systemctl enable sssd-pam.socket && sudo systemctl start sssd-pam.socket
sudo systemctl enable sssd-nss.socket && sudo systemctl start sssd-nss.socket
sudo systemctl enable sssd.service && sudo systemctl restart sssd.service

El servei "oddjobd" és necessari per a què funcioni l'opció *with-mkhomedir* de *authselect*. Els "sockets" són "miniprogrames" depenents del servei SSSD principal que es mantenen a l'escolta per detectar qualsevol petició PAM o NSS feta en el sistema client i llavors executar en aquell moment (i només en aquell moment) el procés adient per realitzar aquesta petició. La gràcia dels "sockets" és que permeten no haver de tenir en marxa aquests processos permanentment sinó només en l'instant que són necessaris.

NOTA: SSSD has some responders which don't have to be running all the time, but could be *socket-activated or dbus-activated* Making these responders socket-activated or dbus-activated would provide a better user experience, as these services could be started on-demand when a client needs them and exit after a period of inactivity. If any system didn't support this kind of behaviour (all that run Systemd do), the administrator should have to explicitly list all the services that might be potentially needed in the *services=* line of [sssd] section (like this: *services=pam,nss*) and those processes will be running all the time, then.

3.-Per comprovar la configuració realitzada, es poden executar vàries comendes com ara ***id usuari*** o ***getent passwd usuari*** , les quals, si tot va bé, mostraran la informació obtinguda del servidor LDAP que hi hagi l'usuari indicat, confirmant que la consulta NSS com a mínim funciona. A partir d'aquí, es pot probar ***su -l usuari*** per veure si l'autenticació funciona.

NOTA: Per saber si l'usuari és local o no, es pot executar la comanda: *getent passwd -s sss usuari* (només es mostrarà si és remot). O també *getent passwd usuari@domini*

Another way to check if new configuration in client has been taken into account, is by using the ***sssctl user-checks usuari*** command. This *sssctl* command works thanks to having installed the "sssd-tools" and "libsss_simpleifp" packages and has many other debug options, like *sssctl config-check, sssctl domain-list, sssctl domain-status domini, sssctl user-show usuari, sssctl group-show grup, sssctl logs-remove, sssctl cache-remove, sssctl restore-local-data* (que serveix per restaurar el backup de la catxé), etc.

NOTA: Per a què funcioni la comanda *sssctl* prèviament cal afegir sota la secció *[sssd]* del fitxer "sssd.conf" la línia *services=ifp* . Aquesta línia és necessària per habilitar un canal de comunicació intern entre la comanda *sssctl* i el servei SSSD (en concret, aquesta línia serveix per iniciar automàticament el servei *sssd-ifp*, responsable d'aquesta comunicació, cada cop que s'executi *systemctl start sssd)*.

SSSD uses by default a number of log files to report information about its operation, located in the "/var/log/sssd" directory. SSSD produces a log file for each domain, as well as an "sssd_pam.log" and an "sssd_nss.log" file. If something goes wrong, you should look up there to troubleshoot. You can change the log level accurancy assigning from 0 (report only critical errors) to 9 (report all traces) value to the *debug_level=* line (below any particular domain section in "sssd.conf" file).

Truc (com esborrar la catxé):

The cached results can potentially be problematic if the stored records become stale and are no longer in sync with the identity provider, so it is important to know how to flush the SSSD cache to fix various problems and update the cache.

The cache can be cleared with the *sss_cache* utility which is used for performing cache cleanup by invalidating records in the SSSD cache. Invalidated records must be reloaded fresh from the identity provider server where the information actually resides, such as FreeIPA or Active Directory for example. Knowing that the -E flag can be used to invalidate all cached entries (with the exception of sudo rules), you could do: *sss_cache -E* Alternatively we can also simply invalidate a specific user only from the cache with the *-u* flag, followed by the account username: *sss_cache -u usuari*

While using the sss_cache command is preferable, it is also possible to clear the cache by simply deleting the corresponding cache files, stored in /var/lib/sss/db directory. Before doing this it is suggested that the SSSD service be stopped (*sudo systemctl stop sssd*). After this we want to delete all files in cache: *rm -rf /var/lib/sss/db/** . Once complete we can start SSSD back up again (*sudo systemctl start sssd*). SSSD should now start up correctly with an empty cache, any user login will now first go directly to the defined identity provider for authentication, and then be cached locally afterwards.

Note it’s recommend to only clear the cache if the identity provider servers performing the authentication within the domain are available, otherwise users will not be able to log in once the cache has been flushed.