

KERBEROS, LE SSO UNIVERSEL

1- PRÉSENTATION ET DÉPLOIEMENT

par Guillaume Rousse
[Ingénieur système à l'INRIA]

Kerberos est un protocole d'authentification réseau, qui présente la particularité d'allier à la fois sécurité et confort d'utilisation, puisqu'il s'agit d'un système d'authentification unique (SSO, Single Sign On). À l'heure où fleurissent les articles vantant les mérites des systèmes de type CAS, assurant un service similaire, mais limité aux seules applications web, il paraît intéressant de présenter cette technologie quelque peu méconnue.

L'article commence par présenter le protocole et ses concepts. Ensuite, il explique le déploiement de l'infrastructure, puis l'intégration des applications à celle-ci. Enfin, il aborde certaines subtilités de configuration et quelques exemples d'utilisation plus avancée.

1 Présentation

1.1 Historique

Kerberos a vu le jour dans les années 1980, au sein du projet Athena, un projet de recherche conjoint entre le MIT, DEC et IBM, d'où sortit également le système X Windows. Il s'agit d'un système conçu pour assurer une authentification fiable et réciproque entre différents nœuds d'un réseau non sécurisé, garantissant la confidentialité des échanges et l'impossibilité de les rejouer. Le tout à une époque où la norme était plutôt à l'utilisation des mots de passe en clair via Telnet, pour situer le contexte...

Vers la fin des années 80, la première version publique est publiée, sous le nom de Kerberos 4. En plus d'être utilisée

pour le projet Athena, cette version est également adoptée comme protocole d'authentification par APS, le système de fichiers distribué conçu au sein de l'université de Carnegie-Mellon.

La version 5, destinée à corriger certaines limites de la version précédente, est la première à faire l'objet d'une spécification, sous la forme de la RFC 1510 en 1993. Cette première spécification sera ensuite remplacée par la RFC 4120 (*The Kerberos Network Authentication Service*) en 2005 et complétée par d'autres, telles que les RFC 3961 (*Encryption and Checksum Specifications*), 3962 (*AES Encryption for Kerberos 5*) et 4121 (CSS API).

1.2 Concepts

Une installation de Kerberos constitue un royaume, c'est-à-dire une entité distincte de toute autre installation. Ce royaume va constituer un espace de nommage unique pour ses différents participants. Par convention et afin d'assurer cette unicité, c'est le nom de domaine DNS mis en majuscules qui détermine le nom de ce royaume. Au domaine **zarb.home** correspond donc le royaume **ZARB.HOME**.

À l'intérieur de ce royaume, chaque participant est identifié par un principal, de la forme **nom@ROYAUME**. Certains acteurs peuvent éventuellement posséder plusieurs principaux, différenciés alors par leur instance, ce qui donne des principaux de la forme **nom/instance@ROYAUME**. Typiquement, les principaux correspondant à des services hébergés sur une machine sont de la forme **service/machine@ROYAUME** et les administrateurs système possèdent des principaux **nom/admin@ROYAUME** dédiés à l'administration, en supplément de leurs principaux d'utilisateur **nom@ROYAUME**.

L'élément central de chaque royaume est le *Key Distribution Center*, ou KDC. Celui-ci est constitué d'une base de données de tous les principaux du royaume, chacun associé à un secret partagé avec l'entité correspondant à ce principal. Pour un utilisateur, ce secret est son mot de passe. Pour un service, ce secret est en général généré aléatoirement à la création du principal. Ce secret est stocké sous la forme d'une clé pour chaque algorithme de chiffrement supporté. D'autres informations relatives au principal peuvent

également être stockées, comme par exemple la durée de validité du principal, la date de dernier changement de mot de passe, etc.

Le KDC joue le rôle d'un tiers de confiance, en distribuant aux utilisateurs des tickets, prouvant leur identité vis-à-vis d'un service cible. Ces tickets sont en fait des jetons d'authentification, à durée de vie limitée afin d'éviter une attaque par rejet. Ils sont totalement opaques pour l'utilisateur et déchiffrables uniquement par l'interlocuteur qu'il a choisi, ce qui assure une authentification réciproque.

Il faut bien comprendre que ce mécanisme n'assure que l'authentification des divers participants. La seule information véhiculée par un ticket est que son possesseur est bien celui qu'il prétend être, mais en aucun cas qui il est. Ceci peut être suffisant pour des scénarios simples (tout utilisateur authentifié est valide), mais ne permet absolument pas de gérer des scénarios plus complexes, basés sur des notions d'autorisation. Et encore moins de gérer l'ensemble des informations liées à la notion de compte informatique (GID, UID, etc.). Pour cette raison, un royaume Kerberos n'est pas une alternative à une base de comptes centralisés, comme un annuaire LDAP par exemple, mais un complément possible, apportant un mécanisme supplémentaire d'authentification, plus sécurisé et plus transparent pour l'utilisateur.

1.3 Fonctionnement

1.3.1 Protocole

De façon plus détaillée, le protocole fait intervenir 4 acteurs :

- le serveur d'authentification (*Authentication Server*),
- le serveur de ticket (*Ticket-Granting Server*),
- le client,
- le service.

Dans la pratique, les deux premiers rôles sont tenus par le KDC, même s'il s'agit de rôles distincts.

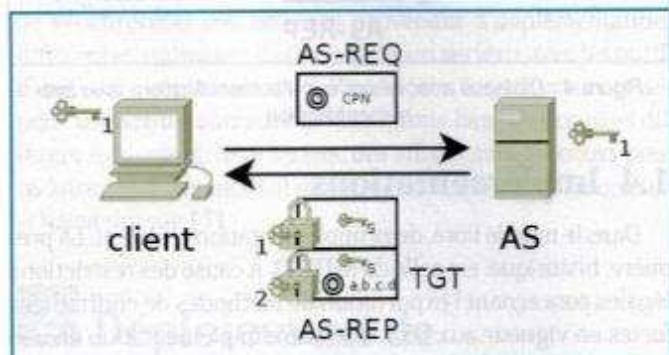


Figure 1 : Dialogue avec le serveur d'authentification

La figure 1 montre le dialogue entre le client et le serveur d'authentification, lorsque l'utilisateur s'authentifie. Ce dialogue est unique pour la session de travail, c'est le principe même du SSO.

Le client envoie une requête (AS-REQ), comportant son principal (*Client Principal Name*, CPN), ainsi qu'une estampille temporelle (*timestamp*), en clair. Cette dernière sert uniquement à s'assurer de la synchronisation des horloges entre le client et le serveur, puisque cette synchronisation est critique pour le reste des opérations. Si la différence est trop importante, le serveur refuse l'authentification immédiatement. De même, si le serveur ne trouve aucun principal dans sa base correspondant à la requête.

Le serveur répond alors par un message (AS-REP) composé de deux parties distinctes :

- une partie chiffrée avec la clé de l'utilisateur, comportant notamment une clé de session maître (S),
- une partie chiffrée avec la clé du serveur de ticket, comportant notamment cette même clé de session, une estampille temporelle et l'adresse IP de l'utilisateur. Cette partie constitue le ticket maître (*Ticket-Granting Ticket*, ou TGT).

Le client est alors en possession d'une clé de session, qu'il ne peut déchiffrer que s'il est le destinataire légitime de ce message, puisqu'il est chiffré avec la clé de l'utilisateur. Celui-ci doit alors saisir son mot de passe pour pouvoir le déchiffrer. Le client possède également un autre ensemble d'informations, le TGT, mais qui lui est totalement opaque, puisqu'il est chiffré avec une clé qu'il ne possède pas.

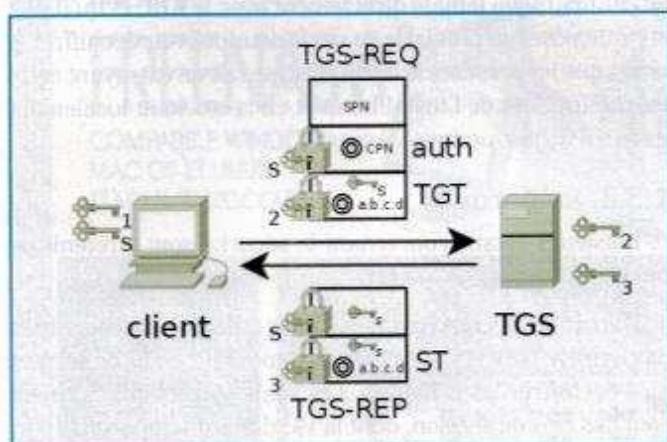


Figure 2 : Dialogue avec le serveur de ticket

La figure 2 illustre maintenant le dialogue entre le client et le serveur d'authentification, lorsque l'utilisateur désire utiliser un service kerberisé. Contrairement au précédent, ce dialogue se reproduit plusieurs fois au cours de la session de travail, mais de façon totalement transparente pour l'utilisateur.

Le client envoie une requête (TGS-REQ) comportant trois parties. D'abord, l'identité du service souhaité (*Service Principal Name*, SPN), en clair. Ensuite, un authenticateur, constitué de son propre principal et d'une estampille temporelle, chiffrée avec la clé de session récupérée à l'étape précédente, afin de prouver qu'il en a connaissance. Et enfin, le TGT également récupéré à l'étape précédente, tel quel.

Le serveur répond par un message (TGS-REP) encore une fois composé de deux parties :

- une partie chiffrée avec la clé de session maître, comportant notamment une clé de session de service (s).
- une partie chiffrée avec la clé du service, comportant notamment cette même clé de session de service, une estampille et l'adresse IP du client. Cette partie constitue le ticket de service (*Service Ticket*, ST).

Le client est donc en possession d'une information qu'il peut déchiffrer (la clé de session) et d'une autre qui lui est totalement opaque, le ticket de service.

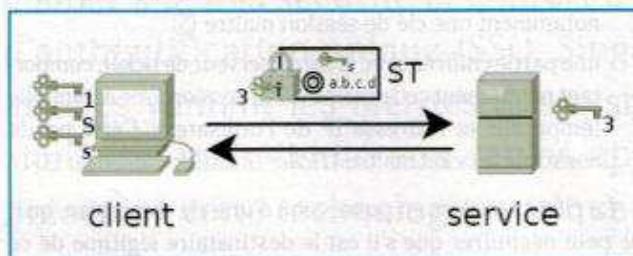


Figure 3 : Dialogue avec le service

Enfin, la figure 3 présente le dialogue entre le client et le service visé. Cette partie n'étant pas normalisée, elle dépend fortement du service. Néanmoins, il est notable que le service ne communique jamais directement avec le KDC et qu'il doit donc disposer au préalable de sa clé pour pouvoir déchiffrer le ticket que lui présente le client. Ceci se fait en extrayant cette clé du KDC lors de l'installation et en la stockant localement dans un fichier, nommé « keytab ».

1.3.2 Remarques

Plusieurs détails concernant la sécurité sont à retenir de ces explications.

D'abord, les secrets partagés par les différents protagonistes ne circulent jamais sur le réseau, même chiffrés. Ils ne servent qu'à déchiffrer les échanges. Les seuls secrets qui circulent sont des clés de session, dont la validité est temporaire.

Ensuite, de nombreux éléments destinés à empêcher des attaques par rejet interviennent. Par exemple, le ticket maître ou les tickets de services contiennent l'adresse IP du client, ce qui les rend inutilisables (en dehors d'une usurpation d'adresse) depuis une autre machine. Mais surtout, les échanges incluent des estampilles temporales, de façon à leur donner une durée de vie limitée. Cet horodatage systématique des tickets suppose une cohérence entre les horloges des différents intervenants, d'où le conseil fréquent de mettre en place un mécanisme de synchronisation, comme NTP, comme préalable à tout déploiement Kerberos. Ces estampilles ne sont pas limitées aux seuls tickets, mais également utilisées dans l'authentificateur de chaque requête TGS-REQ, ce qui limite la fenêtre de vulnérabilité en cas de capture du trafic réseau

à une durée relativement courte, qu'il ne faut pas confondre avec la durée de validité des tickets eux-mêmes. Ces caractéristiques permettent ainsi d'utiliser Kerberos comme un moyen de limiter le principal risque lié à l'utilisation d'un protocole non chiffré, comme FTP ou Telnet par exemple, mais sans le faire disparaître totalement et sans résoudre le problème de la confidentialité non plus. Un tunnel SSH reste une meilleure solution, bien entendu.

Enfin, la phase d'authentification décrite précédemment est vulnérable. En effet, le serveur d'authentification répond à toute requête par un message chiffré avec la clé de l'utilisateur mentionné dans la requête, ce qui permet facilement de mettre en place une attaque par force brute, hors-ligne de surcroît. C'est pourquoi ce fonctionnement a été enrichi dans la version 5 du protocole par un mécanisme optionnel de pré-authentification (PRE-AUTH). Lorsque celui-ci est utilisé, la requête initiale AS-REQ est refusée, avec un code de retour spécifique, pour forcer une nouvelle requête, comportant cette fois-ci une preuve de l'identité de son demandeur. Cette preuve peut prendre plusieurs formes, mais celle la plus couramment utilisée aujourd'hui consiste à ajouter une estampille temporelle, chiffrée avec la clé du demandeur. La figure 4 présente le dialogue d'authentification dans ce cas. D'autres mécanismes existent également et notamment PKINIT, basé sur l'emploi de certificats, plus robuste mais également plus complexe à mettre en place.

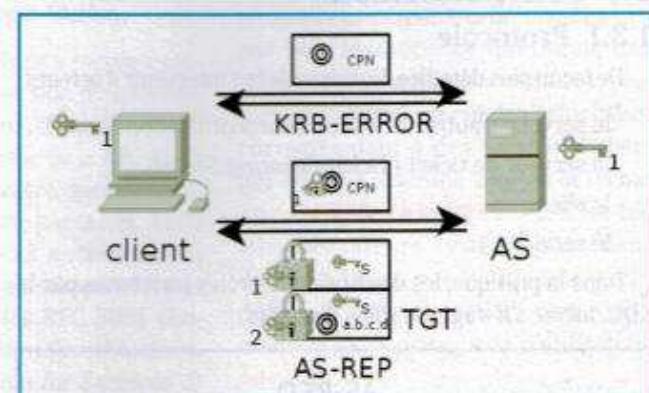


Figure 4 : Dialogue avec le serveur d'authentification, avec pré-authentification

1.4 Implémentations

Dans le monde libre, deux implémentations existent. La première, historique, est celle du MIT [1]. À cause des restrictions légales concernant l'exportation de méthodes de chiffrement fortes en vigueur aux USA, une autre implémentation a vu le jour en Suède, KTH-KRB (du nom de l'Institut Royal de Technologie, KTH en suédois), qui est devenue Heimdal [2] par la suite. Celle-ci est réputée avoir un code plus lisible et plus sain, comme c'est souvent le cas lorsqu'il s'agit de comparer deux logiciels similaires dont l'ancienneté diffère. Il suffit de penser à Sendmail et Postfix, ou Washington IMAP et Dovecot,

par exemple. C'est notamment pour cette raison que les développeurs de Samba ont choisi Heimdal pour implémenter les fonctionnalités Kerberos nécessaires au support d'Active Directory dans Samba 4. Au niveau des fonctionnalités, Heimdal a longtemps eu une certaine avance, comme par exemple la synchronisation différentielle, ou le support d'un annuaire LDAP comme base de données. Mais les versions de MIT Kerberos se succèdent à un rythme soutenu et les deux implémentations semblent maintenant équivalentes.

Dans le monde Apple, Mac OS X s'appuie sur l'implémentation du MIT pour offrir un support de Kerberos natif dans de nombreuses applications et avec une intégration relativement forte, puisqu'il existe une interface graphique pour la gestion des tickets. Néanmoins, ces applications se comportent parfois différemment de ce que l'on observe sous Linux, comme il sera présenté plus loin.

Dans le monde Microsoft, Kerberos est l'une des bases d'Active Directory, il est donc disponible nativement dans n'importe quelle version de Windows depuis Windows 2000. Par contre, il est si intégré dans le système qu'il n'est utilisable que si la machine fait partie d'un domaine Active Directory et uniquement avec ce domaine ou tout autre domaine lié par une relation de confiance. En particulier, il n'y a pas de moyen d'acquérir un TGT manuellement, c'est fait automatiquement lors de l'ouverture d'une session par un utilisateur. Il existe également un port Windows de MIT Kerberos, utilisable avec des applications comme Firefox, par exemple, mais pas avec les applications natives du système.

Ces différentes implémentations sont largement interopérables entre elles, du moins pour tout ce qui fait partie de la spécification. Les clients, par exemple, sont parfaitement utilisables avec n'importe quel serveur. L'implémentation de Microsoft, malgré quelques craintes [3] au moment de son introduction initiale, est également compatible et il existe même une documentation officielle [4] à ce sujet. Par contre, les protocoles d'administration diffèrent et il est impossible de synchroniser des serveurs provenant d'implémentations différentes, ou encore d'administrer un serveur avec les outils d'une autre implémentation. Pire encore, les schémas utilisés pour utiliser un annuaire LDAP comme base de données diffèrent également. Néanmoins, des efforts sont en cours pour réduire ces différences et notamment la standardisation d'un schéma unique [5].

2 Déploiement de l'infrastructure

Un royaume Kerberos s'articule autour d'un serveur de distribution de clés (KDC), mais il ne s'agit pas du seul élément nécessaire pour constituer une infrastructure complète. Cette partie présente l'ensemble de ces éléments.

VOUS RECHERCHEZ UNE SOLUTION DE STOCKAGE SIMPLE À METTRE EN PLACE ?

LP 68

Actuellement en kiosque !

FREENAS 8

COMPATIBLE WINDOWS, MAC OS ET LINUX ET MULTI-PROTOCOLE

LINUX PRATIQUE N°68

N° 68 NOVEMBRE/DÉCEMBRE 2011 CD-ROM OFFERT : TAILS 0.8

LINUX
DÉCOUVRIR, COMPRENDRE ET UTILISER LINUX
PRATIQUE

DISTRIBUTION
CRÉEZ VOTRE SYSTÈME « À LA CARTE » AVEC ARCH LINUX

VIDÉO
DE BEAUX DIAPORAMAS
VIDÉO AVEC IMAGEMOVIE ET OPENSHOT VIDEO EDITOR

SCIENCES
L'ÉDITION LATEX FACILE GRÂCE À LYX 2

SYSTÈME
UIDEV : RÉDIGEZ VOS PROPRES RÈGLES POUR GERER VOS PÉRIPHÉRIQUES

SYSTÈME
VOUS RECHERCHEZ UNE SOLUTION DE STOCKAGE SIMPLE À METTRE EN PLACE ?

FREENAS 8
COMPATIBLE WINDOWS, MAC OS ET LINUX ET MULTI-PROTOCOLE

WEB
DÉVELOPPEMENT
HTML5 : NOUVEAUTÉS ET IMPLICATIONS

GESTION DE CONTENU
DÉPLOYEZ VOS FORUMS A L'AIDE DE BBPRESS

TRAVAIL COLLABORATIF
LE PARTAGE DE DOCUMENTS AVEC FILEZ ET DMANGER

RÉSEAU
UNE GESTION SIMPLE ET SÉCURISÉE DE VOTRE RÉSEAU GRÂCE À INTANGLE

FRANCK MÉTRAL - ERIC DOM - ANDRÉ TOMASSENKO - BRUNO BAPTISTE - THOMAS BONHOMME - JEAN-BAPTISTE BOURGEOIS - CHRISTIAN CHAMON - CLAUDIO FRANCISCO - MARC-JEAN CHAMON

DISPONIBLE
CHEZ VOTRE MARCHAND
DE JOURNAUX JUSQU'AU
23 DÉCEMBRE 2011 ET SUR :

www.ed-diamond.com

L'environnement utilisé pour les exemples qui suivent se compose de 3 machines, tournant sous Mandriva 2010.0 :

- **tcheka.zarb.home** héberge les serveurs Kerberos.
- **loubianka.zarb.home** héberge les services kerberisés.
- **beria.zarb.home** est la machine de l'utilisateur.

L'implémentation utilisée est Heimdal, de façon tout à fait arbitraire. L'utilisation de MIT Kerberos serait quasiment identique, à l'exception des commandes d'administration. Par ailleurs, toutes les machines sont synchronisées sur un serveur de temps unique et la résolution des noms et des adresses IP est correcte sur chacune d'elles. Enfin, le nom du royaume à mettre en place est **ZARB.HOME**.

2.1 Configuration

La configuration de Kerberos se fait dans un fichier unique, **/etc/krb5.conf**, aussi bien pour les clients que pour les serveurs. Ce fichier est organisé en différentes sections, contenant chacune différentes directives. Un fichier de configuration minimal définit les trois éléments suivants :

- le royaume à utiliser par défaut pour l'ensemble des applications, via la directive **default_realm** de la section **libdefaults**,
- les serveurs à utiliser pour le ou les royaume(s) à utiliser, dans la section **realms**,
- une correspondance entre nom de domaine et nom de royaume, de façon à déduire dans quel royaume se trouve une ressource donnée, dans la section **domain_realm**.

Des sections spécifiques propres aux serveurs ou aux applications peuvent ensuite s'y ajouter. Pour l'instant, le fichier minimal suivant est suffisant :

```
[libdefaults]
default_realm = ZARB.HOME

[realms]
ZARB.HOME = {
    kdc      = tcheka.zarb.home
    admin_server = tcheka.zarb.home
    kpasswd_server = tcheka.zarb.home
}

[domain_realm]
.zarb.home = ZARB.HOME
```

Ce fichier est à mettre en place sur chacune des machines.

2.2 Mise en place du KDC

On commence par installer le KDC proprement dit, sur **tcheka**. Comme Heimdal est inclus dans la distribution, la solution la plus simple consiste à installer les paquetages nécessaires. Sur cette machine, il nous faut à la fois les programmes clients (**kinit**, **kadmin**, etc.), contenus dans le

paquetage **heimdal-workstation** et les programmes serveurs (**kdc**, **kadmind**, **kpasswd**), contenus dans le paquetage **heimdal-server**.

Dans le fichier de configuration, on ajoute une section **logging** concernant la gestion des journaux d'activité (logs) des serveurs :

```
[logging]
kdc      = FILE:/var/log/kerberos/kdc.log
kadmind  = FILE:/var/log/kerberos/kadmind.log
kpasswd  = FILE:/var/log/kerberos/kpasswd.log
default  = FILE:/var/log/kerberos/default.log
```

Il faut ensuite mettre en place une clé de stockage. Celle-ci ne sert qu'à chiffrer la base de données sur le disque et n'intervient plus par la suite, si ce n'est lors du déploiement de KDC secondaires. Elle peut être générée aléatoirement pour une meilleure robustesse :

```
[root@tcheka ~]# kstash --random-key
kstash: writing key to '/var/lib/heimdal/m-key'
```

L'étape suivante consiste à se connecter sur le serveur d'administration. Pour l'instant, seule une connexion locale est possible, puisqu'il n'existe encore aucun compte permettant une authentification. Dans ce mode de fonctionnement, il n'y a pas besoin d'un processus serveur en écoute sur un port réseau et ce sont les droits Unix qui déterminent l'accès à la base de données sur le disque. Cette connexion donne accès à une console d'administration interactive, permettant d'exécuter un certain nombre de commandes. Et notamment, la commande **init**, pour initialiser le royaume. Celle-ci pose quelques questions portant sur la durée de vie maximale des tickets, illimitée par défaut :

```
[root@tcheka ~]# kadmin -l
kadmin> init ZARB.HOME
Realm max ticket life [unlimited]:
Realm max renewable ticket life [unlimited]:
```

Cette initialisation crée automatiquement un certain nombre de principaux administratifs, dont le fameux principal du serveur de tickets, **krbtgt/ZARB.HOME@ZARB.HOME**. La commande **get** permet d'examiner ces différents principaux :

```
kadmin> get krbtgt*
Principal: krbtgt/ZARB.HOME@ZARB.HOME
Principal expires: never
Password expires: never
last password change: 2010-04-04 14:53:30 UTC
Max ticket life: unlimited
Max renewable life: unlimited
Kvno: 1
Mkvno: unknown
Last successful login: never
Last failed login: never
Failed login count: 0
Last modified: 2010-04-04 14:53:30 UTC
Modifier: kadmin/admin@ZARB.HOME
```

```
Attributes:
  Keytypes: aes256-cts-hmac-sha1-96(pw-salt), des3-cbc-sha1(pw-salt),
             arcfour-hmac-md5(pw-salt)
PK-INIT ACL:
  Aliases:
```

Parmi les différentes informations énumérées, on retrouve des informations concernant la durée de vie du principal, de son mot de passe et des tickets. On trouve également des traces de l'utilisation du principal pour s'authentifier, ainsi que de sa création puis de ses éventuelles modifications par la suite. Viennent ensuite des propriétés spécifiques au principal, qui sont généralement des restrictions et enfin la liste des algorithmes de chiffrement supportés. Par défaut, sur des versions récentes, DES n'en fait plus partie...

Il est maintenant possible de créer le premier principal utilisateur, avec la commande **add**. Encore une fois, on peut se contenter de la valeur par défaut pour les différentes questions posées :

```
kadmin> add guillaume
Max ticket life [unlimited]:
Max renewable life [unlimited]:
Principal expiration time [never]:
Password expiration time [never]:
Attributes []:
guillaume@ZARB. HOME's Password:
Verifying - guillaume@ZARB. HOME's Password
```

On peut alors vérifier le principal.

```
kadmin> get guillaume
  Principal: guillaume@ZARB. HOME
  Principal expires: never
  Password expires: never
  Last password change: 2010-04-04 15:05:35 UTC
    Max ticket life: 1 day
    Max renewable life: 1 week
      Kvno: 1
      Mvno: unknown
  Last successful login: never
  Last failed login: never
  Failed login count: 0
  Last modified: 2010-04-04 15:05:35 UTC
    Modifier: kadmin/admin@ZARB. HOME
  Attributes:
    Keytypes: aes256-cts-hmac-sha1-96(pw-salt), des3-cbc-sha1(pw-salt),
               arcfour-hmac-md5(pw-salt)
  PK-INIT ACL:
    Aliases:
```

Si tout a l'air en ordre, le service est en mesure de fonctionner. Il ne reste qu'à lancer le KDC pour qu'il soit en mesure de recevoir des requêtes par le réseau. Dans le cas du paquetage Mandriva, un script de démarrage est fourni à cet effet :

```
[root@tcheka heimdal]# service heimdal start
Lancement de Heimdal Kerberos 5 Key Distribution Center: [ OK ]
```

Sur **beria**, on peut alors tester la récupération d'un ticket, depuis le compte d'un utilisateur ordinaire :

```
[guillaume@beria ~]$ kinit
guillaume@ZARB. HOME's Password:
```

Par défaut, **kinit** utilise le nom d'utilisateur local, ainsi que le royaume par défaut défini dans la configuration pour déterminer le principal à utiliser. Si celui-ci n'est pas correct, il est possible de préciser ce dernier sur la ligne de commandes :

```
[guillaume@beria ~]$ kinit guillaume@ZARB. HOME
guillaume@ZARB. HOME's Password:
```

Si tout se passe correctement, ce qui se manifeste par l'absence de message d'erreur, l'utilisateur est maintenant en possession de son TGT, comme le montre la commande **klist** :

```
[guillaume@beria ~]$ klist
Credentials cache: FILE:/tmp/krb5cc_500
  Principal: guillaume@ZARB. HOME
Issued           Expires          Principal
Apr  4 18:07:30 Apr  5 04:07:30 krbtgt/ZARB. HOME@ZARB. HOME
```

2.3 Mise en place du serveur d'administration

Une fois l'authentification fonctionnelle, il est possible de mettre en place des ACL permettant l'accès au serveur d'administration à distance. En effet, ceci permet de manipuler des principaux et d'extraire des clés directement sur les machines hébergeant les services que nous allons kerberiser. De plus, ceci permet de relier les modifications à un compte utilisateur, ce qui offre de meilleures garanties de traçabilité, alors que l'accès local ne le permet pas.

La mise en place d'ACL se fait via un fichier, localisé dans le même répertoire que la base de données, soit **/var/lib/heimdal/kadmind.acl** dans le cas d'une installation respectant le FHS. Chaque ligne de ce fichier correspond à une autorisation de la forme « sujet-action-objet ». Par exemple, la ligne suivante donne tous les droits à notre principal actuel sur l'ensemble du royaume :

```
guillaume@ZARB. HOME a11 *@ZARB. HOME
```

L'utilisation du caractère ***** comme joker est possible pour la 3ème colonne, celle des principaux sur laquelle l'action s'applique. Néanmoins, il ne correspond qu'aux caractères alphanumériques et la règle précédente ne couvre que des principaux dépourvus d'instance. Pour correspondre à tous les principaux possibles, il faut donc deux règles :

```
guillaume@ZARB. HOME a11 *@ZARB. HOME
guillaume@ZARB. HOME a11 */@ZARB. HOME
```

Une fois ces ACL en place, il faut lancer le serveur d'administration (**kadmind**) en supplément du KDC. Dans le cas du paquetage Mandriva, ceci se fait en configurant le script

de démarrage, par la directive **START_KADMIND=yes** dans le fichier **/etc/sysconfig/heimdal**. Il suffit ensuite de relancer le service :

```
[root@tcheka ~]# service heimdal restart
Arrêt de Heimdal Kerberos 5 Key Distribution Center: [ OK ]
Arrêt de Heimdal administration server: [ ÉCHEC ]
Lancement de Heimdal Kerberos 5 Key Distribution Center: [ OK ]
Lancement de Heimdal administration server: [ OK ]
```

Il est maintenant possible d'accéder à la console d'administration à distance et d'utiliser les ACL Kerberos plutôt que les droits Unix sur le serveur pour gérer les autorisations.

```
[guillaume@beria ~]$ /usr/sbin/kadmin
kadmin> get guillaume
guillaume/admin@ZARB.HOME's Password:
kadmin: get guillaume: Client not found in Kerberos database
```

Par défaut, **kadmin** utilise le nom d'utilisateur local et ajoute une instance admin, ce qui correspond à une pratique classique de dissociation des priviléges. Ici, nous avons préféré donner les autorisations d'administration au principal de l'utilisateur, il faut donc forcer l'utilisation de celui-ci via l'option **-p** :

```
[guillaume@beria ~]$ /usr/sbin/kadmin -p guillaume
kadmin> get guillaume
guillaume@ZARB.HOME's Password:
Principal: guillaume@ZARB.HOME
Principal expires: never
Password expires: never
Last password change: 2010-04-04 16:06:37 UTC
Max ticket life: 1 day
Max renewable life: 1 week
Kvno: 1
Nkno: unknown
Last successful login: never
Last failed login: never
Failed login count: 0
Last modified: 2010-04-04 16:06:37 UTC
Modifier: kadmin/admin@ZARB.HOME
Attributes:
Keytypes: aes256-cts-hmac-sha1-96(pw-salt), des3-cbc-sha1(pw-salt),
arcfour-hmac-md5(pw-salt)
PKINIT ACL:
Aliases:
```

Attention, **kadmin** étant situé dans **/usr/sbin**, il est nécessaire de préciser le chemin complet lorsqu'il est utilisé depuis un compte d'utilisateur non privilégié.

2.4 Mise en place du serveur de changement de mot de passe

Le dernier serveur à activer, **kpasswd**, sert à changer les mots de passe. Comme pour le serveur d'administration, ceci se fait en positionnant la variable **START_KPASSWD=yes** dans le fichier **/etc/sysconfig/heimdal**, puis en redémarrant le service :

```
[root@tcheka ~]# service heimdal restart
Arrêt de Heimdal Kerberos 5 Key Distribution Center: [ OK ]
Arrêt de Heimdal administration server: [ OK ]
Arrêt de Heimdal Kerberos 5 password changing server: [ ÉCHEC ]
Lancement de Heimdal Kerberos 5 Key Distribution Center: [ OK ]
Lancement de Heimdal administration server: [ OK ]
Lancement de Heimdal Kerberos 5 password changing server: [ OK ]
```

L'utilisateur peut alors changer son mot de passe via la commande **kpasswd** :

```
[guillaume@beria ~]$ kpasswd
guillaume@ZARB.HOME's Password:
New password:
Verifying - New password:
```

L'inspection du principal via le serveur d'administration permet de constater que ce changement a bien été pris en compte :

```
kadmin> get guillaume
Principal: guillaume@ZARB.HOME
Principal expires: never
Password expires: never
Last password change: 2010-04-05 19:38:12 UTC
Max ticket life: 1 day
Max renewable life: 1 week
Kvno: 2
Nkno: unknown
Last successful login: never
Last failed login: never
Failed login count: 0
Last modified: 2010-04-05 19:38:12 UTC
Modifier: guillaume@ZARB.HOME
Attributes:
Keytypes: aes256-cts-hmac-sha1-96(pw-salt), des3-cbc-sha1(pw-salt),
arcfour-hmac-md5(pw-salt)
PKINIT ACL:
Aliases:
```

Le changement du mot de passe est apparent via la date du dernier changement et la version de la clé (**Kvno**). La modification du principal, d'autre part, est également visible, avec la date du changement et l'auteur de ce changement, qui est cette fois-ci l'utilisateur lui-même.

L'infrastructure étant maintenant en place, il devient possible de l'utiliser concrètement, ce que nous allons voir dans la partie suivante. ■

Références

- [1] <http://web.mit.edu/Kerberos>
- [2] <http://www.h5l.org>
- [3] <http://www.h5l.org/manual/HEAD/info/heimdal-Authorisation-data.html#Authorisation-data>
- [4] <http://technet.microsoft.com/fr-fr/library/bb742433%28en-us%29.aspx>
- [5] <http://tools.ietf.org/html/draft-chu-ldap-kdc-schema-00>

KERBEROS, LE SSO UNIVERSEL

2- INTÉGRATION DES APPLICATIONS

par Guillaume Rousse

L'intégration des applications à une infrastructure Kerberos, ou kerberisation, peut se faire de deux manières différentes. La première, l'authentification Kerberos à proprement parler, consiste à utiliser le mécanisme décrit précédemment, à base de tickets, entre le client et le serveur (figure 1). Il faut bien évidemment un support explicite pour ce mécanisme au niveau du client et du serveur, mais également dans le protocole utilisé.

Ilsagit donc d'une contrainte forte, nécessitant souvent de pouvoir revenir à un mécanisme plus classique comme solution de repli pour être utilisable dans un contexte hétérogène.

La seconde, la validation des mots de passe via Kerberos, ne fait que remplacer la façon dont le serveur vérifie le mot de passe que lui transmet le client de manière tout à fait classique, en se servant de celui-ci pour obtenir un ticket depuis le KDC (figure 2). Cette solution ne nécessite donc aucun support particulier au niveau du client et du protocole, mais elle n'apporte aucun des avantages liés à Kerberos. En particulier, ni le confort du SSO, ni la sécurité associée. Il est donc important de bien faire la distinction entre les deux.

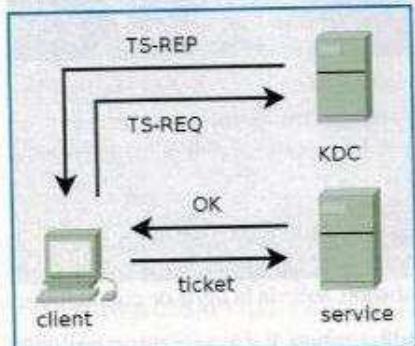


Figure 1 : Authentification Kerberos

1 Principes

La procédure elle-même est relativement homogène d'une application à l'autre. Il faut d'abord créer un principal correspondant au service. Il faut ensuite mettre en place les clés correspondant à ce principal sur la machine hébergeant le service dans un fichier keytab. Ces deux étapes sont présentées successivement.

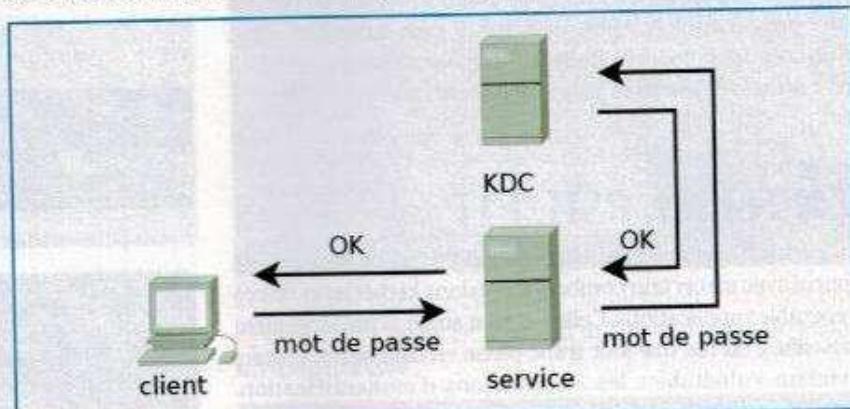


Figure 2 : Validation de mot de passe

1.1 Création du principal

La création d'un principal pour un service n'offre aucune difficulté particulière. Plutôt que choisir soi-même un mot de passe, on préférera en générer un aléatoirement via l'option `-r` de la commande `add`, puisque sa connaissance est inutile. L'option `--use-defaults` permet également d'éviter les questions inutiles.

Le nom du principal à utiliser est toujours constitué de deux parties :

- un nom de service, déterminé en général par l'application.
- une instance correspondant à la machine qui l'héberge.

Typiquement, un principal pour un service web kerberisé sera de la forme **HTTP/machine@DOMAINE.TLD**. Attention cependant aux subtilités liées aux différentes façons de nommer une machine, présentées dans la partie suivante, en page 56.

1.2 Mise en place du fichier keytab

Le fichier keytab, par défaut **/etc/krb5.keytab**, est un fichier particulièrement sensible. En effet, la connaissance de son contenu permet d'usurper l'identité du service, de la même façon que celle de la clé privée d'un certificat x509. Il convient donc de le protéger.

Au niveau de la mise en place, le plus simple est d'utiliser **kadmin** directement depuis la machine cible et la commande **ext_keytab**. Ceci permet de transférer les clés via un canal chiffré et de fusionner automatiquement celles-ci à d'autres qui seraient déjà présentes dans le même fichier keytab cible. Mais il est également possible d'exporter ce fichier sur une autre machine, puis de le transférer via un canal sûr (connexion chiffrée ou en-dehors du réseau) et de le mettre en place directement, ou alors de fusionner son contenu avec un autre via le programme **ktutil**.

Une fois en place, il convient de le protéger, en restreignant ses droits en lecture au minimum. Typiquement, root uniquement, mais il convient de s'assurer que le service auquel il est destiné puisse également y avoir accès, ce qui nécessite parfois d'élargir ces permissions si le service tourne sous une identité non privilégiée. Enfin, si plusieurs services kerberisés sont présents sur la même machine, il peut être intéressant d'utiliser des fichiers keytab séparés pour réduire les risques en cas de compromission de l'un d'entre eux.

2 Telnet, RSH, FTP

Heimdal, comme l'implémentation du MIT d'ailleurs, est fourni avec un certain nombre de versions kerberisées de ces vénérables applications. Celles-ci sont aujourd'hui largement obsolètes, du fait que leur trafic passe en clair sur le réseau, rendant vulnérables les informations d'authentification. L'utilisation de Kerberos, en limitant considérablement les possibilités de rejet, vient compenser cette faiblesse et redonner, dans une certaine limite, un minimum de crédibilité à ces outils historiques.

Néanmoins, il existe aujourd'hui des alternatives à la fois plus sécurisées et plus polyvalentes. De plus, leur valeur pédagogique à titre d'exemple simple à mettre en œuvre est également compromise par la disparition du support de DES par défaut dans les versions récentes de Heimdal ou de MIT Kerberos, qui implique une configuration particulière pour les faire fonctionner. J'ai moi-même renoncé à les utiliser à cet effet après m'être battu pendant une demi-heure avec **telnetd**.

sans possibilité de mettre en place des traces décentes me permettant d'isoler le problème. Bref, autant passer directement à des choses réellement utiles, comme SSH par exemple.

Il faut noter cependant que les clients kerberisés peuvent éventuellement être utiles dans certaines circonstances et notamment l'accès à des équipements limités dans les services qu'ils offrent. Par exemple, les équipements Cisco permettent d'utiliser Kerberos, mais uniquement pour **telnet**, **rlogin**, **rsh** et **rcp**...

3 OpenSSH

OpenSSH est capable d'utiliser l'authentification Kerberos. Il est également capable, en cas d'authentification classique par mot de passe, de récupérer un ticket Kerberos pour l'utilisateur. Ces deux comportements sont présentés successivement.

3.1 Authentification Kerberos

OpenSSH supporte l'authentification Kerberos via GSSAPI. Il s'agit en fait d'une API standardisée fournissant une couche d'abstraction au-dessus de différents mécanismes sous-jacents, dont Kerberos, d'une façon similaire à SASL. Comme l'API de Kerberos n'est pas standardisée, ceci permet d'être compatible avec différentes implémentations d'une manière simple.

Pour activer ce support, il faut créer un principal de type **host/machine@ROYAUME**, puis l'extraire dans un fichier keytab sur le serveur OpenSSH, donc **loubianka** pour nous :

```
[root@loubianka ~]# kadmin -p guillaume
kadmin> add --random-key --use-defaults host/loubianka.zarb.home
guillaume@ZARB.HOME's Password:
kadmin> ext_keytab host/loubianka.zarb.home
```

On peut ensuite vérifier le contenu de ce fichier :

Vno	Type	Principal	Aliases
1	aes256-cts-hmac-sha1-96	host/loubianka.zarb.home@ZARB.HOME	
1	des3-cbc-sha	host/loubianka.zarb.home@ZARB.HOME	
1	arcfour-hmac-md5	host/loubianka.zarb.home@ZARB.HOME	

Enfin, il faut positionner la directive **GSSAPIAuthentication yes** dans le fichier de configuration du serveur SSH, **/etc/ssh/sshd_config**, et relancer le serveur pour qu'il prenne en compte ces modifications.

Côté client, il faut également autoriser ce mode d'authentification. Comme souvent avec OpenSSH, il y a moyen de le faire soit via la configuration, soit via la ligne de commandes.

Dans le cas de la configuration, il y a deux directives, correspondant chacune à un mécanisme distinct :

- **GSSAPIAuthentication** contrôle l'authentification,
- **GSSAPIDelegateCredentials** contrôle le transfert des jetons.

Ces directives sont à placer soit dans le fichier de configuration global **/etc/ssh/sshd_config**, soit dans celui de l'utilisateur **~/.ssh/config**.

Du côté de la ligne de commandes, par contre, il y a deux options, mais qui mélangent les deux mécanismes :

- **-K** active le transfert (donc l'authentification),
- **-k** désactive le transfert.

Pour plus de clarté, on utilisera ici la configuration. On commence par activer l'authentification pour toutes les machines du domaine **zarb.home**. Il est en effet inutile de tenter d'utiliser cette méthode sur des machines qui n'appartiennent pas au royaume Kerberos de l'utilisateur. De plus, cette méthode étant par défaut prioritaire sur les autres méthodes, comme l'authentification par clé publique, elle peut même entraîner des délais, en forçant le client à essayer d'obtenir un ticket de service inutile. Sur **beria**, on met donc en place un fichier de configuration spécifique à l'utilisateur :

```
host *.zarb.home
  GSSAPIAuthentication yes
  GSSAPIDelegateCredentials no
```

On peut alors s'authentifier auprès du KDC, puis vérifier l'état de sa liste de jetons, qui ne contient alors qu'un TGT :

```
[guillaume@beria ~]$ kinit
guillaume@ZARB.HOME's Password:
[guillaume@beria ~]$ klist
Credentials cache: FILE:/tmp/krb5cc_500
  Principal: guillaume@ZARB.HOME

  Issued          Expires        Principal
Apr 15 22:49:47  Apr 16 08:49:48  krbtgt/ZARB.HOME@ZARB.HOME
```

Il n'y a plus qu'à tenter une connexion vers la machine cible, **toubianka**, avec l'option **-v** pour bien suivre la négociation avec le serveur :

```
[guillaume@toubianka ~]$ ssh -v toubianka.zarb.home
OpenSSH 5.4p1, OpenSSL 1.0.0 29 Mar 2010
[...]
debug1: Authentications that can continue: publickey,gssapi-with-mic,password,
Keyboard-interactive
debug1: Next authentication method: gssapi-with-mic
debug1: Authentication succeeded (gssapi-with-mic).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: Requesting X11 forwarding with authentication spoofing.
Last login: Thu Apr 15 22:53:54 2010 from beria.zarb.home
```

C'est bien GSSAPI qui a été utilisé, comme le montre la mention **gssapi-with-mic** dans la trace. Sur la machine d'origine, il est également possible de vérifier à nouveau

l'état de sa liste de jetons et de constater qu'un ticket pour le principal du service demandé s'est ajouté automatiquement à la liste, par la magie du SSO :

```
guillaume@beria ~]$ klist
Credentials cache: FILE:/tmp/krb5cc_500
  Principal: guillaume@ZARB.HOME

  Issued          Expires        Principal
Apr 15 22:52:54  Apr 16 08:52:55  krbtgt/ZARB.HOME@ZARB.HOME
Apr 15 22:53:04  Apr 16 08:52:55  host/toubianka.zarb.home@ZARB.HOME
```

Par contre, sur la machine finale, cette liste de jetons est vide :

```
[guillaume@toubianka ~]$ klist
klist: No ticket file: /tmp/krb5cc_500
```

C'est normal, la délégation n'est pas en place. Qu'à cela ne tienne, il suffit de l'activer, toujours dans le fichier de configuration de l'utilisateur :

```
host *.zarb.home
  GSSAPIAuthentication yes
  GSSAPIDelegateCredentials yes
```

Puis de retester l'opération :

```
[guillaume@beria ~]$ ssh -v toubianka.zarb.home
OpenSSH 5.4p1, OpenSSL 1.0.0 29 Mar 2010
[...]
debug1: Authentications that can continue: publickey,gssapi-with-mic,password,
Keyboard-interactive
debug1: Next authentication method: gssapi-with-mic
debug1: Delegating credentials
debug1: Delegating credentials
debug1: Authentication succeeded (gssapi-with-mic).
debug1: channel 0: new [client-session]
debug1: Requesting no-more-sessions@openssh.com
debug1: Entering interactive session.
debug1: Requesting X11 forwarding with authentication spoofing.
Last login: Thu Apr 15 23:08:44 2010 from beria.zarb.home
```

La trace porte maintenant la mention **Delegating credentials**. Mais ce n'est pas suffisant, la liste de jetons est toujours vide :

```
[guillaume@toubianka ~]$ klist
klist: No ticket file: /tmp/krb5cc_500
```

Pourquoi ? Les logs du serveur fournissent la réponse :

```
2010-04-15T23:15:45 TGS-REQ guillaume@ZARB.HOME From IPv4:192.168.0.7 for
krbtgt/ZARB.HOME@ZARB.HOME [forwarded]
2010-04-15T23:15:45 Request to forward non-forwardable ticket
2010-04-15T23:15:45 Failed building TGS-REP to IPv4:192.168.0.7
2010-04-15T23:15:45 sending 107 bytes to IPv4:192.168.0.7
```

En effet, il ne suffit pas de demander la transmission d'un ticket, il faut encore que celui-ci soit effectivement transmissible. C'est une option à demander spécifiquement au KDC lors de

l'authentification initiale. Il faut donc recommencer celle-ci, en utilisant l'option **-f** de **kinit** et utiliser ensuite **klist** en mode verbeux pour s'assurer du résultat :

```
[guillaume@beria ~]$ kinit -f
guillaume@ZARB.HOME's Password:
[guillaume@beria ~]$ klist -v
Credentials cache: FILE:/tmp/krb5cc_500
  Principal: guillaume@ZARB.HOME
  Cache version: 4

Server: krbtgt/ZARB.HOME@ZARB.HOME
Client: guillaume@ZARB.HOME
Ticket etype: aes256-cts-hmac-sha1-96, kvno 1
Ticket length: 318
Auth time: Apr 15 23:19:24 2010
End time: Apr 16 09:19:25 2010
Ticket flags: pre-authent, initial, forwardable
Addresses: addressless
```

Cette fois-ci, on peut recommencer la connexion SSH et tout fonctionne comme prévu ; on dispose d'un TGT sur la machine d'arrivée :

```
[guillaume@beria ~]$ ssh loubianka.zarb.home
Last login: Thu Apr 15 23:15:46 2010 from beria.zarb.home
[guillaume@loubianka ~]$ klist
Credentials cache: FILE:/tmp/krb5cc_500_1Ngqhb6895
  Principal: guillaume@ZARB.HOME

  Issued          Expires        Principal
Apr 15 23:21:10  Apr 16 09:19:25  krbtgt/ZARB.HOME@ZARB.HOME
```

Le transfert des tickets est à Kerberos ce qu'est le transfert d'agent à l'authentification par clé publique : un confort certain, en permettant de continuer à bénéficier des avantages du SSO même après s'être connecté à une autre machine. Et comme le transfert d'agent, c'est également un risque potentiel en matière de sécurité : si un utilisateur dispose des priviléges suffisants pour lire le fichier contenant la liste des tickets, il peut usurper l'identité de son possesseur. Ce n'est pas strictement lié au transfert, c'est également vrai sur la machine d'origine, mais celle-ci est généralement mieux maîtrisée, par rapport à une machine distante, dont la liste des comptes utilisateurs et de leurs priviléges n'est pas forcément connue.

3.2 Récupération de ticket Kerberos

OpenSSH, immédiatement après une authentification classique par mot de passe, est capable d'obtenir un ticket Kerberos pour l'utilisateur auprès du KDC, à condition bien sûr que le mot de passe soit identique pour les deux systèmes. Ceci permet à l'utilisateur de disposer immédiatement d'un TGT sur la machine cible et lui évite d'appeler **kinit** explicitement. C'est donc un peu le transfert de ticket du pauvre.

Ce comportement se configure par la directive **KerberosAuthentication** dans la configuration du serveur **sshd**, **/etc/ssh/sshd_config**.

4 Apache

Apache, grâce au module **mod_auth_kerb** [1], peut également bénéficier du support de Kerberos. Les applications qui laissent le serveur web gérer l'authentification, comme Nagios, ou celles qui permettent de le faire, comme Cacti, peuvent en bénéficier. Quant aux autres, ceci nécessite des modifications du code qui dépassent le cadre de cet article.

Ce module implémente la méthode Negotiate, introduite à l'origine par Microsoft, qui permet au navigateur et au serveur web de négocier un mécanisme d'authentification spécifique, par opposition aux méthodes classiques Digest et Basic qui imposent chacune un mécanisme unique. Dans le cas d'un environnement Active Directory, ceci permet à un serveur IIS d'utiliser NTLM si le navigateur n'est pas en mesure de présenter un ticket Kerberos valide. Il s'agit de l'authentification Kerberos à proprement parler.

Si le navigateur ne supporte pas Negotiate, ou si la négociation Kerberos échoue, le module repasse à la méthode Basic et vérifie le mot de passe de l'utilisateur grâce au KDC. Il s'agit dans ce cas de validation de mot de passe grâce à Kerberos.

Ces deux comportements sont configurables et peuvent être employés simultanément ou non. Ils sont présentés ici chacun leur tour.

4.1 Authentification Kerberos

Sans surprise, la mise en place côté serveur commence par créer un principal de la forme **HTTP/machine@ROYAUME**, puis l'extraire dans un fichier keytab. Attention, ce fichier doit être accessible en lecture par le processus **apache** :

```
[root@loubianka ~]# kadmin -p guillaume
kadmin> add --random-key --use-defaults HTTP/loubianka.zarb.home
guillaume@ZARB.HOME's Password:
kadmin> ext_keytab HTTP/loubianka.zarb.home
kadmin> quit
[root@loubianka ~]# chgrp apache /etc/krb5.keytab
[root@loubianka ~]# chmod 640 /etc/krb5.keytab
```

Ensuite, il faut déclarer dans la configuration d'Apache le contexte à protéger. Par exemple, l'accès aux pages détaillant la configuration du serveur :

```
<Location /server-status>
  SetHandler server-status
  # network access
  Order allow,deny
  allow from all

  # kerberos authentication
  AuthType Kerberos
  AuthName "Kerberos authentication required"
  KrbMethodNegotiate on
  Require valid-user
</Location>
```

La directive **KrbMethodNegotiate** correspond à l'authentification Kerberos, via la méthode Negotiate.

Côté client, la configuration dépend du navigateur utilisé.

Pour Mozilla et Firefox, il faut autoriser l'utilisation de Negotiate. Parmi les différentes options de configuration, accessibles via l'URL [about:config](#), il faut renseigner la clé **network.negotiate-auth.trusted-uris** avec une chaîne correspondant à la zone de confiance, typiquement le domaine local. Dans l'exemple utilisé ici, il s'agirait donc de **.zarb.home**. Il y a également une clé **network.negotiate-auth delegation-uris**, qui correspond à la possibilité de transfert de tickets, comme dans le cas d'OpenSSH. Ceci permet d'imaginer une application web capable d'agir pour le compte d'un utilisateur vis-à-vis d'une autre ressource, un serveur LDAP par exemple, sans pour autant stocker son mot de passe dans sa session, comme le fait phpLDAPAdmin.

Pour Safari, il n'y a rien à configurer. Par contre, le navigateur manifeste un comportement radicalement différent des autres clients kerberisés vis-à-vis des noms de services, tel que décrit plus loin à la page 56.

Voilà pour les navigateurs que je connais. Pour Internet Explorer, il faut que le serveur web figure dans l'une des zones autorisées à cet effet par sa configuration de sécurité, par défaut la zone **Intranet** et que l'option **Authentification Windows Intégrée** soit activée. La documentation du module Apache mentionne également Konqueror. Pour Chrome, il y a un ticket ouvert [2], il semble que le support soit en cours de finalisation.

Comme d'habitude, on peut vérifier l'évolution de la liste des tickets avant et après l'accès à la ressource demandée. Avant :

```
[guillaume@beria ~]$ klist
Credentials cache: FILE:/tmp/krb5cc_500
Principal: guillaume@ZARB.HOME

Issued Expires Principal
Apr 17 18:05:48 Apr 18 04:05:48 krbtgt/ZARB.HOME@ZARB.HOME
```

Et après :

```
[guillaume@beria ~]$ klist
Credentials cache: FILE:/tmp/krb5cc_500
Principal: guillaume@ZARB.HOME

Issued Expires Principal
Apr 17 18:05:48 Apr 18 04:05:48 krbtgt/ZARB.HOME@ZARB.HOME
Apr 17 18:06:05 Apr 18 04:05:48 HTTP/loubianka.zarb.home@ZARB.HOME
```

4.2 Validation de mot de passe

En complément, ou en remplacement de l'authentification Kerberos, le module peut se contenter de valider les mots de passe transmis via la méthode classique (Basic). Ce comportement s'obtient via la directive **KrbMethodKSPasswd** dans la configuration d'Apache. L'exemple précédent, configuré

comme suit, offre donc un mécanisme de repli pour les clients incapables pour une raison ou une autre de présenter un ticket Kerberos valide :

```
<Location /server-status>
  SetHandler server-status
  # network access
  Order allow,deny
  allow from all

  # kerberos authentication
  AuthType Kerberos
  AuthName "Kerberos authentication required"
  KrbMethodNegotiate on
  KrbMethodKSPasswd on
  Require valid-user
</Location>
```

5 OpenLDAP

OpenLDAP supporte également Kerberos et permet à la fois l'authentification et la validation de mots de passe.

5.1 Authentification Kerberos

OpenLDAP propose deux modes d'authentification : le premier, dit simple, utilise un couple DN et mot de passe, tandis que le second, SASL, permet d'utiliser un des mécanismes supportés par ce sous-système. Parmi ceux-ci, il y a justement GSSAPI, ce qui permet d'utiliser Kerberos. Il s'agit donc d'offrir un choix au client dans la façon dont il peut s'authentifier.

Le principal à créer, puis à extraire sur le serveur, est de la forme **ldap/machine@ROYAUME**. Ici encore, il faut prendre garde à ce que le serveur LDAP puisse accéder au fichier keytab, car il tourne sous une identité non privilégiée :

```
[root@loubianka ~]# kadmin -p guillaume
kadmin> add --random-key --use-defaults ldap/loubianka.zarb.home
guillaume@ZARB.HOME's Password:
kadmin> ext_keytab ldap/loubianka.zarb.home
kadmin> quit
[root@loubianka ~]# chgrp ldap /etc/krb5.keytab
[root@loubianka ~]# chmod 640 /etc/krb5.keytab
```

Il n'y a pas de configuration spécifique à mettre en place, l'authentification via SASL étant toujours activée. Par contre, il faut veiller à installer le greffon qui assure le support de GSSAPI. En effet, ceux-ci sont souvent distribués dans des paquetages séparés pour réduire les dépendances. Dans le cas de Mandriva, il s'agit du paquetage **libsasl2-plug-gssapi** pour la version 32 bits et **lib64sasl2-plug-gssapi** pour la version 64 bits.

Côté client, il est nécessaire également d'installer ce même paquetage. Ensuite, tout client LDAP capable d'utiliser le mode d'authentification SASL permet de bénéficier du SSO.

en précisant éventuellement l'utilisation de GSSAPI si plusieurs mécanismes sont disponibles pour SASL. Par exemple, **ldapwhoami**:

```
[guillaume@beria ~]$ klist
Credentials cache: FILE:/tmp/krb5cc_500
Principal: guillaume@ZARB.HOME

Issued Expires Principal
Apr 17 18:05:48 Apr 18 04:05:48 krbtgt/ZARB.HOME@ZARB.HOME
[guillaume@beria ~]$ ldapwhoami -Y GSSAPI -h loubianka.zarb.home
SASL/GSSAPI authentication started
SASL username: guillaume@ZARB.HOME
SASL SSF: 56
SASL data security layer installed.
dn:uid=guillaume,cn=gssapi,cn=auth
[guillaume@beria ~]$ klist
Credentials cache: FILE:/tmp/krb5cc_500
Principal: guillaume@ZARB.HOME

Issued Expires Principal
Apr 17 18:05:48 Apr 18 04:05:48 krbtgt/ZARB.HOME@ZARB.HOME
Apr 17 18:06:05 Apr 18 04:05:48 ldap/loubianka.zarb.home@ZARB.HOME
```

Ce premier exemple montre que l'authentification est fonctionnelle, mais l'utilisateur n'est pas identifié comme attendu : au lieu du DN de sa propre entrée, il se voit attribuer un autre identifiant dans la branche **cn=gssapi,cn=auth**. Ceci peut être suffisant pour un contrôle d'accès faisant juste la différence entre un accès anonyme et un accès authentifié, mais pas pour des ACL plus spécifiques, typiquement celles contrôlant l'accès aux entrées utilisateur. Il faut donc mettre en place une correspondance, via une expression régulière dans la configuration du serveur LDAP, **/etc/openldap/slapd.conf**:

```
authz-regexp "uid=(^,)*,cn=gssapi,cn=auth" "uid=$1,ou=users,dc=zarb,dc=home"
```

Et le tour est joué :

```
[guillaume@beria ~]$ ldapwhoami -Y GSSAPI -h loubianka.zarb.home
SASL/GSSAPI authentication started
SASL username: guillaume@ZARB.HOME
SASL SSF: 56
SASL data security layer installed.
dn:uid=guillaume,ou=users,dc=zarb,dc=home
```

Il devient alors trivial d'édition facilement une entrée quelconque avec son éditeur de texte préféré, grâce à **ldapvi**:

```
[guillaume@beria ~]$ ldapvi -Y GSSAPI -h loubianka.zarb.home
uid=guillaume
SASL/GSSAPI authentication started
SASL username: guillaume@ZARB.HOME
SASL SSF: 56
SASL data security layer installed.
1 entry read
add: 0, rename: 0, modify: 1, delete: 0
Action? [y/YqQvVebB*rsf+?] y
Done.
```

5.2 Validation de mot de passe

La documentation d'OpenLDAP parle de mode passe-tout (*pass-through*) pour désigner un fonctionnement dans lequel le serveur LDAP délégué la validation du mot de passe à une entité extérieure, qui peut être un serveur Kerberos. Dans ce cas, le serveur LDAP n'offre aucune alternative au client, qui effectue une opération BIND classique, mais le serveur vérifie ce mot de passe via SASL au lieu d'effectuer une comparaison avec un mot de passe stocké. Typiquement, ceci permet d'éviter d'avoir à synchroniser des mots de passe entre deux services.

La mise en place de ce mécanisme passe par l'utilisation d'un daemon supplémentaire, **saslauthd**, chargé de répondre aux requêtes d'authentification. La configuration, dans le fichier **/etc/sysconfig/saslauthd**, se limite à indiquer l'utilisation de Kerberos :

```
# Authentications mechanism (for list see saslauthd -v)
SASL_AUTHMECH=kerberos5
# Hostname for remote IMAP server (if rimap auth mech is used)
SASL_MECH_OPTIONS=
# Extra options (for list see saslauthd -h)
SASLAUTHD_OPTS=
```

On peut ensuite lancer le daemon, puis tester l'authentification directement, via le client **testsaslauthd** :

```
[root@loubianka ~]# service saslauthd start
Lancement de saslauthd: [ OK ]
Creating hardlink from /var/run/saslauthd/mux to /var/spool/postfix/
Var/run/saslauthd/
[root@loubianka ~]# testsaslauthd -u guillaume@ZARB.HOME -p secret
0: OK "Success."
```

Il ne reste plus qu'à indiquer à **slapd** qu'il doit utiliser utiliser **saslauthd**, dans son fichier de configuration spécifique pour SASL, **/etc/sasl2/slappd.conf**, qui se réduit à cette ligne unique :

```
pwcheck_method: saslauthd
```

Enfin, pour chaque utilisateur concerné, il faut mettre en place dans l'attribut **userPassword** une chaîne de la forme **{SASL}user@royaume**, au lieu du mot de passe attendu. Ceci permet donc de sélectionner les utilisateurs concernés et de garder une authentification interne pour d'autres, typiquement les comptes administratifs spécifiques au serveur LDAP.

Pour plus de détails concernant ces deux façons d'intégrer Kerberos et OpenLDAP, voir mon précédent article [3] sur le sujet.

6 NFS

NFS est le système de fichiers réseau historique du monde Unix. De même que pour les autres vénérables applications évoquées plus haut (**telnet**, **rsh**, etc.), ceci se ressent largement au niveau de la sécurité.

En effet, le modèle d'authentification classique de NFS est basé sur les identificateurs numériques (UID) transmis par le client. Il n'est donc envisageable que vers des machines de confiance, sur lesquelles aucun utilisateur n'est capable de sélectionner comme il l'entend ce fameux identificateur. Et donc, qu'il ne possède pas les priviléges d'administration système, via le compte root, **sudo**, ou tout autre mécanisme. Si ce modèle était sans doute pertinent auparavant, dans le cadre de stations de travail fixes, avec des utilisateurs possédant des comptes dépourvus de priviléges, il l'est de moins en moins aujourd'hui, avec des utilisateurs nomades, ou qui gèrent eux-mêmes leurs machines. La méthode qui consiste à partitionner l'ensemble du système de fichiers exporté en sous-unités, chacune accessible à un groupe restreint de machines, ne permet que de limiter les possibilités d'usurpation d'identité. De plus, il reste très fragile d'authentifier une machine par son nom ou son adresse IP. Et surtout, la gestion d'un parc un peu conséquent devient vite un cauchemar...

Pour pallier cette faiblesse, NFSv4 a introduit deux changements. D'abord, ce ne sont plus les identificateurs numériques qui circulent entre le client et le serveur, mais des noms d'utilisateurs, qualifiés par un domaine (encore un...). En plus de rendre le protocole largement plus indépendant du monde Unix, ceci permet surtout de découpler l'identité de l'utilisateur entre le client et le serveur, en utilisant le nom comme seul point commun. Ensuite, il devient possible d'utiliser GSSAPI pour prouver l'authenticité de ce nom d'utilisateur, ce qui fait reposer la confiance sur un mécanisme fiable et conçu à cet effet, plutôt que sur une adresse IP.

Ce support de GSSAPI et donc de Kerberos, a ensuite été rétrocarré dans NFSv3 et s'emploie de la même manière. Néanmoins, tous les exemples utilisés ici se basent sur NFSv4.

6.1 Mise en place

La mise en place de Kerberos pour NFSv4 est largement plus complexe que pour les autres applications. Et notamment parce que dans ce cas, l'application, c'est le noyau Linux lui-même. En effet, même si l'existe aujourd'hui des implémentations en espace utilisateur, elles sont soit obsolètes, soit expérimentales (ganesha) et c'est le noyau qui constitue le serveur et le client NFS. Et tout ce qui touche le noyau est généralement plus complexe à mettre en œuvre.

Tout d'abord, l'implémentation de GSSAPI dans le noyau ne supporte encore que DES comme algorithme de chiffrement. Dans le même temps, les différentes implémentations de Kerberos font disparaître progressivement le support de ce même algorithme et ne le reconnaissent même plus par défaut. C'est le cas de l'implémentation MIT à partir de la version 1.8 et de Heimdal à partir de la version 1.3. Dans ce cas, il faut donc impérativement d'abord activer le support

de DES au niveau de la configuration Kerberos pour toutes les machines concernées (KDC, serveur NFS, client NFS), puis s'assurer que les différents principaux impliqués possèdent une clé utilisant cet algorithme.

Le premier pré-requis correspond à la mise en place de la directive **allow_weak_crypto** dans la section **libdefaults** du fichier de configuration :

```
[libdefaults]
default_realm = ZARB.HERE
allow_weak_crypto = true
```

Le second pré-requis nécessite de modifier la liste des clés par défaut lors de la création d'un principal, dans la section **kadmin** du fichier de configuration, en s'assurant que la variable **default_keys** comporte la valeur **des-cbc-crc:pw-salt**. Ainsi, tout principal possédera une clé DES dès sa création :

```
[kadmin]
default_keys = des-cbc-crc:pw-salt aes256-cts-hmac-sha1-96:pw-salt
des3-cbc-sha1:pw-salt arcfour-hmac-md5:pw-salt
```

Remarque

Techniquement, il est possible d'ajouter des clés à un principal déjà existant, mais ceci aboutit à des clés correspondant à des mots de passe différents, ce qui n'est pas une très bonne idée.

Attention, à la différence d'une application classique, l'utilisateur final n'interagit pas directement avec le serveur NFS, il interagit d'abord avec sa propre machine, qui intègre en fait un système de fichiers distant localement, via une opération de montage. Il y a un acteur supplémentaire, sous la forme d'un daemon tournant sur la machine de l'utilisateur, nécessitant lui aussi son propre principal, stocké dans son propre fichier keytab. Il y a donc cette fois-ci un principal machine pour le serveur et un principal machine pour le client, tous les deux de la forme **nfs/machine@ROYAUME**.

6.2 Côté serveur

Une fois les problèmes de compatibilité réglés, la mise en place d'un principal dans un fichier keytab reste très classique :

```
[root@loubianka ~]# kadmin -p guillaume
kadmin> add --random-key --use-defaults nfs/loubianka.zarb.home
guillaume@ZARB.HERE's Password:
kadmin> ext_keytab nfs/loubianka.zarb.home
```

Par contre, il est utile de vérifier que le principal possède bien une clé DES et que celle-ci est présente dans le fichier keytab :

Vno	Type	Principal
2	des-cbc-crc	nfs/loubianka.zarb.home@ZARB.HOME
2	des3-cbc-sha1	nfs/loubianka.zarb.home@ZARB.HOME
2	arcfour-hmac-md5	nfs/loubianka.zarb.home@ZARB.HOME
2	aes256-cts-hmac-sha1-96	nfs/loubianka.zarb.home@ZARB.HOME

Ensuite, il est nécessaire d'exporter un système de fichiers via NFSv4 et d'utiliser GSSAPI comme modèle de sécurité. Ceci se fait en spécifiant respectivement les options **fsid=0** et **sec=krb5**, en plus des options habituelles, dans le fichier **/etc(exports)**:

```
/home * .zarb.home(rw,no_subtree_check,fsid=0,sec=krb5)
```

Le daemon **rpc.idmapd**, en charge du mécanisme de correspondance d'identité, doit être configuré. D'abord, il faut un domaine NFSv4, c'est-à-dire une chaîne de caractères arbitraire, mais identique à celui utilisé par le client. Comme pour le royaume Kerberos, l'usage est d'utiliser le nom de domaine DNS. Ensuite, il faut une stratégie de conversion de nom en identifiant numérique. Le plus simple est d'utiliser la méthode par défaut, basée sur NSS. Cette configuration se fait dans le fichier **/etc/idmapd.conf**:

```
Verbosity = 0
PipesDirectory = /var/lib/nfs/rpc_pipefs
Domain = zarb.home

[Mapping]
Nobody-User = nobody
Nobody-Group = nogroup

[Translation]
Method = nsswitch
```

Enfin, il faut veiller à ce que les processus nécessaires soient bien activés. Les processus suivants sont nécessaires côté serveur :

- **rpc.idmapd** pour la correspondance d'identité.
- **rpc.gssd** et **rpc.svcgssd** pour l'authentification.

De plus, **rpcbind** reste nécessaire pour le lancement de ces processus, même si NFSv4 n'utilise pas le port mapper.

Dans le cas d'une distribution Mandriva, ceci se fait en renseignant les variables **NEED_IDMAPD** et **NEED_GSSD** dans le fichier **/etc/sysconfig/nfs-common** et **NEED_SVCGSSD** dans le fichier **/etc/sysconfig/nfs-server**.

Ensuite, il faut lancer les différents services nécessaires, à savoir **rpcbind**, **nfs-common** et **nfs-server**, dans cet ordre :

```
[root@loubianka ~]# service rpcbind start
Lancement de rpcbind: [ OK ]
[root@loubianka ~]# service nfs-common start
Lancement de NFS common utilities: [ OK ]
```

```
Lancement de NFS common utilities: [ OK ]
Lancement de rpc.idmapd: [ OK ]
Lancement de rpc.gssd: [ OK ]
[root@loubianka ~]# service nfs-server start
Exportation des répertoires pour NFS kernel daemon...
Lancement de NFS kernel daemon: [ OK ]
Lancement du service nfssd: [ OK ]
Lancement du service rpc.svcgssd: [ OK ]
Reloading rpc.idmapd: [ OK ]
Lancement de rpc.mountd: [ OK ]
```

6.3 Côté client

La mise en place du fichier keytab se fait comme d'habitude :

```
[root@loubianka ~]# kadmin -p guillaume
kadmin> add --random-key --use-defaults nfs/beria.zarb.home
guillaume@ZARB.HOME's Password:
kadmin> ext_keytab nfs/beria.zarb.home
```

Comme sur le serveur, il est préférable de vérifier le contenu de ce fichier :

```
[root@beria ~]# ktutil list
FILE:/etc/krb5.keytab:
Vno Type Principal Aliases
1 des-cbc-crc nfs/beria.zarb.home@ZARB.HOME
1 des3-cbc-sha1 nfs/beria.zarb.home@ZARB.HOME
1 arcfour-hmac-md5 nfs/beria.zarb.home@ZARB.HOME
1 aes256-cts-hmac-sha1-96 nfs/beria.zarb.home@ZARB.HOME
```

Ensuite, il faut s'assurer que les différents processus nécessaires soient bien activés. Seuls **rpcbind**, **rpc.idmapd** et **rpc.gssd** sont nécessaires, il faut donc renseigner les variables **NEED_IDMAPD** et **NEED_GSSD** dans le fichier **/etc/sysconfig/nfs-common**, puis lancer les services **rpcbind** et **nfs-common** :

```
[root@beria ~]# service rpcbind start
Lancement de rpcbind: [ OK ]
[root@beria ~]# service nfs-common start
Lancement de NFS common utilities: [ OK ]
Lancement de rpc.idmapd: [ OK ]
Lancement de rpc.gssd: [ OK ]
```

6.4 Authentification des utilisateurs

Une fois ceci fait, il est alors possible d'effectuer le montage de façon classique, en spécifiant l'utilisation de Kerberos comme option de sécurité. Attention cependant, les systèmes de fichiers sont exportés en NFSv4 dans une arborescence propre, c'est donc / qu'il faut monter et pas /home :

```
[root@beria ~]# mount -t nfs -o sec=krb5 loubianka:/ /mnt/home
```

L'utilisation du principal machine par le client pour effectuer le montage peut être vérifiée en inspectant son propre cache d'authentification Kerberos, qui est stocké dans un fichier situé dans /tmp :

```
[root@beria ~]# klist -c /tmp/krb5cc_machine_ZARB_HOME
Credentials cache: FILE:/tmp/krb5cc_machine_ZARB_HOME
Principal: nfs/beria.zarb.home@ZARB_HOME

Issued          Expires        Principal
Aug 14 16:24:33 Aug 15 16:24:33  krbtgt/ZARB_HOME@ZARB_HOME
Aug 14 16:24:33 Aug 15 16:24:33  nfs/beria.zarb.home@ZARB_HOME
```

L'utilisateur peut donc maintenant tenter d'accéder à ses fichiers. Néanmoins, sans ticket Kerberos, la tentative d'accès au point de montage bloque indéfiniment. Un coup d'œil aux logs montre des messages d'erreur de ce type :

```
Aug 15 14:31:20 beria rpc.gssd[9787]: getting credentials for client
with uid:
      500 for server ioubianka.zarb.home
Aug 15 14:31:20 beria rpc.gssd[9787]: WARNING: Failed to create krb5
context
      for user with uid 500 for server ioubianka.zarb.home
```

L'explication est assez simple : pour accéder à un système de fichiers kerberisé, il faut impérativement posséder un ticket Kerberos valide. Une fois que l'utilisateur s'authentifie, l'accès au système de fichiers devient possible et un coup d'œil au cache d'authentification montre qu'un ticket pour le service NFS a été automatiquement obtenu.

```
[guillaume@beria ~]$ kinit
guillaume@ZARB_HOME's Password:
[guillaume@beria ~]$ klist
Credentials cache: FILE:/tmp/krb5cc_500
Principal: guillaume@ZARB_HOME

Issued          Expires        Principal
Aug 15 14:53:06 Aug 16 08:53:06  krbtgt/ZARB_HOME@ZARB_HOME
[guillaume@beria ~]$ ls /mnt/home/
[guillaume@beria ~]$ ls /mnt/home/
guillaume/ lost+found/
[guillaume@beria ~]$ klist
Credentials cache: FILE:/tmp/krb5cc_500
Principal: guillaume@ZARB_HOME

Issued          Expires        Principal
Aug 15 14:53:06 Aug 16 08:53:06  krbtgt/ZARB_HOME@ZARB_HOME
Aug 15 14:53:06 Aug 16 08:53:06  nfs/ioubianka.zarb.home@ZARB_HOME
```

Les permissions effectives d'accès au système de fichiers correspondent à celles de l'utilisateur sur le serveur, même si celui-ci correspond à un UID différent. En effet, contrairement aux versions précédentes du protocole, ce ne sont plus des identifiants numériques qui circulent entre le client et le serveur, mais des noms d'utilisateurs, qualifiés par un nom de domaine (encore un) NFSv4. Sur chaque machine, c'est le processus **rpc.idmapd** qui s'occupe de convertir ce nom d'utilisateur en identifiant numérique local. Lorsque Kerberos est utilisé, c'est le principal de l'utilisateur qui détermine son nom d'utilisateur NFS. Les deux conséquences principales de ce mécanisme, c'est que d'une part les identifiants numériques d'un utilisateur peuvent varier entre le client et le serveur,

d'autre part qu'il n'est plus nécessaire d'avoir confiance dans le client pour identifier l'utilisateur, puisque celui-ci doit présenter un ticket Kerberos pour s'authentifier.

Ici, le principal **guillaume@ZARB_HOME** est converti par le daemon **rpc.svcgssd** sur le serveur en nom d'utilisateur NFSv4 **guillaume@zarb.home**, puis le daemon **rpc.idmapd** recherche ensuite un utilisateur **guillaume** et obtient l'UID de celui-ci par l'appel système **getpwname()**, puisqu'il est configuré pour utiliser NSS (figure 3).

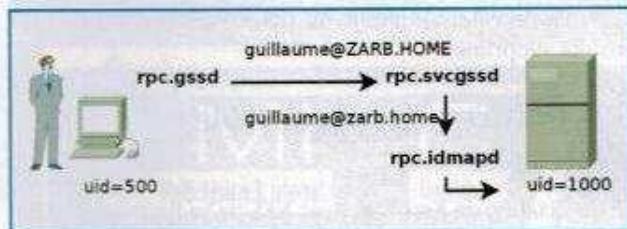


Figure 3 : Accès NFS authentifié par Kerberos, utilisateur connu

Qu'en est-il de l'utilisateur root ? Contrairement à ce que l'explication précédente pourrait laisser supposer, il peut également accéder au système de fichiers, alors qu'il n'a pas de ticket Kerberos. En fait, par défaut, c'est le ticket Kerberos obtenu lors du montage via le principal machine qui est utilisé pour cet utilisateur.

Remarque

Ceci peut être modifié par l'option **-n** de **rpc.gssd**.

Dans le cas présent, ce principal **nfs/beria.zarb.home@ZARB_HOME** devient le nom d'utilisateur NFSv4 **nfs@zarb.home**, mais l'utilisateur **nfs** n'existe pas sur le serveur, c'est l'utilisateur par défaut, à savoir **nobody**, qui est utilisé (figure 4).

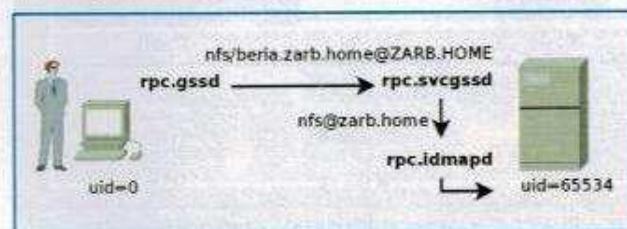


Figure 4 : Accès NFS authentifié par Kerberos, utilisateur inconnu

Attention cependant, il reste nécessaire de disposer d'un ticket Kerberos valide pour obtenir cet accès. Contrairement à d'autres systèmes, comme AFS par exemple, il n'y a pas d'accès anonyme possible. Un daemon tournant sous une identité non privilégiée, comme apache par exemple, ne bénéficie donc pas du principal machine comme l'utilisateur root et sera incapable d'accéder à un tel système de fichiers, comme par exemple les pages personnelles des utilisateurs stockées dans leur répertoire **~/public_html**. Il faut donc

dans ce cas mettre en place un mécanisme d'obtention et de maintien de tickets Kerberos non interactif, comme k5start [4] par exemple.

Un autre détail intéressant, c'est qu'effacer son ticket Kerberos après l'accès initial n'empêche pas les accès suivants au système de fichiers. La FAQ [5] du projet Linux NFSv4 précise que ceci est dû au fait que le kernel garde en fait une copie du contexte GSSAPI négocié avec le serveur et qu'effacer le cache d'authentification en espace utilisateur n'affecte pas cette copie. Autrement dit, l'accès reste possible pour toute la durée de validité initiale du ticket, même si celui-ci est ensuite supprimé.

6.5 Chiffrement et contrôle d'intégrité

L'utilisation de Kerberos avec NFS ne se limite pas à l'authentification des utilisateurs. Elle permet également de sécuriser les communications entre le client et le serveur contre les menaces extérieures, à savoir l'écoute et la modification du trafic réseau.

La mise en place de l'une ou l'autre de ces mesures se fait en modifiant l'option **sec**, dans le fichier **/etc(exports)**. La valeur **krb5p** correspond au chiffrement, tandis que la valeur **krb5i** correspond au contrôle d'intégrité. L'utilisation de **tcpdump** suffit à constater l'efficacité de la première mesure, l'autre est plus difficile à constater expérimentalement.

Une mesure rudimentaire permet de s'apercevoir que les performances ne sont guère affectées. Le tableau 1 donne le résultat de tests pour la commande **dd if=/dev/zero of=/mnt/file bs=32k count=10000**, qui montre une pénalité de débit de 3% pour le chiffrement et de 1% pour le contrôle d'intégrité.

Mode	Temps	Débit
krb5p	30,8241s	10,6 MB/s
krb5i	30,2268s	10,8 MB/s
krb5	29,995s	10,9 MB/s

Tableau 1 : Comparaison des performances d'écriture

mécanisme de SSO avec l'ouverture d'une session utilisateur. Ceci permet donc d'arriver au même niveau d'intégration de Kerberos qu'une machine Windows faisant partie d'un domaine Active Directory, par exemple.

Il existe en fait deux modules, **pam_krb5** [6] et **pam-krb5** [7], qui ne diffèrent en fait que sur des détails. Il s'agit essentiellement du support de Kerberos 4 et AFS, qui n'existe que dans le premier. Ils s'emploient donc de la même manière, à quelques différences de syntaxe près.

Voici un exemple de configuration, dans laquelle deux des rôles assurés par PAM (authentification et changement de mot de passe) sont configurés pour utiliser d'abord la base utilisateur locale, à savoir les fichiers **/etc/passwd** et **/etc/shadow**, puis Kerberos si l'utilisateur n'y est pas défini, ou si le mot de passe donné ne correspond pas. Pour les deux autres rôles (autorisation et ouverture de session), Kerberos n'intervient pas. L'utilisation de l'option **minimum_uid** évite des requêtes inutiles vers le KDC pour des utilisateurs système, typiquement root.

```
auth required pam_env.so
auth sufficient pam_unix.so likeauth nullok
auth sufficient pam_krb5.so use_first_pass minimum_uid=500
auth required pam_deny.so

account sufficient pam_unix.so
account required pam_deny.so

password sufficient pam_unix.so md5 shadow
password sufficient pam_krb5.so use_authok
password required pam_deny.so

session required pam_limits.so
session optional pam_unix.so
```

D'autres options de configuration, correspondant aux options de la commande **kinit**, peuvent être définies dans cette configuration PAM. Dans le cas du module **pam-krb5**, elles peuvent également être définies dans une section spécifique du fichier de configuration de Kerberos, **/etc krb5.conf**. Ainsi, pour obtenir des tickets transmissibles et renouvelables pendant une semaine :

```
[appdefaults]
pam = {
    forwardable = true
    renew_lifetime = 7d
}
```

Dans le cas d'un utilisateur nomade, sur un portable par exemple, il n'est pas souhaitable de dépendre de la disponibilité du KDC pour authentifier un utilisateur, de façon à pouvoir également s'authentifier hors-ligne. Il faut donc utiliser un compte utilisateur local, avec un mot de passe local. Mais il peut être intéressant de pouvoir néanmoins bénéficier de l'intégration de Kerberos quand le KDC est disponible. En rendant l'authentification locale nécessaire (au lieu de suffisante) de

7 PAM

PAM (*Pluggable Authentication Modules*) est le système d'authentification modulaire utilisé sous Linux. Kerberos peut être utilisé à ce niveau, non pas comme mécanisme d'authentification, puisqu'il n'est pas possible de passer un ticket, mais pour valider le mot de passe saisi par l'utilisateur. L'intérêt majeur, c'est de récupérer ainsi automatiquement un TGT au passage et donc de faire correspondre l'initialisation du

façon à continuer le traitement du mot de passe même en cas de succès, en rendant l'authentification Kerberos optionnelle et en supprimant la clause finale d'échec automatique, on arrive à un tel résultat :

```
auth required pam_env.so
auth required pam_unix.so likeauth nullok
auth optional pam_krb5.so use_first_pass minimum_uid=500

password required pam_unix.so md5 shadow
password optional pam_kerberos.so use_authok
```

Bien entendu, ceci n'est qu'une astuce de configuration, qui suppose que le mot de passe local soit identique au mot de passe Kerberos. Et il ne suffit pas d'utiliser une configuration similaire pour le changement de mot de passe pour obtenir une garantie réelle qu'ils le resteront. Une véritable solution au problème de disponibilité d'une source d'authentification centralisée sur une machine nomade est à chercher du côté du module **pam_ccreds**, qui fournit une solution générique de mise en cache, ou alors de SSSD [8], qui inclut notamment cette fonctionnalité. C'est ce type de mécanisme qui est mis en œuvre sous Windows, pour les machines intégrées à un domaine Active Directory. Attention cependant aux implications en matière de sécurité, puisque ces mécanismes impliquent de stocker des empreintes des mots de passe sur des postes de travail utilisateur, généralement beaucoup plus exposés à la compromission que les serveurs d'authentification.

Pour finir sur cette partie, il faut rappeler que PAM, comme Kerberos, ne gère que la partie authentification des comptes utilisateurs, pas leur définition. Pour obtenir une solution complète, il est nécessaire de coupler Kerberos avec une base de comptes et l'exemple d'un tel couplage avec un annuaire LDAP est présenté dans la partie suivante (page 61). ■

Références

- [1] <http://modauthkerb.sourceforge.net/>
- [2] <http://code.google.com/p/chromium/issues/detail?id=19>
- [3] <http://www.unixgarden.com/index.php/administration-reseau/>
- Encore-plus-loin-avec-openldap :
- [4] <http://www.eyrie.org/~eagle/software/kstart/k5start.html>
- [5] http://www.citi.umich.edu/projects/nfsv4/linux/faq/#krb5_006
- [6] https://fedorahosted.org/pam_krb5
- [7] http://www.eyrie.org/~eagle/software/pam_krb5
- [8] <http://fedorahosted.org/sssd>

MISC 58
Actuellement
en kiosque !

SÉCURITÉ ET ARCHITECTURE PC : L'IMPOSSIBLE CONFIANCE ?



DISPONIBLE
CHEZ VOTRE MARCHAND
DE JOURNAUX JUSQU'AU
23 DÉCEMBRE 2011 ET SUR :

www.ed-diamond.com

KERBEROS, LE SSO UNIVERSEL

3- SUBTILITÉS DIVERSES ET MISE EN ŒUVRE AVANCÉE

par Guillaume Rousse

La partie précédente a montré l'utilisation de Kerberos au sein de différentes applications, à partir de cas d'école. Néanmoins, les cas réels sont généralement plus complexes et certains détails viennent parfois compliquer la donne. Cette partie présente donc certains problèmes de mise en œuvre de Kerberos et la façon d'y remédier.

1 Identification de service

Nous avons vu précédemment que les noms de principaux à utiliser pour les services kerberisés étaient basés sur le nom de la machine qui les héberge. Cependant, il existe de nombreuses manières de désigner la même machine :

- son adresse IP.
- son nom pleinement qualifié.
- son nom non qualifié.

Bien entendu, une machine peut posséder plusieurs adresses et plusieurs noms, notamment dans le cas des hôtes virtuels HTTP, ce qui augmente encore le nombre de possibilités. Or, Kerberos fournit une authentification réciproque, c'est-à-dire qu'il garantit non seulement l'identité de l'utilisateur au service, mais également celle du service à l'utilisateur. Il faut donc pouvoir établir un lien entre l'identifiant utilisé par cet utilisateur, qui peut donc être variable, et la preuve de cette identité, qui elle correspond à un identifiant précis.

Pour rappeler un cas mieux connu, le problème se pose exactement de la même façon pour les certificats x509, utilisés dans le protocole SSL ou TLS

pour garantir l'identité du serveur auquel ils sont rattachés. Ces certificats possèdent un champ CN à valeur unique, qui doit correspondre strictement au nom utilisé par le client pour que la négociation réussisse.

Dans le cas de Kerberos, il faut donc pouvoir convertir un nom de service en nom de principal. Cette résolution, pour reprendre le terme utilisé pour les conversions de nom de machine en adresse IP, peut se faire soit côté client, par un mécanisme de canonicalisation, soit côté serveur, par la mise en place d'alias. Et si aucune des deux solutions n'est disponible, il faut contourner le problème.

1.1 Canonicalisation

La solution historique à ce problème consiste à laisser la charge de cette résolution au client, c'est-à-dire que c'est à lui de déterminer quel est le principal exact à réclamer au KDC, via une étape de normalisation, ou canonicalisation, du nom de machine demandé.

Le mécanisme utilisé à cet effet dans les implémentations Unix, bien qu'unaniment considéré comme peu satisfaisant au niveau de la sécurité, consiste à utiliser le DNS et à prendre comme nom canonique celui renvoyé par une double résolution, de nom en adresse IP d'abord, d'adresse IP en nom ensuite.

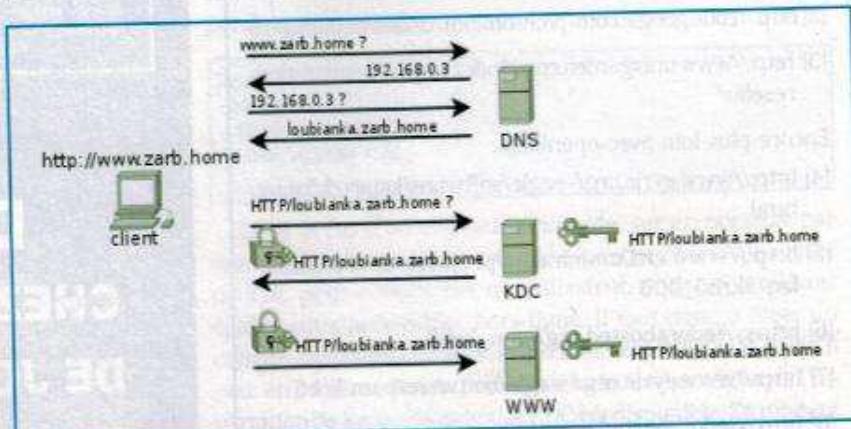


Figure 1 : Résolution de principal côté client : canonicalisation

Autrement dit, le nom canonique d'une machine c'est l'enregistrement de type A associé à l'adresse IP associée au nom demandé par l'utilisateur.

La figure 1 illustre ce fonctionnement, pour un hôte virtuel **HTTP www.zarb.home** hébergé sur la machine **loubianka.zarb.home**. Le navigateur web effectue la canonicalisation du nom de l'hôte virtuel fourni dans l'URL, demande un ticket pour le principal **HTTP/loubianka.zarb.home@ZARB.HOME** au KDC et présente ce ticket au serveur web.

Bien sûr, si ce comportement est la règle pour la majorité des clients tournant sous Unix, il y a aussi des exceptions. Safari, par exemple, le navigateur web natif de Mac OS X, n'effectue aucune canonicalisation et demande un ticket pour un principal correspondant exactement au nom saisi par l'utilisateur dans sa requête. Dans l'exemple précédent, il s'agit du principal **HTTP/www.zarb.home@ZARB.HOME**, inconnu du KDC et la demande de ticket ne peut donc pas aboutir. Il est intéressant de noter au passage que la pile Kerberos de Mac OS X n'est pas spécifique, il s'agit de l'implémentation du MIT. Autrement dit, cette différence de comportement est due à l'application, pas à la bibliothèque sous-jacente.

La conséquence majeure de ce comportement, que l'on peut considérer comme majoritaire dans un environnement Unix, c'est qu'en plus d'une bonne synchronisation des horloges, il faut également maîtriser l'infrastructure DNS en place sur l'ensemble des machines du royaume. En effet, toutes les machines doivent pouvoir effectuer des résolutions de nom inverses cohérentes, sous peine d'échecs à répétition lors des négociations de tickets.

1.2 Alias de principaux

Une solution plus moderne à ce problème consiste à déporter la charge de la résolution du nom de principal au KDC, via la mise en place d'alias. Dans ce cas, le serveur est capable d'identifier le principal pour lequel délivrer un ticket, même si celui-ci ne correspond pas exactement au nom demandé par le client.

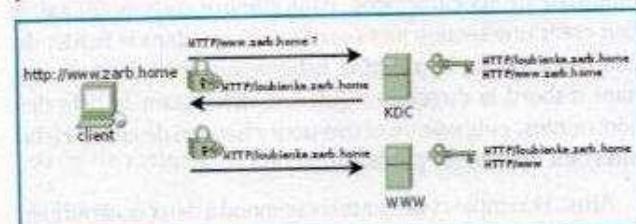


Figure 2 : Résolution de principal côté serveur : alias

```
[root@loubianka ~]# kadmin -p guillaume
kadmin> modify --alias=HTTP/www.zarb.home HTTP/loubianka.zarb.home@ZARB.HOME
guillaume@ZARB.HOME's Password:
kadmin> get -o PrincipalAliases HTTP/loubianka.zarb.home
```

```
Principal: HTTP/loubianka.zarb.home@ZARB.HOME
Aliases: HTTP/www.zarb.home@ZARB.HOME
kadmin> exit
[root@loubianka ~]# ktutil list
FILE:/etc/krb5.keytab:
Name          Type           Principal                Aliases
arcfour-hmac-md5  HTTP/loubianka.zarb.home@ZARB.HOME
aes256-cts-hmac-sha1-96  HTTP/loubianka.zarb.home@ZARB.HOME
des3-cbc-sha1   HTTP/loubianka.zarb.home@ZARB.HOME
```

1.3 Principaux multiples

Une dernière possibilité pour gérer le problème, lorsque les deux précédentes solutions ne sont pas disponibles, consiste à le contourner. En effet et contrairement à x509, il est tout à fait possible d'affecter plusieurs principaux à une même ressource. Il suffit pour cela de les déployer simultanément dans le fichier keytab du serveur et celui-ci sélectionnera automatiquement celui correspondant au ticket qui lui est présenté par le client.

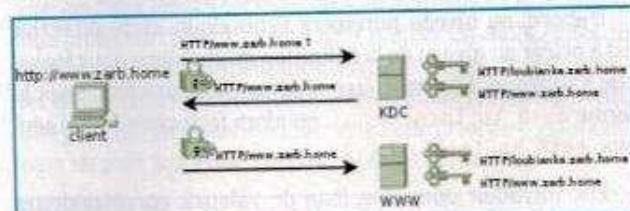


Figure 3 : Pas de résolution de principal

2 Transformation de nom d'utilisateur

Un nom d'utilisateur, pour Kerberos, c'est un principal, donc un utilisateur qualifié par un royaume. Dans le cas où l'application sous-jacente utilise ce nom, celui-ci doit généralement être converti, pour correspondre au format usuel de l'application.

Cette conversion peut être le fait de l'application, quand celle-ci est au fait du mécanisme d'authentification. C'est le cas d'OpenLDAP, par exemple, comme vu précédemment, par le biais de la directive **authz-regexp**, permettant de convertir ce nom en DN. Par contre, quand Kerberos n'est qu'une possibilité parmi d'autres, comme dans le cas d'une application web déléguant l'authentification au serveur web, ceci n'est pas possible. Dans le cas de Nagios ou Cacti, par exemple, l'application récupère dans la variable d'environnement **REMOTE_USER** un nom de type **utilisateur@ROYAUME** sans pouvoir identifier le besoin d'appliquer un traitement spécifique.

Le module **mod_krb5** permet d'effectuer lui-même cette conversion, lorsque la directive **KrbLocalUserMapping**

est vraie. Cette conversion se base en fait sur la fonction `krb5_aname_to_localname` de l'API Kerberos, décrite dans la page de manuel du même nom. Cette fonction consiste à retirer le nom du royaume du principal, mais uniquement si celui-ci correspond au royaume par défaut, c'est-à-dire celui configuré par la directive `default_realm` dans la configuration Kerberos du service. Autrement dit, pour un domaine par défaut `ZARB.HOME`, on obtient les conversions suivantes :

- `utilisateur@ZARB.HOME` devient `utilisateur`.
- `utilisateur/admin@ZARB.HOME` devient `utilisateur/admin`.
- `utilisateur@OTHER.HOME` devient `utilisateur@OTHER.HOME`.

Ce comportement peut être modifié via des règles de conversion introduites par la directive `auth_to_local` dans la configuration Kerberos. Ce mécanisme est décrit succinctement dans [krb5.conf\(5\)](#) et un peu plus en détail dans le guide de l'administrateur, mais reste relativement obscur.

D'abord, au niveau purement syntaxique, cette directive est à placer au niveau de la définition du royaume pour lequel elle s'applique. Elle peut être répétée plusieurs fois, sous la forme `auth_to_local = ...`, ou alors factorisée en un seul bloc `auth_to_local = { }`.

Elle introduit donc une liste de valeurs, correspondant chacune à une règle de conversion. Ces règles sont appliquées dans l'ordre dans lequel elles sont définies, jusqu'à ce que l'une d'elle s'applique. Si aucune ne correspond, aucune conversion n'est effectuée.

Ces règles peuvent être de trois types :

- DB : fichier
- RULE : expression
- DEFAULT

Le premier type ne semble être utilisé nulle part et la documentation précise d'ailleurs que le support pour cette fonctionnalité n'est pas disponible par défaut...

Le second type s'utilise avec une expression à la syntaxe cryptique de la forme `[nombre:gabarit](motif)(substitution)` :

- **nombre** correspond au nombre de composants des principaux auxquels la règle correspond.
- **gabarit** correspond au gabarit à appliquer à ces différents composants, chacun d'eux étant référencé sous la forme 1, 2, ...
- **motif** est une expression régulière optionnelle, qui sert à filtrer le résultat obtenu.
- **substitution** est une expression régulière optionnelle, qui sert à transformer le résultat obtenu.

Enfin, le dernier type correspond à la règle par défaut, qui n'est valable que pour des principaux à un seul composant.

Voici un exemple, composé de trois règles, gérant respectivement les principaux avec une instance `admin`, les principaux du domaine `OTHER.HOME`, puis tous les autres cas :

```
[realms]
ZARB.HOME = {
    auth_to_local = RULE:[2:$1:$2]("^.*;admin$)/;admin//"
    auth_to_local = RULE:[1:$1$0]("^.*@OTHER.HOME$)/0.*//"
    auth_to_local = DEFAULT
}
```

Une fois cette configuration en place, on obtient les conversions suivantes :

- `utilisateur@ZARB.HOME` devient `utilisateur`.
- `utilisateur/admin@ZARB.HOME` devient `utilisateur/admin`.
- `utilisateur@OTHER.HOME` devient `utilisateur`.

Pour conclure sur cette section, la documentation mentionne également la possibilité de mettre en place une table de correspondance directe, via la directive `auth_to_local_names`, qui peut se révéler plus simple d'emploi pour des cas simples.

3 Mise en œuvre avancée

Maintenant que nous avons vu les bases d'un déploiement fonctionnel de Kerberos, il est possible d'examiner certaines options de mise en œuvre plus avancées, destinées à améliorer la disponibilité, la sécurité, ou l'intégration avec le reste de l'infrastructure.

3.1 Politique de mots de passe

Le serveur de changement de mots de passe permet de mettre en place une politique de validation. Ceci permet d'imposer certains critères minimaux de robustesse et donc, de refuser des mots de passe jugés trop faibles vis-à-vis de cette politique.

Par défaut, cette politique consiste juste en une longueur minimale de six caractères. Pour étendre cette politique, il faut créer une section `password_quality` dans le fichier de configuration de la machine hébergeant `kpasswdd`, contenant d'abord la directive `policies` définissant la liste des contraintes, puis une directive pour chacune de celles-ci, lui affectant une valeur précise.

Ainsi, l'exemple ci-dessous correspond à deux contraintes :

- une longueur minimale de 8 caractères,
- une complexité minimale de 3 classes de caractères.

```
[password_quality]
policies = minimum-length character-class
min_length = 8
min_classes = 3
```

Il est également possible d'utiliser un programme externe. L'interface est simplissime :

- le nom du principal et le nouveau mot de passe sont transmis via STDIN sous la forme de couples clé:valeur sur des lignes distinctes.
- la chaîne 'APPROVED' sur STDOUT correspond à une réponse positive, toute autre chaîne correspond à la raison d'une réponse négative.

L'exemple suivant montre l'utilisation d'un script Perl, puis le script en question. Celui-ci commence par vérifier que le mot de passe n'est pas identique au principal, puis utilise les bindings Perl de Cracklib pour vérifier la présence éventuelle de mots issus d'un dictionnaire.

```
[password_quality]
policies = external-check
external_program = /usr/local/share/heimdal/check-cracklib.pl

#!/usr/bin/perl

use strict;
use Crypt::Cracklib;

my $database = '/usr/share/cracklib/pw_dict';

my %params;

sub check_basic {
    my $principal = shift;
    my $passwd = shift;

    if ($principal eq $passwd) {
        return "Principal name as password is not allowed";
    }
    return "ok";
}

sub badpassword {
    my $reason = shift;
    print "$reason\n";
    exit 1
}

while (<>){
    last if /end$/;
    if (/^([:]+):(.+)$/) {
        die "key value pair not correct: $_";
    }
    $params{$1} = $2;
}

die "missing principal" if !defined $params{'principal'};
die "missing password" if !defined $params{'new-password'};

my $reason;

$reason = check_basic($params{'principal'}, $params{'new-password'});
badpassword($reason) if $reason ne "ok";

$reason = fascist_check($params{'new-password'}, $database);
badpassword($reason) if $reason ne "ok";

print "APPROVED\n";
exit 0
```

3.2 Configuration DNS

La configuration de Kerberos, nous l'avons vu, se fait dans un fichier `/etc/krb5.conf`. Le déploiement et la mise à jour de ce fichier sur l'ensemble du parc pose un problème logistique et une solution de gestion de configuration comme Puppet ou Cfengine n'est pas toujours suffisante pour y remédier totalement.

Une solution fort pratique consiste à déporter une partie de cette configuration des postes de travail vers une source centralisée déjà configurée, à savoir le DNS. En effet, en utilisant des enregistrements TXT et SRV, on peut définir les éléments minimums nécessaires au fonctionnement de Kerberos, à savoir :

- le royaume par défaut pour ce domaine,
- les serveurs à utiliser pour ce royaume.

L'exemple suivant reprend la configuration utilisée jusqu'ici, dans un fichier de définition de zone au format utilisé par Bind. Le KDC est annoncé comme disponible via TCP ou UDP, tandis que les deux autres services ne sont disponibles que dans un seul protocole chacun. Ce n'est pas vraiment un choix, plutôt une limitation de l'implémentation. Pour les enregistrements SRV, les trois valeurs numériques correspondent respectivement à une priorité, un poids à utiliser en cas de priorité équivalente, puis au port à utiliser, suivi du nom de la machine.

```
$ORIGIN zarb.home.
; royaume par défaut
kerberos TXT "ZARB.HOME"
; KDC
kerberos_tcp SRV 0 0 88 tcheka
kerberos_udp SRV 0 0 88 tcheka
; serveur d'administration
kerberos-admin_tcp SRV 0 0 749 tcheka
; serveur de changement de mot de passe
_kpasswd_udp SRV 0 0 464 tcheka
```

Côté client, l'utilisation de ces enregistrements est conditionnée par les directives `dns_lookup_realm` et `dns_lookup_kdc`, dans le fichier de configuration. Par défaut, la première est fausse, mais la seconde vraie, ce qui permet d'utiliser `kinit` sans aucune configuration, à condition de préciser explicitement le principal complet, avec son royaume. Ce qui veut dire que la commande `kinit` sera insuffisante, puisque le royaume ne peut pas être déterminé, mais `kinit -p guillaume@ZARB.HOME` fonctionnera, puisque le royaume est cette fois-ci identifiable et qu'il est possible d'obtenir la liste des KDC pour celui-ci depuis le DNS. Il faut bien sûr respecter la convention liant le nom d'un royaume Kerberos avec le nom d'une zone DNS pour pouvoir déterminer correctement le serveur à interroger.

Bien entendu, si les valeurs recherchées sont déjà présentes dans le fichier de configuration, le DNS n'est pas consulté. Il n'y a donc pas à craindre de trafic réseau inutile et ceci permet éventuellement de surcharger localement une configuration générale en cas de besoin spécifique.

Attention cependant aux conséquences en termes de sécurité que représente l'utilisation d'un service dont l'identité n'est pas garantie. En effet, un attaquant qui serait en mesure d'usurper l'adresse IP du serveur DNS légitime aurait alors la main sur cette configuration. En l'absence d'un mécanisme de sécurité adéquat, comme DNSSEC par exemple, cette option est donc déconseillée dans un environnement sensible.

3.3 RéPLICATION DE SERVEURS

Comme pour tout autre service, il est nécessaire pour améliorer la disponibilité de Kerberos de ne pas dépendre d'un serveur unique et d'être capable de fonctionner même quand celui-ci défaillera pour une raison ou une autre. La configuration du royaume peut alors être modifiée pour définir deux KDC différents, de la façon suivante :

```
[realms]
ZARB_HOME = {
    kdc      = tcheka.zarb.home
    kdc      = loubianka.zarb.home
    admin_server = tcheka.zarb.home
    kpasswd_server = tcheka.zarb.home
}
```

Pour mettre en place le second KDC, il faut installer un serveur heimdal sur une deuxième machine et y transférer la clé de stockage créée lors de l'initialisation du royaume, dans le fichier **/var/lib/heimdal/m-key**. Il faut ensuite utiliser l'un des deux mécanismes suivants, pour assurer la synchronisation avec le KDC primaire.

3.3.1 RéPLICATION COMPLÈTE

La réPLICATION COMPLÈTE consiste à transférer l'intégralité de la base Kerberos du KDC maître au KDC esclave. Il s'agit d'un mécanisme de type *push*, c'est-à-dire que c'est le maître qui initie la connexion.

D'abord, il faut mettre en place le principal **hprop/machine@ROYAUME**, qui a dû être créé automatiquement à l'initialisation du royaume, dans le fichier keytab du serveur esclave, de façon à permettre l'authentification du serveur maître :

```
[root@loubianka ~]# kadmin -p guillaume
kadmin> add --random-key --use-defaults iprop/tcheka.zarb.home
guillaume@ZARB_HOME's Password:
```

Il faut ensuite lancer le daemon **hpropd** sur l'esclave, qui bloque en attendant de recevoir des données et lancer la commande **hprop** sur le maître :

```
[root@loubianka ~]# hpropd
...
[root@tcheka ~]# hprop loubianka
```

Un simple coup d'œil au répertoire **/var/lib/heimdal** permet de constater la présence d'un fichier **heimdal.db**, preuve du succès du transfert. Pour une vérification plus approfondie,

il est possible de lancer un serveur d'administration localement et de s'y connecter. Bien sûr, il faut également avoir transféré le fichier définissant les ACL :

```
[root@loubianka ~]# service heimdal start
Lancement de Heimdal Kerberos 5 Key Distribution Center: [ OK ]
Lancement de Heimdal administration server: [ OK ]
[root@loubianka ~]# kadmin -p guillaume --admin-server=localhost
kadmin> get guillaume
guillaume@ZARB_HOME's Password:
Principal: guillaume@ZARB_HOME
Principal expires: never
Password expires: never
Last password change: 2018-04-06 20:25:11 UTC
Max ticket life: 1 day
Max renewable life: 1 week
Kvno: 1
Mkvno: unknown
Last successful login: never
Last failed login: never
Failed login count: 0
Last modified: 2018-04-26 20:58:00 UTC
Modifier: guillaume@ZARB_HOME
Attributes:
Keytypes: aes256-cts-hmac-sha1-96(pw-salt), des3-cbc-sha1(pw-salt),
arcfour-hmac-md5(pw-salt)
PX-INIT ACL:
Aliases:
```

L'automatisation de cette procédure consiste à lancer automatiquement **hpropd** sur l'esclave à chaque connexion, via **inetd** ou **xinetd**, et à utiliser une tâche **cron** sur le maître pour lancer **hprop** à intervalles réguliers.

3.3.2 RéPLICATION INCRÉMENTALE

L'autre mécanisme consiste à transférer immédiatement les changements se produisant sur le serveur maître vers le serveur esclave. Les synchronisations sont donc plus rapides, mais légèrement plus complexes à mettre en œuvre, puisqu'il faut garder un état de part et d'autre.

Il faut d'abord mettre en place dans le fichier keytab de chaque serveur un principal de la forme **iprop/machine@ROYAUME** :

```
[root@tcheka ~]# kadmin -p guillaume
kadmin> add --random-key --use-defaults iprop/tcheka.zarb.home
guillaume@ZARB_HOME's Password:
kadmin> ext_keytab iprop/tcheka.zarb.home
[root@loubianka ~]# kadmin -p guillaume
kadmin> add --random-key --use-defaults iprop/loubianka.zarb.home
guillaume@ZARB_HOME's Password:
kadmin> ext_keytab iprop/loubianka.zarb.home
```

L'étape suivante consiste à configurer la synchronisation sur le maître, dans le fichier **/var/lib/heimdal/slaves**. Ce fichier comporte une ligne par serveur esclave, contenant le nom du principal correspondant à celui-ci, comme créé à l'étape précédente.

Enfin, il faut lancer de part et d'autre les processus nécessaires, à savoir **ipropd-master** sur le maître et **ipropd-slave**

sur l'esclave. Sur une distribution Mandriva, ceci se fait en renseignant les variables adéquates dans le fichier `/etc/sysconfig/heimdal`, puis en relançant le service `heimdal`.

Sur le maître, le fichier de configuration doit ressembler à ceci :

```
...  
# Set to yes to have ipropd-master started  
START_MASTER=yes  
MASTER_ARGS="--detach"  
...
```

Et sur l'esclave, il doit ressembler à cela :

```
...  
# Set to yes to start ipropd-slave, set MASTER to hostname of the  
ipropd-master  
START_SLAVE=yes  
SLAVE_ARGS="--detach"  
MASTER="tcheka.zarb.home"  
...
```

Si tout est configuré correctement, les logs de l'esclave devraient comporter des messages de ce type :

```
Aug 15 18:02:20 loubianka ipropd-slave[4830]: connexion successful to master:  
tcheka.zarb.home[192.168.0.1]  
Aug 15 18:02:20 loubianka ipropd-slave[4830]: ipropd-slave started at version: 9  
Aug 15 18:02:20 loubianka ipropd-slave[4830]: receive complete database  
Aug 15 18:02:20 loubianka ipropd-slave[4830]: receive complete database, version 159
```

Le numéro de révision de la base, 159, peut être retrouvé sur le maître grâce à la commande `iprop-log` :

```
[root@tcheka ~]# iprop-log dump  
...  
create: ver = 157, timestamp = 2010-08-15 17:52:47, len = 682  
principal = iprop/loubianka.zarb.home@ZARB.HOME  
expires = never  
password exp = never  
attributes = invalid, client, server, postdate, renewable,  
proxiable, forwardable  
kvno = 1  
[]  
modify: ver = 158, timestamp = 2010-08-15 17:52:47, len = 566  
principal = iprop/loubianka.zarb.home@ZARB.HOME  
password exp = never  
mod time  
mod name  
kvno = 2  
key data  
tl data  
modify: ver = 159, timestamp = 2010-08-15 17:52:47, len = 566  
principal = iprop/loubianka.zarb.home@ZARB.HOME  
attributes = client, server, postdate, renewable, proxiable,  
forwardable  
mod time  
mod name  
kvno = 1
```

Cette commande permet de s'apercevoir que la création d'un principal s'effectue en fait en plusieurs modifications distinctes de la base. Et pour éviter une éventuelle attaque de type *race condition* entre la création et l'affectation du mot de passe, le principal n'est désactivé qu'à la dernière étape :

- le principal est créé avec un attribut **invalid**,
- un mot de passe aléatoire lui est affecté,
- l'attribut **invalid** est supprimé.

3.4 Gestion de la cohérence entre bases

Comme rappelé dans la section concernant PAM (voir partie précédente), Kerberos n'assure qu'une fonction d'authentification et nécessite donc d'être couplé avec une base de comptes pour pouvoir gérer des comptes utilisateurs POSIX. Quelle que soit la forme que prenne cette base (fichier `/etc/passwd`, NIS, LDAP), le problème consiste à assurer la cohérence de ces deux bases : chaque utilisateur doit en effet posséder une entrée dans la base Kerberos et une autre dans la base de comptes.

Le problème se complique encore plus en ce qui concerne la synchronisation des mots de passe, puisque les bases de comptes utilisateurs sont conçues pour être auto-suffisantes et comprennent également un mot de passe pour l'utilisateur. Il faut donc s'assurer que l'utilisateur ne soit pas confronté à des problèmes de mot de passe différent selon que l'un ou l'autre système soit utilisé pour l'authentifier.

Il existe plusieurs stratégies possibles pour gérer ces deux problématiques. Pour la synchronisation de la liste des éléments, il est possible par exemple, de créer automatiquement un principal Kerberos pour chaque utilisateur présent dans la base des comptes et supprimer ceux ne correspondant pas à un utilisateur. Pour la synchronisation des mots de passe, il est possible d'utiliser un point central, comme un script CGI par exemple, qui propage les nouveaux mots de passe vers les deux bases. Ou alors de s'assurer que les mots de passe de la base de comptes ne soient jamais utilisés, mais ceci reste assez utopique dans un environnement hétérogène.

Lorsque la base de comptes utilisateurs est stockée dans un annuaire LDAP, une autre stratégie permet de traiter le premier, voire le second problème. Cette stratégie repose sur trois étapes successives :

- stocker la base Kerberos dans l'annuaire LDAP,
- fusionner les enregistrements des deux bases,
- utiliser les attributs Kerberos pour l'authentification.

3.4.1 Stockage LDAP

L'utilisation d'un annuaire LDAP comme solution de stockage, à la place d'un fichier bdb classique, est possible grâce au *backend* HDB d'Heimdal. Celui-ci possède à l'heure actuelle une limitation assez forte, à savoir qu'il est capable de communiquer avec le serveur LDAP uniquement à travers une socket Unix locale, et pas par TCP/IP. En conséquence, les deux serveurs doivent être hébergés sur la même machine. Cette limitation devrait néanmoins disparaître dans une version ultérieure.

Du côté Kerberos, ceci se configure dans la section **kdc** du fichier de configuration :

```
[kdc]
database = {
    zarb.home = {
        dbname = ldap:ou=kerberos,dc=zarb,dc=home
        realm = ZARB.HOME
        acl_file = /var/lib/heimdal/kadmind.acl
        mkey_file = /var/lib/heimdal/m-key
    }
}
```

Du côté LDAP, il faut d'abord s'assurer que le serveur écoute sur une socket Unix, en passant l'argument **ldapi:///** au lancement du serveur. Typiquement, ceci se fait en renseignant la variable **SLAPDURLLIST** dans le fichier **/etc/sysconfig/ldap**.

Ensuite, il faut configurer le serveur, soit dans son fichier de configuration **/etc/openldap/slapd.conf**, soit dans la branche **cn=config** si cette fonctionnalité est utilisée. Il faut d'abord charger le schéma spécifique à Heimdal. Ensuite, il faut configurer l'accès via la socket, qui est géré par SASL et notamment transposer l'identité sous laquelle se présentera HDB en un identifiant LDAP. Ensuite, il faut définir les ACL permettant de donner les pleins pouvoirs à HDB sur sa branche, tout en restreignant les accès en lecture à l'attribut **krb5Key**, correspondant aux clés des principaux. Enfin, il faut mettre en place les index nécessaires. Au final, ces diverses modifications ressemblent à ceci :

```
# schema
include /usr/share/openldap/schema/krb5-kdc.schema

# local socket access
sasl-secprops minssf=0

authz-regexp "gidNumber=0\|uidNumber=0,cn=peercred,cn=external,cn=auth"
              "cn=heimdal,dc=zarb,dc=home"

# ACLs
access to dn.subtree="ou=kerberos,dc=zarb,dc=home"
        by dn.exact="cn=heimdal,dc=zarb,dc=home" write
        by * break

access to attrs=krb5Key
        by self read
        by * none

# Indexes
index krb5PrincipalName eq
```

Enfin, il faut également s'assurer que le processus **slapd** peut accéder à la clé de stockage de la base Kerberos et donc que les droits sur le fichier **/var/lib/heimdal/m-key** lui permettent de le lire.

```
[root@tcheka ~]# chgrp slapd /var/lib/heimdal/m-key
[root@tcheka ~]# chmod 640 /var/lib/heimdal/m-key
```

Une fois cette configuration en place, il suffit d'initialiser le royaume, comme vu précédemment. S'il s'agit de convertir le format de stockage d'un royaume déjà existant, il est possible

d'utiliser une des techniques de synchronisation vues précédemment depuis un autre serveur, ou plus simplement d'exporter la base (**kadmin -l dump**), d'effectuer les modifications, puis de réimporter la base (**kadmin -l load**).

D'une façon générale, ce changement de mode de stockage ne change absolument rien à l'utilisation du royaume. Bien que nécessaire à la mise en place de notre stratégie de gestion de la cohésion, il ne s'agit donc pas d'une étape suffisante. Elle apporte cependant l'avantage de consolider d'autres données relatives aux utilisateurs dans une infrastructure unique, possédant déjà ses propres procédures de synchronisation, de sauvegarde et de restauration. La réplication entre annuaires remplace ainsi avantageusement les mécanismes natifs de Kerberos pour synchroniser les données entre KDC.

3.4.2 Fusion des bases

À l'issue de l'étape précédente, le contenu de l'annuaire LDAP est réparti entre plusieurs branches. Les objets correspondant aux comptes POSIX sont dans la branche **ou=users,dc=zarb,dc=home**, tandis que ceux correspondant aux principaux Kerberos sont dans la branche **ou=kerberos,dc=zarb,dc=home** (figure 4).

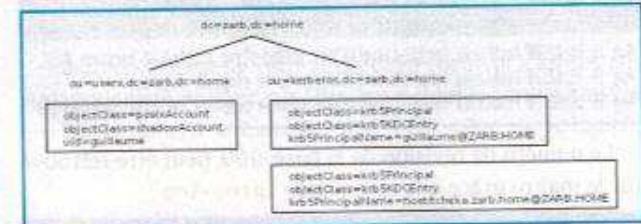


Figure 4 : Données utilisateur et Kerberos séparées

A un utilisateur correspondent donc deux objets distincts, comme le montre la requête suivante :

```
[root@tcheka ~]# ldapsearch -x uid=guillaume -LL
version: 1

dn: uid=guillaume,ou=users,dc=zarb,dc=home
homeDirectory: /home/users/guillaume
cn: Guillaume Rousse
uid: guillaume
uidNumber: 500
gidNumber: 500
loginShell: /bin/bash
objectClass: account
objectClass: posixAccount
objectClass: shadowAccount
objectClass: top

dn: krb5PrincipalName=guillaume@ZARB.HOME,ou=kerberos,dc=zarb,dc=home
objectClass: top
objectClass: account
objectClass: krb5Principal
objectClass: krb5KDCEntry
krb5PrincipalName: guillaume@ZARB.HOME
uid: guillaume
krb5KeyVersionNumber: 1
krb5MaxLife: 86400
krb5MaxRenew: 604800
krb5KDCFlags: 126
```

Or, les classes de ces deux objets sont compatibles, il est donc possible de les fusionner en un seul. On aboutit alors à ceci :

```
[root@tcheka ~]# ldapsearch -x uid=guillaume -LL
version: 1

dn: uid=guillaume,ou=users,dc=zarb,dc=home
homeDirectory: /home/users/guillaume
cn: Guillaume Rousse
uid: guillaume
uidNumber: 500
gidNumber: 500
loginShell: /bin/bash
krb5KDCFlags: 126
krb5KeyVersionNumber: 1
krb5MaxLife: 86400
krb5MaxRenew: 604800
krb5PrincipalName: guillaume@ZARB.HOME
objectClass: account
objectClass: krb5KDCEntry
objectClass: krb5Principal
objectClass: posixAccount
objectClass: shadowAccount
objectClass: top
```

Une vérification rapide montre que l'objet LDAP n'est plus visible par le KDC :

```
[root@tcheka ~]# kadmin -p guillaume
kadmin> get guillaume
guillaume@ZARB.HOME's Password:
kadmin: get guillaume: Client not found in Kerberos database
```

En effet, le serveur est configuré pour rechercher ses données dans la branche **ou=kerberos,dc=zarb,dc=home** uniquement, alors que les principaux se retrouvent maintenant répartis entre cette branche, pour ceux correspondant à des services, et dans la branche **ou=users,dc=zarb,dc=home**, pour ceux correspondant à des utilisateurs. Il faut donc modifier la configuration du KDC pour modifier la base de recherche et la décaler à la racine, **dc=zarb,dc=home**.

```
[kdc]
database = {
    zarb.home = {
        dbname = ldap:dc=zarb,dc=home
        realm = ZARB.HOME
        acl_file = /var/lib/heimdal/kadmind.acl
        mkey_file = /var/lib/heimdal/m-key
    }
}
```

Ce changement est à répercuter également au niveau des ACL du serveur LDAP :

```
access to dn.subtree="dc=zarb,dc=home"
  by dn.exact="cn=heimdal,dc=zarb,dc=home" write
  by * break
```

Après rechargement des deux serveurs, la vérification montre que la configuration est pleinement fonctionnelle :

```
[root@tcheka ~]# kadmin -p guillaume
kadmin> get guillaume
guillaume@ZARB.HOME's Password:
```

```
Principal: guillaume@ZARB.HOME
Principal expires: never
Password expires: never
Last password change: never
Max ticket life: 1 day
Max renewable life: 1 week
Kvno: 1
Mkvno: unknown
Last successful login: never
Last failed login: never
Failed login count: 0
Last modified: 2010-08-15 19:54:48 UTC
Modifier: unknown
Attributes:
  Keytypes: aes256-cts-hmac-sha1-96(pw-salt), des3-cbc-
            sha1(pw-salt),
            arcfour-hmac-md5(pw-salt)
PK-INIT ACL:
Aliases:
```

À l'issue de cette étape, les principaux correspondant à des utilisateurs sont fusionnés avec les comptes utilisateurs dans la branche **ou=users,dc=zarb,dc=home**, tandis que ceux correspondant à des services sont restés dans la branche **ou=kerberos,dc=zarb,dc=home** (figure 5).

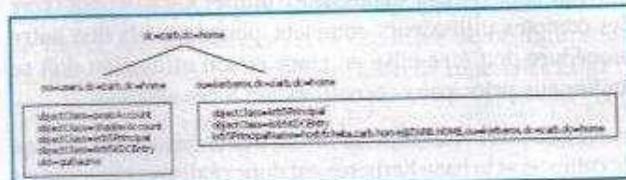


Figure 5 : Données utilisateur et Kerberos réunies

Attention, le serveur Kerberos n'a aucune connaissance de cette cohabitation des données au sein de l'annuaire et agit comme s'il s'agissait des siennes uniquement. Autrement dit, lorsque **kadmin** est utilisé pour créer des principaux, il crée des objets LDAP appartenant aux classes **krb5Principal** et **krb5KDCEntry**, pas à l'ensemble des classes assurant le double rôle de principal Kerberos et de compte POSIX. De plus, il les place là où il est configuré pour effectuer ses recherches, donc à la base de l'annuaire. Pire encore, s'il est utilisé pour détruire un principal, il va détruire le compte utilisateur associé, puisqu'il s'agit du même objet...

Il est possible d'améliorer un peu la configuration pour remédier à ces problèmes. D'abord, il est possible de spécifier une base pour la création d'objets différente de la base de recherche, avec la directive **hdb-ldap-create-base** :

```
[kdc]
database = {
    zarb.home = {
        dbname = ldap:dc=zarb,dc=home
        realm = ZARB.HOME
        acl_file = /var/lib/heimdal/kadmind.acl
        mkey_file = /var/lib/heimdal/m-key
    }
}
hdb-ldap-create-base = ou=kerberos,dc=zarb,dc=home
```

Ensuite, pour empêcher la destruction accidentelle de comptes utilisateurs via **kadmin**, il est possible de jouer sur les ACL, pour limiter les permissions associées à l'entité **cn=heimdal,dc=zarb,dc=home** dans la branche des utilisateurs à la modification des seuls attributs relatifs à Kerberos, en utilisant la syntaxe **attrs=@class**, qui évitera l'énumération explicite des attributs d'une classe :

```
# heimdal has total control in kerberos branch
access to dn.subtree="ou=kerberos,dc=zarb,dc=home"
  by dn.exact="cn=heimdal,dc=zarb,dc=home" write
  by * break

# heimdal has write control on kerberos attributes
# in user branch
access to dn.subtree="ou=users,dc=zarb,dc=home"
  attrs=@Krb5DCEEntry
  by dn.exact="cn=heimdal,dc=zarb,dc=home" write
  by * break

access to dn.subtree="ou=users,dc=zarb,dc=home"
  attrs=@Krb5Principal
  by dn.exact="cn=heimdal,dc=zarb,dc=home" write
  by * break
```

Néanmoins, il reste impossible d'utiliser **kadmin** pour créer des comptes utilisateurs complets, pour lesquels une autre procédure doit être mise en place et son utilisation doit se limiter aux principaux correspondant à des services.

À l'issue de cette étape, la fusion des entrées entre la base de comptes et la base Kerberos est donc réalisée et les risques d'incohérence entre les deux largement réduits.

3.4.3 Utilisation des attributs Kerberos pour l'authentification

Il reste néanmoins un problème de synchronisation des attributs d'authentification. En effet, l'objet correspondant à l'utilisateur comprend toujours un attribut **userPassword**, appartenant à la classe **account**, et l'attribut **krb5Key**, appartenant à la classe **Krb5Principal**. La requête authentifier suivante montre ces attributs. Les valeurs différentes pour l'attribut **krb5Key** correspondent aux différentes clés, chacune d'entre elles correspondant à un algorithme de chiffrement différent :

```
[guillaume@beria ~]$ ldapsearch -x -D uid=guillaume,ou=users,dc=zarb,dc=home
-W -LL uid=guillaume krb5Key userPassword krb5KeyVersionNumber
Enter LDAP Password:
version: 1

dn: uid=guillaume,ou=users,dc=zarb,dc=home
krb5Key:: Mz2gAwIBAaFHMENgAwIBEqE+BDyurHeQyG1Kh/P7/m0ChVvbdHk0gTWRr3Hc01Bkt1b9
8tUtkGm6742C0xb3svG7/rhrd21xd4zK4wzBeemHTAb0AMCAQ0hFA0SWkFSQ15IT01FZ3VpbGxd
W1
krb5Key:: MG2gAwIBAaE/M02gAwIBENE28DS32EM0t0gfuJr8NB0Cxujhbed9s8xqg8Z83a0KC8
j2+6jykyCcAr0lJ46rnH+NrnsSe0hWGRADAgEDoR0EEt080kiuSE9NRWdtaikstXVt20==
krb5Key:: Mf2gAwIBAaE3MDNgAwIBF6E8Cw1LXmgyp7Ac4c/ZTYBwZiygAhkrnVi9AP5p0f+gF
XhdAII18jb//g09KgIdM0ugAwIBA6EU88JaQVJCLkhPTUVndW1sbGf1WU=
krb5KeyVersionNumber: 1
userPassword:: eINTSEF9N2duMet30WF0Y1mFc8nUkjZk1LSm1tWmdvbVVnDm=
```

Dans cette situation, une authentification par l'annuaire LDAP, via une opération de type bind, va utiliser l'attribut

userPassword, tandis qu'une authentification par le KDC, via une requête de type AS-REQ, va utiliser l'attribut **krb5Key**. En l'absence de mécanisme de synchronisation, ces deux attributs peuvent très bien correspondre à des valeurs différentes. À moins de pouvoir garantir qu'un utilisateur ne sera jamais confronté qu'à un seul de ces mécanismes, ou alors qu'il saura toujours identifier celui auquel il est confronté, c'est la porte ouverte à un certain nombre de confusions...

OpenLDAP propose une solution, grâce au greffon (overlay, dans la terminologie OpenLDAP) **smbk5pwd**. Ce greffon permet en fait de synchroniser les différents attributs servant à l'authentification pour Samba, Kerberos et LDAP lorsqu'une opération étendue de changement de mots de passe (**passwd** **exop**) est effectuée sur le serveur LDAP. Ceci concerne à la fois les attributs correspondant aux mots de passe eux-mêmes (**sambaLMPassword**, **sambaNTPassword**, **krb5Key** et **userPassword**), mais également les attributs associés (**sambaPwdLastSet**, **krb5KeyVersionNumber**, etc...).

L'installation de ce greffon peut être légèrement délicate, puisqu'il ne fait pas partie des extensions standards d'OpenLDAP et qu'il faut compiler manuellement les sources contenues dans le répertoire **contrib/slapd-modules/smbk5pwd**. Sur Mandriva, il est disponible dans le paquetage **openldap-smbk5pwd**, distinct du paquetage **openldap** pour des raisons de dépendances. Une fois installé, cependant, il suffit de charger le fichier du greffon dans la configuration globale du serveur LDAP, puis l'activer dans la configuration de la base souhaitée :

```
moduleload smbk5pwd.so
...
database bdb
suffix "dc=zarb,dc=home"
...
overlay smbk5pwd
```

Un simple changement de mot de passe via la commande **ldappassword** permet de constater la synchronisation des attributs :

```
[guillaume@beria ~]$ ldapsearch -x -D uid=guillaume,ou=users,dc=zarb,dc=home
-W -LL uid=guillaume
Enter LDAP Password:
version: 1

dn: uid=guillaume,ou=users,dc=zarb,dc=home
userPassword:: eINTSEF9N2duMet30WF0Y1mFc8nUkjZk1LSm1tWmdvbVVnDm=
krb5KeyVersionNumber: 2
krb5Key:: Mf2gAwIBAaE/M02gAwIBENE28DS32EM0t0gfuJr8NB0Cxujhbed9s8xqg8Z83a0KC8
j2+6jykyCcAr0lJ46rnH+NrnsSe0hWGRADAgEDoR0EEt080kiuSE9NRWdtaikstXVt20==
krb5Key:: Mf2gAwIBAaE3MDNgAwIBF6E8Cw1LXmgyp7Ac4c/ZTYBwZiygAhkrnVi9AP5p0f+gF
XhdAII18jb//g09KgIdM0ugAwIBA6EU88JaQVJCLkhPTUVndW1sbGf1WU=
krb5KeyVersionNumber: 1
userPassword:: eINTSEF9N2duMet30WF0Y1mFc8nUkjZk1LSm1tWmdvbVVnDm=
```

L'examen des valeurs des attributs **userPassword** et **krb5Key** montre que ceux-ci ont été modifiés, mais la preuve la plus flagrante est l'incrémentation de l'attribut

krb5KeyVersionNumber, dont la valeur est passée à 2. L'examen via **kadmin** montre également qu'il y a eu modification, via ce même numéro de version et la date de modification :

```
[guillaume@beria ~]$ /usr/sbin/kadmin -p guillaume
kadmin> get guillaume
guillaume@ZARB.HOME's Password:
        Principal: guillaume@ZARB.HOME
        Principal expires: never
        Password expires: never
        last password change: never
        Max ticket life: 1 day
        Max renewable life: 1 week
            Krb5cc: 2
            Mktvno: unknown
        Last successful login: never
        Last failed login: never
        Failed login count: 0
        Last modified: 2010-08-15 22:21:53 UTC
        Modifier: guillaume@ZARB.HOME
        Attributes:
            Keytypes: des-cbc-crc(pw-salt), aes256-cts-hmac-sha1-96(pw-salt),
                      des3-cbc-sha1(pw-salt), arcfour-hmac-md5(pw-salt)
        PKINIT ACL:
        Aliases:
```

Néanmoins, cette synchronisation n'est possible que si le changement de mot de passe se fait via l'opération spécifique (**password exop**), puisqu'il faut disposer du mot de passe en clair pour le chiffrer de différentes manières. Si le mot de passe est changé via une opération classique de modification d'attribut (mod), cette synchronisation n'aura pas lieu. Et malheureusement, c'est cette opération qui est utilisée dans le *backend* HDB, comme il est facile de s'en rendre compte :

```
[guillaume@beria -]$: /usr/sbin/kadmin -p guillaume
kadmin> cpw guillaume
guillaume@ZARD_HOME's Password:
guillaume@ZARD_HOME's Password:
Verifying - guillaume@ZARD_HOME's Password:
kadmin> get guillaume
Principal: guillaume@ZARD_HOME
Principal expires: never
Password expires: never
Last password change: never
    Max ticket life: 1 day
    Max renewable life: 1 week
        Kvno: 3
        Mkvno: unknown
Last successful login: never
    Last failed login: never
    Failed login count: 0
        Last modified: 2010-08-16 06:27:24 UTC
        Modifier: unknown
Attributes:
    Keytypes: des-cbc-crc(pw-salt), aes256-cts-hmac-sha1-96(pw-salt),
              des3-cbc-sha1(pw-salt), arcfour-hmac-md5(pw-salt)
PKINIT ACL:
Aliases:
```

Une fois encore, l'incrémentation du numéro de version et la date de modification montrent le changement, côté Kerberos. Par contre, côté LDAP, même si c'est plus difficile à mettre en évidence, l'attribut **userPassword** n'a pas été modifié :

```
[guillaume@beria ~]# ldapsearch -x -D uid=guillaume,ou=users,dc=zarb,dc=home  
-W -H ldap://guillaume krb5Key userPassword krb5KeyVersionNumber
```

```
Enter LDAP Password:  
version: 1  
  
dn: uid=guillaume,ou=users,dc=zerf,dc=home  
userPassword:: e1NTSEF9W05jd2R025gMnxrhjNKSjd0MW71YVFrTzdrZXMQZE=  
krb5KeyVersionNumber: 3  
krb5Key:: MNg0AwIBAeEWkCz2gAwIBaEAm8CSQ1cpVnS/ZLbhNx0FVkv7m4J0k4r040oujaxSP36  
88Tyka5iHtAb0uNCAD0RFAQSNFKSQ151tB1F23Vpb6xhdW1  
krb5Key:: MGZgAwIBaE/HfMENgAwIBEq+@0yy0lN2oh4kj073pIyN8E0Nz+C+vLRf8AY6j10KyF  
HieqGPVNWb5m5slystyneC17ufP9gMvCtgA5vKtHtAb0uNCAD0RFAQSNFKSQ151tB1F23Vpb6xhd  
W1  
krb5Key:: MGNgAwIBaE/MD2gMwIBxEKE2B00o@he7iVw/KjAn51Yi2cZw6H1S8Cx0ZgBnFBZj  
j8885P/BC6V1u1xzw2R4097J42LoohEw64DaBgE9nRQEETj0BuKtS29Rwd1akxsXXVjZ0=  
krb5Key:: MF7gAwIBaE3X0BwMw189f6uBcyJgkWz7/YSW80gA51Hw1Nny7a++/V4S5Th7P+8CM  
RPhhFD55sivPgfCQ0KtDw8sAwIBAeEBBjaQJCLkrP7UvndWtSbsG1bM0=
```

Pour faire face à ce problème, le greffon **smbkb5pwd** propose une autre solution, à savoir l'utilisation d'une valeur spécifique pour l'attribut **userPassword**, **{K5KEY}**, assurant l'utilisation des clés Kerberos pour l'authentification par l'annuaire lors des opérations de type bind. Il n'y a donc plus de duplication de valeur et donc plus de synchronisation à effectuer. Pour mettre en place automatiquement cette valeur particulière et éviter qu'elle ne soit remplacée par une empreinte classique du nouveau mot de passe au prochain changement, il suffit de forcer son utilisation via la directive de configuration **password-hash** dans la configuration de l'annuaire LDAP :

```
# enforce kerberos credentials usage when changing password  
password-krb {K5KEY}
```

Il suffit alors de changer ce mot de passe, puis d'interroger l'annuaire, pour constater que l'attribut ***userPassword*** possède bien la valeur souhaitée, encodée en base 64 à cause des accolades :

```
[guillaume@beria ~]$ ldappasswd -x -D uid=guillaume,ou=users,dc=zarb,dc=home  
-W uid=guillaume,ou=users,dc=zarb,dc=home -s foo  
Enter LDAP Password:  
[guillaume@beria ~]$ ldapsearch -x -D uid=guillaume,ou=users,dc=zarb,dc=home  
-W -L uid=guillaume krb5Key userPassword krb5KeyVersionNumber  
Enter LDAP Password:  
version: 1  
  
dn: uid=guillaume,ou=users,dc=zarb,dc=home  
userPassword:: e8sIS8VzI0==  
krb5KeyVersionNumber: 4  
krb5Key:: MfNgAwIBAeEwHc2AwIBaaMwC8R5o196AM036gvb1nLT73gRZ70s19PCydy9k0jxlnh  
VWLLUgt1HTAb0AMCAQ0hF4QSWkFSQ15T01F23vpbGxhW1!  
krb5Key:: M0ZGAvIBAfFHMVg9h18Ecjt+80zgeExnHt2XKfFwPx+620qf3r8grL7v5Jrap7t8Wjt  
97X0LezYzdK5m0allowPfp2gxwule8l1vFKh01HTAb0AMCAQ0hF4QSWkFSQ15T01F23vpbGxh  
W1!  
krb5Key:: MGNgAwIBAeE/M0ZGAvIBWeKE2B0TtrjbvGbMkiYzI247WXKd3jV5h247f5j31Xy641  
jx7ihmFQ2zzuy5j/fL+ErJ5FB5johWg6AD4gEDo80EEjt89hJwSE9NWRw1ak0s3XV6Z0=  
krb5Key:: MfNgAwIBAf3MDNgwIBF6eubC4ycaNKH+MRF149823X08pSjVrRzG9Kd+AkMw805tg1  
aTAj2TeibLhkq0B4ka1nd8ugAwIBA6EiB8Bja0YQJ0LktPtuVwIw1sbGf1bmU=  
(guillaume@beria ~$ perl -M MIME::Base64 -E 'say decode_base64("e8sIS8VzI0=")  
'KSKEY1
```

A l'issue de cette dernière étape, la stratégie est donc en place et assure à la fois la cohérence des comptes utilisateurs avec leurs principaux Kerberos, ainsi que l'utilisation d'un mot de passe unique quel que soit le mode d'authentification (LDAP ou Kerberos) utilisé. ■

KERBEROS,

LE SSO UNIVERSEL

4- RELATION DE CONFIANCE ENTRE ROYAUMES

par Guillaume Rousse

Jusqu'ici, les exemples présentés concernaient l'accès à des ressources appartenant au même royaume Kerberos que l'utilisateur. Or, ceci n'est pas toujours le cas dans la réalité, lorsqu'il existe plusieurs royaumes Kerberos différents au sein d'une même entité. Cette situation peut être volontaire, les différents royaumes jouant une fonction de cloisonnement, ou non, typiquement lorsqu'un domaine Active Directory est mis en place pour gérer un parc de machines Windows et qu'il vient avec son propre royaume Kerberos.

Comme vu précédemment, le principe de base de l'authentification Kerberos consiste à présenter un ticket d'accès à une ressource. Ce ticket s'obtient auprès du KDC du royaume Kerberos auquel appartient cette ressource, à condition de pouvoir s'y authentifier. Le mécanisme de SSO consiste à s'authentifier une fois auprès du service d'authentification (AS) via un secret partagé entre l'utilisateur et le KDC, le mot de passe, lors du dialogue AS-REQ/AS-REP, puis à s'authentifier ensuite autant de fois que nécessaire auprès du service de tickets (TS) via le ticket récupéré à l'étape précédente, lors du dialogue AS-REQ/AS-REP (figure 1).

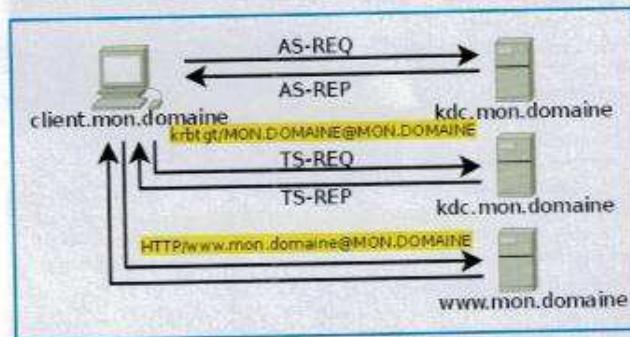


Figure 1 : Accès à une ressource du royaume de l'utilisateur

Dans le cas où cette ressource appartient à un autre royaume, il n'y a plus de secret partagé entre l'utilisateur et le KDC susceptible de délivrer le ticket souhaité et donc, aucune possibilité d'initier le mécanisme (figure 2).

Pour sortir de cette impasse, Kerberos propose de mettre en place une relation de confiance entre les royaumes. Concrètement, cette relation se matérialise par la mise en place

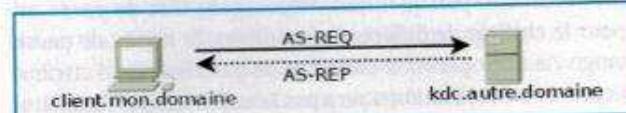


Figure 2 : Accès à une ressource d'un autre royaume d'un secret partagé entre les deux KDC, sous la forme d'un principal unique, pour une relation unidirectionnelle, ou d'un couple de principaux, pour une relation mutuelle, présent dans les deux royaumes avec des clés identiques. Ces principaux sont de la forme `krbtgt/DOMAINE1@DOMAINE2` (figure 3).

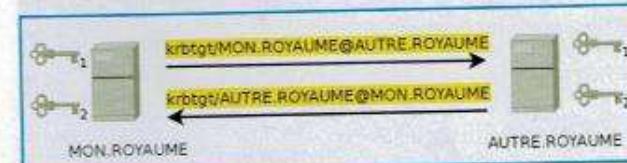


Figure 3 : Relation de confiance

Une fois cette relation en place, l'utilisateur peut donc obtenir un ticket pour le principal de la relation de confiance auprès de son propre KDC et s'en servir ensuite pour s'authentifier auprès de l'autre KDC pour obtenir le ticket souhaité, au prix d'un échange TGS-REQ/TGS-REP supplémentaire (figure 4).

1 Confiance entre royaumes Unix

La mise en place d'une relation de confiance entre royaumes Unix est relativement simple. Il suffit en effet de créer les deux principaux dans chacun des royaumes, en veillant bien sûr à utiliser le même mot de passe.

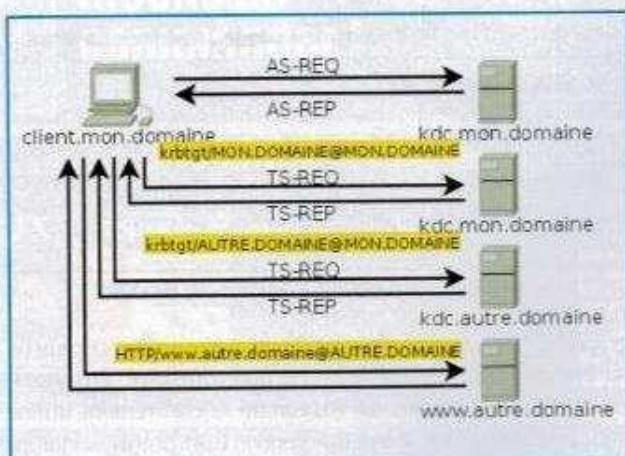


Figure 4 : Accès à une ressource d'un autre royaume, avec relation de confiance

Dans notre cas, il faut ajouter une 4^eme machine **kolyma.unix.zarb.home**, avec un serveur Heimdal et un royaume Kerberos **UNIX.ZARB.HOME** fonctionnels. Une fois ceci fait, les principaux peuvent être créés sur les deux serveurs :

```
[root@tcheka ~]# kadmin -l
kadmin> add --use-defaults krbtgt/UNIX.ZARB.HOME@ZARB.HOME
krbtgt/UNIX.ZARB.HOME@ZARB.HOME's Password:
Verifying - krbtgt/UNIX.ZARB.HOME@ZARB.HOME's Password:
kadmin> add --use-defaults krbtgt/ZARB.HOME@UNIX.ZARB.HOME
krbtgt/ZARB.HOME@UNIX.ZARB.HOME's Password:
Verifying - krbtgt/ZARB.HOME@UNIX.ZARB.HOME's Password:
kadmin> exit
...
[root@kolyma ~]# kadmin -l
kadmin> add --use-defaults krbtgt/UNIX.ZARB.HOME@ZARB.HOME
krbtgt/UNIX.ZARB.HOME@ZARB.HOME's Password:
Verifying - krbtgt/UNIX.ZARB.HOME@ZARB.HOME's Password:
kadmin> add --use-defaults krbtgt/ZARB.HOME@UNIX.ZARB.HOME
krbtgt/ZARB.HOME@UNIX.ZARB.HOME's Password:
Verifying - krbtgt/ZARB.HOME@UNIX.ZARB.HOME's Password:
kadmin> exit
```

A moins d'utiliser les enregistrements DNS pour la configuration, il faut également ajouter le nouveau royaume dans le fichier de configuration des bibliothèques Kerberos. En effet, toute l'intelligence est située sur le client et c'est lui qui détermine le royaume de la ressource à laquelle il cherche à accéder et donc, le KDC auquel il doit s'adresser. C'est le rôle de la section **domain_realm**, qui établit une correspondance entre des noms de domaine ou de machines et un royaume Kerberos précis.

La configuration précédente devient donc :

```
[libdefaults]
default_realm = ZARB.HOME

[realms]
ZARB.HOME = {
    kdc      = tcheka.zarb.home
    admin_server = tcheka.zarb.home}
```

```
kpasswd_server = tcheka.zarb.home
}
UNIX.ZARB.HOME = {
    kdc      = kolyma.unix.zarb.home
    admin_server = kolyma.unix.zarb.home
    kpasswd_server = kolyma.unix.zarb.home
}

[domain_realm]
.zarb.home = ZARB.HOME
.unix.zarb.home = UNIX.ZARB.HOME
```

Une fois cette configuration en place, il est possible de tenter l'accès à un service situé dans l'autre royaume :

```
[guillaume@beria ~]$ kinit
guillaume@ZARB.HOME's Password:
[guillaume@beria ~]$ klist
Credentials cache: FILE:/tmp/krb5cc_500
Principal: guillaume@ZARB.HOME

  Issued          Expires        Principal
Aug 17 23:20:34 Aug 18 09:20:34 krbtgt/ZARB.HOME@ZARB.HOME
[guillaume@beria ~]$ ldapsearch -Y GSSAPI -L -h kolyma.unix.zarb.home
SASL/GSSAPI authentication started
SASL username: guillaume@ZARB.HOME
SASL SSF: 56
SASL data security layer installed.
version: 1

dn: dc=zarb,dc=home
dc: zarb
objectClass: top
objectClass: domain

[guillaume@beria ~]$ klist
Credentials cache: FILE:/tmp/krb5cc_500
Principal: guillaume@ZARB.HOME

  Issued          Expires        Principal
Aug 17 23:20:34 Aug 18 09:20:34 krbtgt/ZARB.HOME@ZARB.HOME
Aug 17 23:20:38 Aug 18 09:20:34 krbtgt/UNIX.ZARB.HOME@ZARB.HOME
Aug 17 23:20:38 Aug 18 09:20:34 ldap/kolyma.unix.zarb.home@UNIX.ZARB.HOME
```

Le principal de la relation de confiance apparaît bien dans la liste des tickets de l'utilisateur. Les logs des deux KDC montrent également ces différents échanges. Sur le KDC du royaume de l'utilisateur :

```
2010-08-17T23:20:34 AS-REQ guillaume@ZARB.HOME from IPv4:192.168.0.5 for
krbtgt/ZARB.HOME@ZARB.HOME
2010-08-17T23:20:38 TGS-REQ guillaume@ZARB.HOME from IPv4:192.168.0.5 for
krbtgt/UNIX.ZARB.HOME@ZARB.HOME
```

Sur le KDC du royaume de la ressource :

```
2010-08-17T23:20:38 TGS-REQ guillaume@ZARB.HOME from IPv4:192.168.0.5 for
ldap/kolyma.unix.zarb.home@UNIX.ZARB.HOME [concealable]
2010-08-17T23:20:38 Client not found in database: no such entry found in hdb
2010-08-17T23:20:38 cross-realm ZARB.HOME -> UNIX.ZARB.HOME
```

2 Confiance entre royaume Unix et domaine Active Directory

Établir une confiance entre un royaume Unix et un domaine Active Directory est plus compliqué, pour plusieurs raisons. D'abord, il y a les différences d'implémentation d'une part et les nombreuses informations erronées que l'on peut trouver à ce sujet un peu partout. Ensuite, il ne s'agit pas de deux concepts équivalents, car un domaine Active Directory est bien plus qu'un simple royaume Kerberos, et qu'une simple relation de confiance n'a que peu d'intérêt si elle ne s'accompagne pas d'une correspondance des comptes utilisateurs entre les deux systèmes.

Cette partie montre donc comment établir une relation fonctionnelle, étape par étape, ainsi que quelques exemples d'utilisation concrète de celle-ci.

3 Mise en place de la relation de confiance

En fait, la véritable difficulté consiste à établir le secret partagé, qui implique une clé identique de part et d'autre, c'est-à-dire le même mot de passe chiffré par le même algorithme, pour qu'ils puissent déchiffrer les messages qu'ils se transmettent. Or, la liste des algorithmes supportés et de ceux réellement utilisés pour générer ces clés varie en fonction de chaque implémentation et des versions de celles-ci. Cette différence se manifeste forcément lorsque l'on tente de faire communiquer un KDC Active Directory et un KDC Unix, mais elle serait tout aussi évidente en tentant de faire communiquer une version ancienne de MIT Kerberos avec une version récente de Heimdal...

Donc, contrairement aux explications qui datent de Windows 2000, il est totalement inexact qu'il faille brider le chiffrement à DES pour tous les principaux du royaume Unix. Heureusement, d'ailleurs, vu que cet algorithme qui date des années 1970 est largement périmé à l'heure actuelle. D'une part, la seule contrainte sur l'algorithme de chiffrement concerne uniquement les principaux de la relation de confiance. Ensuite, la limite supérieure n'est pas DES, mais l'algorithme configuré côté Active Directory pour cette relation, lui-même limité par la liste des algorithmes supportés.

Le tableau 1 donne les algorithmes par défaut et disponibles en fonction des différentes versions de Windows.

Du côté Windows, une relation de confiance vers un royaume Kerberos est une forme restreinte de confiance vers un autre domaine Active Directory. Elle peut se configurer en ligne de commandes avec l'outil **netdom.exe**, ou via l'interface

Version	Algorithme maximal	Algorithme par défaut
Windows 2000	DES	DES
Windows 2003	DES	DES
Windows 2003R2	RC4	DES
Windows 2008	AES	RC4

Tableau 1 : Chiffrements supportés par Windows

graphique (figure 5). Cette relation peut être unidirectionnelle ou bidirectionnelle, transitive ou non-transitive. Ensuite, il vaut mieux augmenter au maximum le chiffrement utilisé pour la relation, sauf si on a une version Unix préhistorique en face. Là par contre, c'est uniquement en ligne de commandes et de plus, l'outil varie en fonction de la version de Windows.

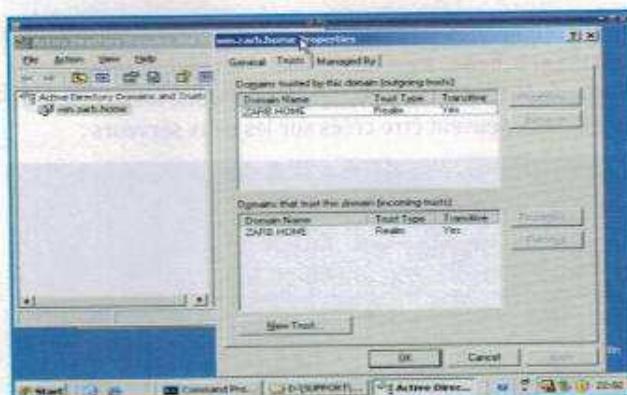


Figure 5 : Configuration d'une relation de confiance sous Windows Server 2003

Pour Windows 2003R2, il est nécessaire d'utiliser la commande **ktpass**, qui fait partie des *Support Tools* pour Windows 2003, disponible depuis le CD d'installation ou sur le site web de Microsoft. Cette commande s'utilise ainsi :

```
C:\> ktpass /MitRealmName ZARB_HOME /TrustEncryp RC4
```

Pour Windows 2008, la même opération se fait avec la commande **ksetup**, qui est installée par défaut :

```
C:\> ksetup /SetEncTypeAttr ZARB_HOME AES256-SHA1
```

Du côté Unix, la mise en place se fait comme précédemment. Par contre, il faut veiller à ce qu'il n'y ait aucune clé utilisant un algorithme de chiffrement plus fort que celui configuré pour la relation de confiance. En effet, il y a bien négociation de la clé à utiliser en fonction des capacités des deux protagonistes, mais ces négociations n'interviennent qu'entre le client et chacun des serveurs tour à tour, jamais entre les serveurs directement. Le plus simple est donc de générer le principal comme d'habitude, puis de supprimer les clés excédentaires, à l'aide de la commande **kadmin del enctype**:

```
[root@tcheka ~]# kadmin -l
kadmin> add --use-defaults krbtgt/WIN.ZARB.HOME@ZARB.HOME
krbtgt/WIN.ZARB.HOME's Password:
Verifying - krbtgt/WIN.ZARB.HOME@ZARB.HOME's Password:
kadmin> get -o Principal,Keytypes krbtgt/WIN.ZARB.HOME@ZARB.HOME
Principal: krbtgt/WIN.ZARB.HOME@ZARB.HOME
Keytypes: des-cbc-crc(pw-salt), aes256-cts-hmac-sha1-96(pw-salt), des3-cbc-
sha1(pw-salt), arcfour-hmac-md5(pw-salt)
kadmin> del enctype krbtgt/WIN.ZARB.HOME@ZARB.HOME aes256-cts-hmac-sha1-96
kadmin> del enctype krbtgt/WIN.ZARB.HOME@ZARB.HOME des3-cbc-sha1
kadmin> get -o Principal,Keytypes krbtgt/WIN.ZARB.HOME@ZARB.HOME
Principal: krbtgt/WIN.ZARB.HOME@ZARB.HOME
Keytypes: des-cbc-crc(pw-salt), arcfour-hmac-md5(pw-salt)
kadmin> add --use-defaults krbtgt/ZARB.HOME@WIN.ZARB.HOME
krbtgt/ZARB.HOME@WIN.ZARB.HOME's Password:
Verifying - krbtgt/ZARB.HOME@WIN.ZARB.HOME's Password:
kadmin> get krbtgt/ZARB.HOME@WIN.ZARB.HOME
Principal: krbtgt/ZARB.HOME@WIN.ZARB.HOME
Keytypes: des-cbc-crc(pw-salt), arcfour-hmac-md5(pw-salt)
kadmin> del enctype krbtgt/ZARB.HOME@WIN.ZARB.HOME des256-cts-hmac-sha1-96
kadmin> del enctype krbtgt/ZARB.HOME@WIN.ZARB.HOME des3-cbc-sha1
kadmin> get -o Principal,Keytypes krbtgt/ZARB.HOME@WIN.ZARB.HOME
Principal: krbtgt/ZARB.HOME@WIN.ZARB.HOME
Keytypes: des-cbc-crc(pw-salt), arcfour-hmac-md5(pw-salt)
```

```
C:> ksetup /AddKdc ZARB.HOME tcheka.zarb.home
C:> ksetup
default realm = win.zarb.home (NT Domain)
ZARB.HOME:
    KDC = tcheka.zarb.home
    Realm Flags = 0x0 none
No user mappings defined
```

Pour les versions plus récentes, par contre, cet outil n'est pas disponible et il faut faire la modification directement dans le registre. Il faut en fait ajouter une clé correspondant au nom du royaume au niveau de la clé **HK_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\Kerberos\Domains**, avec une valeur **KdcNames** de type **REG_MULTI_SZ** contenant les noms des différents KDC.

3.2 Mise en place d'une correspondance de compte utilisateur

Techniquement, la partie purement Kerberos est en place. Il devient donc possible d'accéder à un service Windows depuis le monde Unix (et vice versa). Néanmoins, ceci ne permet d'utiliser que des services avec une politique d'accès autorisant n'importe quel utilisateur authentifié, sans distinction. Pour n'importe quelle politique d'accès un peu plus évoluée, il faut pouvoir associer un compte utilisateur au principal Kerberos, de façon à gérer ses autorisations. Il faut donc également mettre en place une correspondance des comptes utilisateurs.

Du côté Windows, ceci se fait en ajoutant une identité alternative pour l'utilisateur concerné, de la forme **Kerberos:principal@MON.ROYAUME.UNIX**. Ceci peut se faire directement via l'interface LDAP d'Active Directory, en ajoutant cette valeur à l'attribut multivalue **altSecurityIdentities** au compte utilisateur, ou alors via l'interface graphique (figure 6).



Figure 6 : Configuration d'une relation de confiance sous Windows Server 2003

3.1 Configuration du royaume externe

La relation est en place, il faut maintenant configurer sur chaque machine le royaume Kerberos externe.

Sur la machine Linux, il suffit d'utiliser les enregistrements DNS, qui sont automatiquement mis en place par Active Directory, ou alors d'ajouter explicitement le nouveau royaume au fichier de configuration **/etc/krb5.conf** :

```
[libdefaults]
default_realm = ZARB.HOME

[realms]
ZARB.HOME = {
    kdc      = tcheka.zarb.home
    kdc_master = tcheka.zarb.home
    admin_server = tcheka.zarb.home
    kpasswd_server = tcheka.zarb.home
}
WIN.ZARB.HOME = {
    kdc      = taogei.win.zarb.home
}

[domain_realm]
.zarb.home = ZARB.HOME
.win.zarb.home = WIN.ZARB.HOME
```

Sur la machine Windows, c'est un peu plus pénible à faire. En effet, si les enregistrements DNS sont massivement utilisés dans Active Directory pour localiser toutes sortes de ressources, Windows est incapable d'utiliser cette technique pour identifier les KDC des domaines Kerberos externes. Il faut donc les configurer explicitement.

Pour Windows XP ou Server 2003, il est possible d'utiliser l'outil **ksetup.exe**, lui aussi en provenance du paquetage *Support Tools*. Dans notre cas, ceci donne :

Cette association permet donc à un utilisateur se présentant à un service Windows avec un principal issu du royaume Unix d'être immédiatement reconnu comme un utilisateur Windows à part entière.

Du côté Unix, c'est moins évident à réaliser, voire impossible, et dépend principalement de la façon dont sont gérés les comptes utilisateurs. Dans le cas d'un annuaire LDAP, l'utilisation du schéma **krb5-kdc**, présenté dans la partie précédente (section 3.4), permet d'associer un compte à un principal, mais à un seul uniquement, puisque l'attribut **krb5PrincipalName** n'est pas multivalué. S'il existe déjà une association avec un principal du royaume Unix, il n'est donc pas possible d'associer le compte avec un principal du royaume Active Directory. Ceci dit, ce problème semble très théorique, car je n'ai pas encore rencontré de cas où cette correspondance était nécessaire et il est toujours possible de le contourner en utilisant le principal du royaume Unix à la place du principal Active Directory dès l'ouverture de session, comme présenté ci-après (section 3.4).

3.3 Accès à un partage CIFS depuis le monde Unix

Kerberos est le mécanisme d'authentification principal dans un domaine Active Directory, NTLM n'étant utilisé que si cette solution n'est pas disponible. En conséquence, la plupart des services Windows sont utilisables de cette manière et les partages CIFS n'échappent pas à la règle. Démonstration immédiate avec l'outil **smbclient** de Samba :

```
[guillaume@bertrand ~]$ klist
Credentials cache: FILE:/tmp/krb5cc_500
  Principal: guillaume@ZARB. HOME

  Issued      Expires      Principal
Sep 10 23:56:35 Sep 11 09:56:35 krbtgt/ZARB. HOME@ZARB. HOME
[guillaume@bertrand ~]$ smbclient -k //laogai/documents
OS=Windows Server 2003 3790 Service Pack 2|Server=Windows Server 2003 5.2|
smb: > exit
[guillaume@bertrand ~]$ klist
Credentials cache: FILE:/tmp/krb5cc_500
  Principal: guillaume@ZARB. HOME

  Issued      Expires      Principal
Sep 10 23:49:29 Sep 11 09:49:29 krbtgt/ZARB. HOME@ZARB. HOME
Sep 10 23:49:36 Sep 11 09:49:29 krbtgt/WIN.ZARB. HOME@ZARB. HOME
Sep 10 23:56:18 Sep 11 09:49:29 laogai@WIN.ZARB. HOME
Sep 10 23:56:18 Sep 11 09:49:29 laogai@WIN.ZARB. HOME
```

Le premier principal est celui du service de ticket pour le royaume **ZARB. HOME**, le second est le principal de la relation de confiance avec le royaume **WIN.ZARB. HOME** et le troisième celui de la machine **laogai** (les habitués du monde Windows y reconnaîtront la syntaxe pour un compte utilisateur machine). Le fait que ce dernier soit dupliqué semble être une anomalie bénigne.

3.4 Ouverture de session Windows via le royaume Kerberos Unix

Une autre possibilité offerte par la relation de confiance consiste à pouvoir ouvrir une session sur un poste de travail Windows en s'authentifiant via le royaume Kerberos Unix.

Ceci permet notamment de simplifier les procédures de synchronisation de mots de passe, en court-circuitant complètement le mot de passe stocké dans Active Directory. Et ceci permet également de contourner la difficulté à associer le principal de l'utilisateur dans le domaine Active Directory à un compte utilisateur Unix, puisque c'est le principal Unix qui est alors utilisé pour s'identifier.



Figure 7 : Ouverture d'une session Windows par authentification externe

Sur Windows Server 2003 et les versions antérieures, le choix du domaine d'authentification (local, domaine Active Directory ou royaume Kerberos) se fait sous la forme d'une liste déroulante. Par contre, à partir de Vista, cette liste n'existe plus et il faut indiquer explicitement ce domaine par la syntaxe **ROYAUME utilisateur**, ou encore **utilisateur@ROYAUME**. Pour l'utilisateur normalement constitué, le fait d'avoir à utiliser une syntaxe différente de ses habitudes constitue malheureusement une source de confusion majeure. Il devrait être possible néanmoins de réduire celle-ci en changeant le domaine d'authentification par défaut, ce qui est possible grâce à une option de configuration cachée dans les méandres des politiques disponibles, mais je n'ai jamais testé.

Lorsque l'utilisateur s'authentifie via le domaine Active Directory lors de l'ouverture de la session, c'est-à-dire le cas classique, voici à quoi ressemble sa liste de tickets :

```
[~]>klist.exe tgt
Cached TGT:
ServiceName: krbtgt
TargetName: krbtgt
FullServiceName: guillaume
DomainName: WIN.ZARB. HOME
TargetDomainName: WIN.ZARB. HOME
AltTargetDomainName: WIN.ZARB. HOME
TicketFlags: 0x40e00000
KeyExpirationTime: 1/1/1601 2:00:00
StartTime: 9/14/2010 22:16:17
EndTime: 9/15/2010 8:16:17
RenewUntil: 9/21/2010 22:16:17
TimeSkew: 1/1/1601 2:00:00
```

```
C:\>klist tickets
Cached Tickets: (2)

Server: krbtgt/WIN.ZARB.HOME@WIN.ZARB.HOME
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
End Time: 9/15/2010 8:16:17
Renew Time: 9/21/2010 22:16:17

Server: host/taogai.win.zarb.home@WIN.ZARB.HOME
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
End Time: 9/15/2010 8:16:17
Renew Time: 9/21/2010 22:16:17
```

Les deux tickets correspondent au service de ticket du domaine Active Directory et au service d'ouverture de session sur la machine.

Par contre, quand l'utilisateur s'authentifie via le royaume Kerberos Unix, cette liste ressemble alors à ceci :

```
C:\>klist tgt
Cached TGT:
ServiceName: krbtgt
TargetName: krbtgt
FullServiceName: guillaume
DomainName: ZARB.HOME
TargetDomainName: ZARB.HOME
AltTargetDomainName: ZARB.HOME
TicketFlags: 0x40e00000
KeyExpirationTime: 1/1/1601 2:00:00
StartTime: 9/28/2010 23:12:34
EndTime: 9/27/2010 23:12:34
RenewUntil: 9/27/2010 23:12:34
TimeSkew: 1/1/1601 2:00:00
C:\>klist tickets
Cached Tickets: (5)

Server: krbtgt/ZARB.HOME@ZARB.HOME
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
End Time: 9/21/2010 23:12:34
Renew Time: 9/27/2010 23:12:34

Server: krbtgt/WIN.ZARB.HOME@ZARB.HOME
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
End Time: 9/21/2010 23:12:34
Renew Time: 9/27/2010 23:12:34

Server: krbtgt/ZARB.HOME@ZARB.HOME
KerbTicket Encryption Type: Unknown (18)
End Time: 9/21/2010 23:12:34
Renew Time: 9/27/2010 23:12:34

Server: krbtgt/ZARB.HOME@ZARB.HOME
KerbTicket Encryption Type: Unknown (18)
End Time: 9/21/2010 23:12:34
Renew Time: 9/27/2010 23:12:34

Server: cifs/taogai.win.zarb.home@WIN.ZARB.HOME
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
End Time: 9/21/2010 9:16:48
Renew Time: 9/21/2010 23:16:48

Server: host/taogai.win.zarb.home@WIN.ZARB.HOME
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
End Time: 9/21/2010 9:16:41
Renew Time: 9/21/2010 23:16:41
```

Cette fois-ci, les tickets correspondent au service de ticket du royaume Unix (deux fois, dont l'une avec un chiffrement apparemment non supporté), à la relation de confiance, puis au service d'accès aux fichiers et d'ouverture de session de la machine locale.

3.5 Accès à un site web Unix depuis le monde Windows

Une fois l'utilisateur authentifié via le royaume Kerberos Unix, il est relativement simple d'accéder à une ressource Unix, tel que le serveur web mis en place dans la deuxième partie (section 4). La commande **klist** permet de constater l'acquisition automatique d'un nouveau ticket, correspondant au principal **HTTP/loubianka.zarb.home@ZARB.HOME** du serveur web visé.

```
C:\>klist tickets
Cached Tickets: (6)

Server: krbtgt/ZARB.HOME@ZARB.HOME
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
End Time: 9/15/2010 23:17:09
Renew Time: 9/21/2010 23:17:09

Server: krbtgt/WIN.ZARB.HOME@ZARB.HOME
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
End Time: 9/15/2010 23:17:09
Renew Time: 9/21/2010 23:17:09

Server: krbtgt/ZARB.HOME@ZARB.HOME
KerbTicket Encryption Type: Unknown (18)
End Time: 9/15/2010 23:17:09
Renew Time: 9/21/2010 23:17:09

Server: HTTP/loubianka.zarb.home@ZARB.HOME
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
End Time: 9/15/2010 23:17:09
Renew Time: 9/21/2010 23:17:09

Server: cifs/taogai.win.zarb.home@WIN.ZARB.HOME
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
End Time: 9/15/2010 9:16:48
Renew Time: 9/21/2010 23:16:48

Server: host/taogai.win.zarb.home@WIN.ZARB.HOME
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
End Time: 9/15/2010 9:16:41
Renew Time: 9/21/2010 23:16:41
```

4 Résolution de problèmes

Kerberos est un système relativement complexe et il n'est pas rare de devoir tâtonner quelque peu lorsqu'on tente de le mettre en place. Voici quelques conseils qui peuvent s'avérer utiles en cas d'erreur, dans le monde Unix tout d'abord, dans le monde Windows ensuite.

4.1 Unix

Du côté client, les problèmes se traduisent généralement par un message d'erreur générique, ne permettant pas d'identifier le problème précisément. Cette opacité est principalement due à la nature de couche d'abstraction lorsque GSSAPI est utilisée, mais également sans doute au souci d'éviter au maximum toute fuite d'information à un éventuel attaquant.

Quoiqu'il en soit, voici l'exemple type d'une connexion SSH qui échoue à utiliser la méthode d'authentification **gssapi-with-mic**:

```
debug1: Next authentication method: gssapi-with-mic
debug1: Unspecified GSS failure. Minor code may provide more information
Server not found in Kerberos database
debug1: Unspecified GSS failure. Minor code may provide more information
Server not found in Kerberos database
debug1: Unspecified GSS failure. Minor code may provide more information
```

L'utilisation de la commande **klist**, pour inspecter la liste des tickets, permet de vérifier que le ticket de service est bien présent :

```
guillaume@beria ~]$ klist
Credentials cache: FILE:/tmp/krb5cc_500
Principal: guillaume@ZARB.HOME

Issued Expires Principal
Apr 15 22:52:54 Apr 16 08:52:55 krbtgt/ZARB.HOME@ZARB.HOME
Apr 15 22:53:04 Apr 16 08:52:55 host/toubianka.zarb.home@ZARB.HOME
```

Ceci élimine tout de suite un problème de configuration au niveau du client, puisqu'il arrive à communiquer avec le KDC et à obtenir un ticket de service. Le problème se situe donc vraisemblablement au niveau du service cible.

L'inspection des logs de ce service ne révèle cependant rien d'intéressant. Le seul message relatif à Kerberos est le suivant :

```
debug1: ssh_gssapi_storecreds: Not a GSSAPI mechanism
```

Bref, aucune piste. On en revient donc à la *check-list* classique suivante :

- les horloges des machines sont-elles synchronisées ?
- le fichier **keytab** est-il lisible par le processus associé au service ?
- le fichier **keytab** comprend-il un principal correctement formé pour le service et l'hôte ?
- l'algorithme de chiffrement le plus fort utilisé par le principal est-il supporté par le service ?

Et dans ce cas précis, le fichier **keytab** était tout simplement vide...

L'autre piste qui s'avère parfois utile, ce sont les logs du KDC, qui gardent les traces de toutes les demandes de TGT d'une part, de ticket de service d'autre part. Les messages suivants sont assez classiques :

```
2010-12-13T11:17:49 Failed to verify AP-REQ: Clock skew too great
2010-12-13T14:23:56 Failed to decrypt PA-DATA -- guillaume@ZARB.HOME
  (enctype aes256-cts-hmac-sha1-96)
error Decrypt integrity check failed
```

Le premier correspond à un décalage d'horloge trop grand. Le second à une simple erreur de saisie de mot de passe.

4.2 Windows

Du côté client, il ne faut pas s'attendre au moindre message d'erreur utilisable. Par contre, il est possible d'utiliser la commande **klist**, présente de base dans Windows Server 2008 et dans le *Resource Kit* pour la version Server 2003, pour inspecter la liste des tickets de l'utilisateur, comme sous Unix. Il existe également une version graphique, **kerbtray**, qui se loge dans la zone de notification et qui est fort pratique pour dépanner les utilisateurs à distance.

Du côté serveur, pour activer les traces, il faut modifier la base de registre. Les détails techniques sont présentés dans cet article de la base de connaissance [1].

Conclusion générale

Cette série d'article a présenté Kerberos, en partant des concepts et des principes de base, puis en exposant des cas de mise en œuvre simples dans un environnement minimal et enfin, en dévoilant certains problèmes que l'on rencontre lorsque l'on sort de la maquette pour déployer dans un véritable environnement réel, avec une multitude de contraintes diverses.

Cette dernière partie n'a aucune prétention à être exhaustive, juste de donner un ordre d'idée des difficultés qui existent dans un univers hétérogène et de la nécessité de bien maîtriser les mécanismes sous-jacents pour comprendre ce qui se passe. J'aurais pu continuer par exemple sur des sujets encore plus amusants, comme les relations de confiance entre un nombre quelconque de royaumes, mais ce genre de problème sort largement du cadre d'un article d'introduction et il existe de meilleures sources d'informations, comme par exemple l'excellent ouvrage d'O'Reilly sur le sujet, « Kerberos: The Definitive Guide » [2].

En plus de cette complexité, Kerberos n'apporte finalement qu'un mécanisme

d'authentification, qu'il faut ajouter et intégrer au système de gestion des comptes utilisateurs déjà en place. Bien que ceci ait l'avantage de ne pas forcément remettre en cause l'existant, ce qui rend la transition plus simple, il est raisonnable de se poser la question fatidique : le jeu en vaut-il vraiment la chandelle ?

La réponse à cette question ne peut être qu'individuelle, tant elle dépend du contexte. En particulier, plus que l'hétérogénéité des machines auxquelles on souhaite offrir des services kerberisés, c'est leur mode de gestion qui va déterminer le coût de déploiement d'une part et le bénéfice du résultat d'autre part.

En effet, et j'ai essayé de le prouver tout au long de l'article, toutes les implémentations Kerberos sont compatibles, c'est l'immense avantage d'un standard. Mais par contre, leur mise en œuvre implique certains pré-requis, comme un domaine Active Directory dans le cas de Windows, et un minimum de bagage technique dans le cas Unix. Dans un environnement idéal, homogène, maîtrisé de bout en bout par une équipe technique, avec des utilisateurs dont les prérogatives sont strictement limitées à l'utilisation des ressources informatiques, ces conditions sont faciles à réaliser. Mais dans un environnement plus ouvert, avec des visiteurs par exemple, ou des utilisateurs gérant eux-mêmes leur machine, comme c'est souvent le cas dans le monde universitaire, c'est beaucoup plus difficile. En effet, l'utilisateur moyen montre généralement une forte préférence pour suivre une procédure fastidieuse mais connue, comme entrer dix fois de suite son mot de passe et cliquer, plutôt que de consacrer dix minutes à lire une documentation pour faire autrement. Quiconque a déjà essayé d'expliquer que l'un des intérêts de l'authentification par clé pour SSH, c'était d'utiliser un agent plutôt que de saisir sa phrase de passe à chaque connexion en aura déjà fait l'expérience...

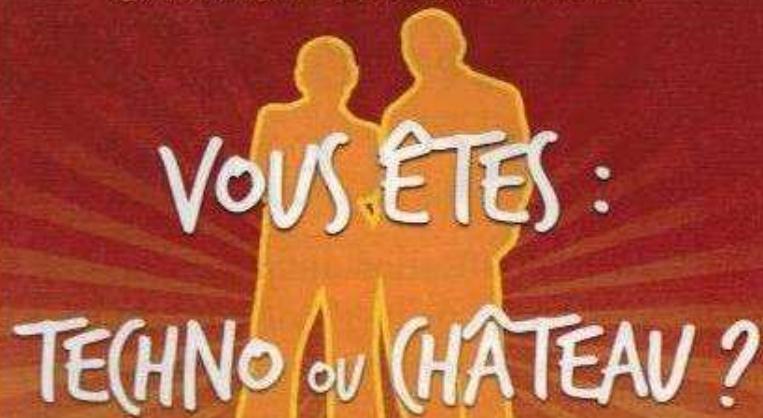
Au final, il appartient à chacun de déterminer si les intérêts d'un mécanisme d'authentification à la fois fortement sécurisé et pratique à utiliser surpassent le coût de déploiement et éventuellement de formation des utilisateurs, s'il n'est pas possible de rendre son utilisation transparente pour eux. Et bien sûr, à moins de vivre dans un monde idéal, il vaut mieux toujours prévoir un mécanisme complémentaire de repli pour les multiples situations de la vraie vie dans lesquelles les merveilles du SSO sécurisé ne sera pas accessible... ■

Références

- [1] <http://support.microsoft.com/kb/837361/en-us>
- [2] <http://oreilly.com/catalog/9780596004033>

LINAGORA
FORMATION

LINAGORA FORMATION LANCE UNE
CAMPAGNE JUSQU'A FIN 2011 :



Participez à une de nos formations*
et recevez votre cadeau :

**Un séjour dans un château
ou une tablette tactile !**

* Offre valable sur une large sélection de formations
disponible sur :

www.linagora.com

LINAGORA

OBM

**LA MESSAGERIE COLLABORATIVE
D'ENTREPRISE OPEN SOURCE**

Découvrez la nouvelle version d'OBM 2.4

- Synchronisation avec smartphones, PDA et iPhone®
- Synchro calendrier Thunderbird / Outlook®
- Amélioration du Webmail full Ajax Web 2.0
- Messagerie instantanée
- Consultation de calendriers externes etc...

Sur notre site
www.obm.org