Table des matières

1 RESUME

Ce projet de fin d'étude consiste à construire une infrastructure VOIP à l'aide d'un serveur Asterisk et un serveur d'authentification Radius. Le but est de pouvoir intégrer les 2 serveurs dans un réseau existant à infrastructure VOIP de la société ***Switzernet***.

La motivation principale de ce projet est de mettre en place un système de facturation de la société ***Switzernet***.

Asterisk est un PBX servant de commutateur ou d'interface pour n'importe quel type d'équipements téléphoniques (analogiques, numériques, VOIP…).

Radius est un serveur d'authentification qui permet d'identifier un utilisateur par son nom et un mot de passe ou par certificat électronique ou par adresse MAC.

Ce rapport détaille les étapes nécessaires pour arriver à mettre en place cette infrastructure VOIP à l'aide d'un serveur Asterisk qui identifie les utilisateurs et facture les appels via Radius. Le travail effectué dans ce projet est subdivisé en 3 parties essentielles :

- Partie recherche sur la possibilité d'intégrer Radius dans le serveur Asterisk.
- Partie conception d'une plateforme de test en se basant sur la solution trouvée au premier point.
- Dernière partie test final de la plateforme réalisée avec l'infrastructure VOIP de la société Switzernet.

Durant ce projet, chaque étape citée-ci-dessus a été documentée et envoyée au fur à mesure de l'avancement du travail à la société Switzernet pour valider le passage de ces étapes.

Ce rapport résume les moyens possibles pour intégrer Radius dans un serveur Asterisk. Il cite les étapes de l'installation et la configuration des composants nécessaires pour mettre en place une plateforme d'authentification des utilisateurs et de facturation via radius.
Il contient certaines critiques à propos des solutions trouvées et testées durant ce travail. Il propose aussi des options qui peuvent être intégrées avec un serveur Asterisk et Radius par exemple l'utilisation d'un serveur MYSQL par ces derniers pour externaliser les données d'identification et de facturation.
Notre rapport peut être considéré comme un guide d'installation et de configuration de plusieurs composants utiles et optionnels pour l'implémentation d'un mécanisme de facturation d'une plateforme VOIP utilisant ces 2 types de serveurs.
Dans notre projet, nous avons mis en annexe à part tous les fichiers de configuration nécessaires et utiles pour la construction de notre plateforme de test (serveur Asterisk, Radius et MYSQL).

2 INTRODUCTION

De nos jours, on commence à entendre parler de l'utilisation de la téléphonie IP et les avantages de cette révolution. C'est une des méthodes qui permet aux gens ordinaires de communiquer gratuitement avec d'autres utilisateurs via internet, par exemple Skype. Et de même les entreprises ont commencé à utiliser cette technologie VOIP pour réduire leurs factures téléphoniques, surtout que l'infrastructure VOIP ne coûte pas trop cher pour la mettre en œuvre dans une entreprise, de point de vue matériel.

Actuellement, la technologie VoIP (Voice over IP) ou ToIP (Telephony over IP), s'est étendue et intéresse aussi de plus en plus des nombreux fournisseurs d'accès à Internet.
Depuis plusieurs années, ce secteur est en pleine croissance grâce à deux composantes essentielles. Le premier est que les utilisateurs (comme entreprises) sont en majorité séduits par le gain dans leurs factures téléphoniques et la souplesse dans l'administration de ce type d'infrastructure. Le deuxième est que les constructeurs investissent énormément pour proposer des offres les plus complètes possibles, et certains produits ont atteint une stabilité et maturité dans le marché, de plus l'existence de plusieurs solutions gratuites (open source).

Le but de ce projet est de trouver les moyens possibles d'intégrer de façon interne ou externe Radius dans Asterisk. Pour ce faire, on doit étudier de prés le principe de fonctionnement d'Asterisk et le serveur Radius. Suite à un résultat concret de cette partie précédente, on va implémenter les différentes plateformes pour mettre en place le principe d'authentification /facturation. Ce qui nous amène à une installation et configuration d'un serveur Radius et d'un serveur Asterisk, et à un test du fonctionnement de l'identification à travers 2 client SIP enregistrés au prés d'un serveur Asterisk. La dernière étape, est d'intégrer ces résultats obtenus dans l'infrastructure existante de la société Switzernet. Donc mettre en place un serveur Asterisk dépendant du système de facturation de la société, gérer les propriétés de routages mise par le système de facturation dans Asterisk, et répliquer la base de données Asterisk des enregistrements SIP dans une base données MYSQL.

La motivation de ce projet est de mettre en place un système efficace de facturation de la société ***Switzernet***. Ce projet a été subdivisé sur trois grandes parties importantes qui sont :

- Analyser l'implémentation interne ou externe de Radius dans Asterisk et documenter les étapes d'une solution possible pour l'authentification/facturation.
- Construire une plateforme de test du fonctionnement de l'authentification à travers 2 clients SIP dans Asterisk et la facturation via un serveur Radius.
- Intégrer ces précédents composants dans l'infrastructure existante de Switzernet selon son système de facturation.

Dans ce projet, on va utiliser des environnements open source qui sont efficaces dans le domaine VOIP, et qu'on va expliquer leurs utilités, fonctionnements et surtout comment les intégrer dans une infrastructure VOIP existante de la société Switzernet.

2.1 Asterisk

2.1.1 Historique

« Le nom Asterisk est l'origine du symbole éponyme * qui sous les environnements Unix joue le rôle de joker lors de la recherche ou de sélection de répertoires ou de fichiers. » [9]

Asterisk est un logiciel télécom open-source, et a été développé en langage C sous Linux par Mark Spencer de la société américaine Digium Inc (devenu le sponsor d'Asterisk). Avec l'effort employé par des utilisateurs expérimentés dans les forums et les tutoriaux, Asterisk a atteint une maturité et a gagné la confiance des utilisateurs. Cette réussite est due à l'amélioration du code source, la rédaction des documentations, l'existence de l'aide en ligne, la correction de certains bugs…

Il est apparu dans l'année 2002. Ce projet Asterisk a été utilisé au début pour le tester, puis qu'à la base, il est commercialisé sous forme d'une License GPL. Et à travers ces expériences, Asterisk a pu séduire les utilisateurs ou les entreprises de toutes tailles par ses multiples fonctions et sa stabilité qu'il a acquit au fil du temps.

2.1.2 Définition

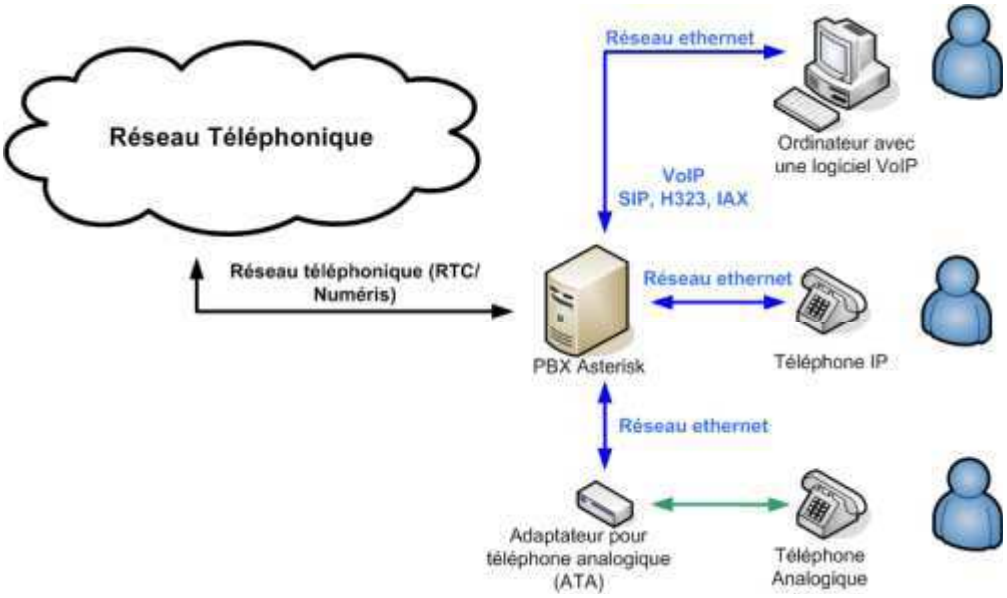
Asterisk est un PBX qui permet d'interfacer n'importe quelle type de matériel de téléphonie (analogique, numérique, VOIP…). Il est compatible presque avec touts les matérielles téléphoniques, que ce soit numériques, ou que soit analogiques. C'est un logiciel bénéficiant d'une grande portabilité sous Linux, ou Windows, BSD… Le plus important est qu'il supporte plusieurs protocoles VOIP. Il est connu par sa richesse et sa flexibilité des fonctionnalités qu'il propose.

2.1.3 Rôle

« Asterisk est IP-PBX qui transforme un ordinateur en "central téléphonique" ou "***PABX***"(***Private Automatic Branch eXchange***), qui est un **autocommutateur téléphonique privé » [1]**. Ce PBX est un commutateur qui relie dans une entreprise les appels d'un poste quelconque vers un autre (appels internes) ou avec un réseau téléphonique public (appels externes).

Asterisk a le rôle d'un middleware entre les technologies de téléphonie VOIP (TDM, SIP …) et les applications (conférence, messagerie vocale, …).
Ce PBX est basé sur le protocole IP. Donc les communications et les paquets échangés sont transportés sous forme plusieurs protocoles de la voix qu'on veut (***SIP, H.323, ADSI, MGCP***).

Cette image et certaines caractéristiques ont été prises depuis la source [1] car elle illustre le fonctionnement de notre PBX.



2.1.4 *Caractéristiques*

- Asterisk offre toutes les fonctions d'un PBX et ses services associés :

  - ✓ la conférence téléphonique.
  - ✓ les répondeurs interactifs.
  - ✓ la mise en attente d'appels.
  - ✓ la distribution des appels.
  - ✓ les mails vocaux.
  - ✓ la musique d'attente.
  - ✓ la génération d'enregistrement d'appels pour l'intégration avec des systèmes de facturation.

- Asterisk fonctionne sur les principaux systèmes d'exploitation (Linux, BSD, Windows, Mac OS X).

- Ces protocoles gèrent la communication et le transport entre les correspondants :

  - ✓ ***H.323*** : il est assez complexe, présente des failles et il est rarement utilisé car il est remplacé par SIP.
  - ✓ ***SIP (Session Initiation Protocol) :*** il est beaucoup utilisé pour la voix sur IP, il est apprécié pour sa simplicité, il ressemble à http, FTP.
  - ✓ ***IAX (Inter Asterisk eXchange)*** : il est développé par Digium pour permettre le dialogue entre serveurs Asterisk. Il est plus simple et rapide que SIP. Donc c'est un protocole propre à Asterisk.
  - ✓ ***MGCP (Media Gateway Control Protocol)***.
  - ✓ ***SCCP (Skinny Client Control Protocol)*** : protocole propriétaire de Cisco.

- Asterisk peut également jouer le rôle de registre et passerelle avec les réseaux publics (RTC, GSM, etc.).

- « Il peut être utilisé dans la téléphonie d'entreprise ***en interne comme en externe***. Ainsi, une entreprise multisite pourra utiliser Asterisk pour ses ***communications entre sites*** et ainsi ***économiser tous les frais*** de télécommunication » [1].

2.2 Radius

2.2.1 Historique

La partie historique ainsi que la partie définition sont prise de [2].

C'est un Projet de la société Livingston sur un protocole d'authentification standard :

- Janvier 1997 : Première version de RADIUS RFC 2058 (authentication) et 2059 (accounting).
- Avril 1997 : Deuxième version de RADIUS RFC 2138 (authentication) et 2139 (accounting).
- Juin 2000 : La dernière version de RADIUS RFC 2865 (authentication) et 2866 (accounting).

2.2.2 Définition

RADIUS (**Remote Authentication Dial-In User Service**) est un protocole client-serveur permettant de centraliser des données d'authentification. Cet identification se fait en vérifiant nom d'utilisateur (***attribut 1 User-Name***) et de mot de passe (***attribut 2 User-Password ou 3 Chap-Password***).

Lorsqu'on parle de Radius, ça implique directement authentification et comptabilisation.

- ***Authentification (authentication)***:

  - ✓ Protocole d'authentification à distance
  - ✓ Centralisation des données d'authentification
  - ✓ Gestion des connexions utilisateurs à des services distants

- ***Comptabilisation (accounting)***:

  - ✓ Utilisé pour assurer la journalisation et la facturation

2.2.3 Rôle

Le protocole RADIUS permet de faire la liaison entre des besoins d'identification et une base d'utilisateurs en assurant le transport des données d'authentification de façon normalisée. L'opération d'authentification est initiée par ***un client du service RADIUS (NAS: Network Access Server)***. Ensuite ce serveur traite la requête de la demande de connexion en accédant si nécessaire à une base externe : base de données SQL, annuaire LDAP, comptes d'utilisateur de machine ou de domaine, et par la suite envoie soit une réponse positive ou négative d'authentification.

Dans le cadre de ce projet, le travail consiste à intégrer un serveur Radius dans un réseau existant à infrastructure VOIP de la société ***Switzernet*** à l'aide d'un serveur Asterisk. Ceci dont le but de gérer l'identification des utilisateurs et la facturation par le serveur Radius.

3 ARCHITECTURE GENERALE DU PROJET

L'architecture de notre projet se compose de deux téléphones VOIP, d'un serveur Asterisk et d'un serveur d'authentification FreeRadius.

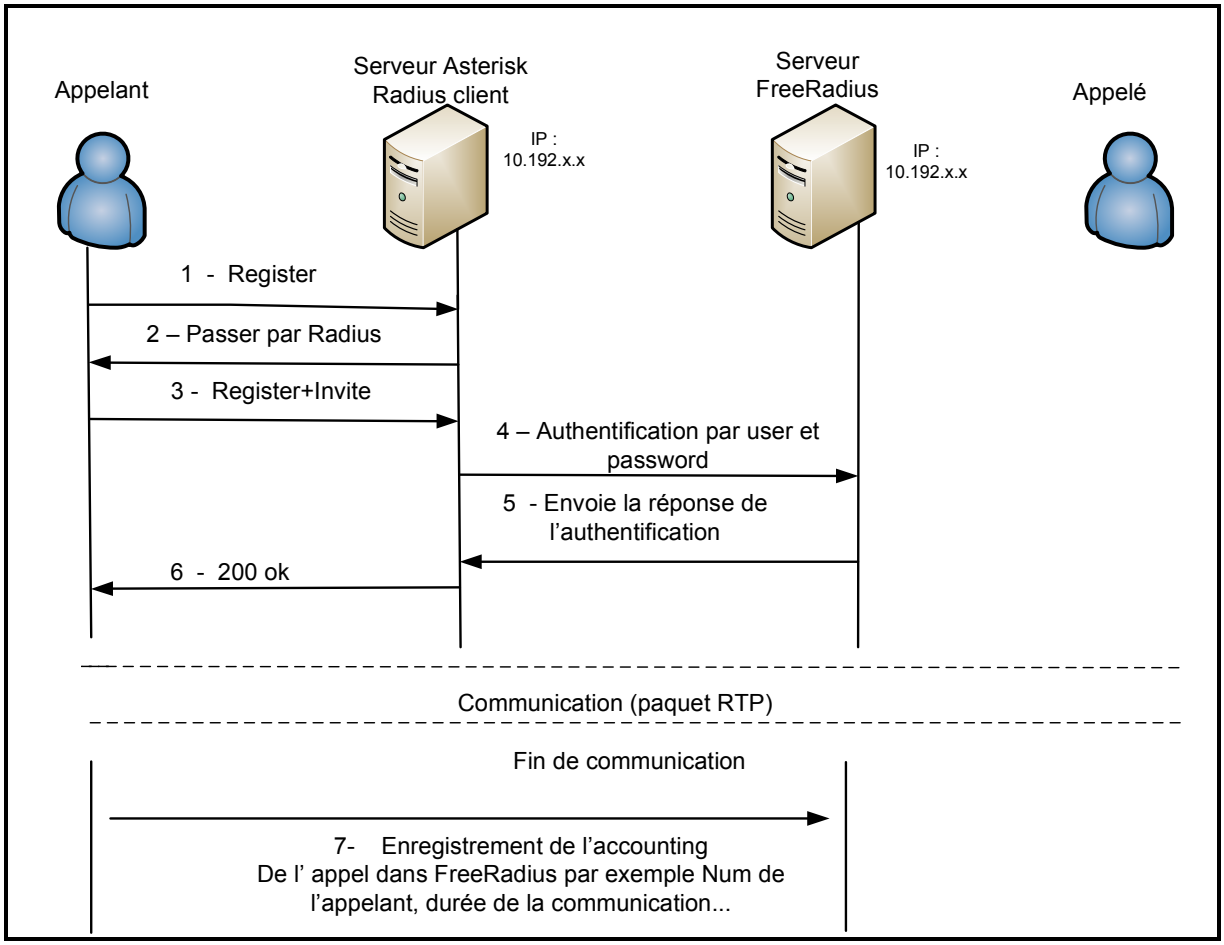


Figure 1: Architecture générale du projet

1. L'user agent SIP va s'enregistrer dans Asterisk par son numéro de téléphone et un mot de passe.
2. Quand il essaye de composer le numéro de l'appelé, il reçoit un message qu'il doit passer par FreeRadius pour effectuer cet appel.
3. Il renvoie une autre demande de connexion à FreeRadius.
4. Le client radius se charge d'envoyer le nom d'utilisateur et son mot de passe pour s'identifier dans FreeRadius.
5. S'il reçoit une réponse positive donc l'utilisateur et le mot de passe sont corrects. Si ce n'est pas le cas, on aura une réponse négative et l'appel ne peut pas être établi.
6. L'user agent reçoit un message qu'il a le droit d'effectuer cet appel donné par FreeRadius, et le téléphone sonnera chez le destinataire.
7. On envoie les données de l'accounting de cet appel dans FreeRadius.

4 ANALYSE THEORIQUE DU PROTOCOLE RADIUS

Le but de notre projet est d'utiliser un serveur Asterisk dans une infrastructure VOIP à travers des utilisateurs s'authentifiant via un serveur Radius. Et ce PBX doit gérer les authentifications et la facturation à l'aide de ce serveur Radius.
Donc pour commencer, il faut étudier de prés les caractéristiques du protocole Radius, le principe d'authentification sur un serveur Radius, les types d'authentifications, et de quoi on a besoin pour utiliser toutes les fonctionnalités de ce serveur. [4]

4.1 Description du protocole Radius

4.1.1 Origine

Radius a pour objectif au début de répondre au problème d'authentification pour des accès à distants, par liaison téléphonique, vers les réseaux des fournisseurs d'accès ou des entreprises. Et maintenant, il est envisagé de l'utiliser pour authentifier les postes de travail sur les réseaux locaux (filaires ou sans fil).

4.1.2 Définition

RADIUS (***Remote Authentication Dial-In User Service***) est un protocole client-serveur permettant de centraliser des données d'authentification.

Radius est un protocole qui répond au modèle ***AAA***. Ces initiales résument ces 3 fonctions :

- ***A = Authentication*** : authentifier l'identité du client.
- ***A = Authorization*** : accorder les droits au client.
- ***A = Accounting*** : enregistrer les données de compatibilité de l'usage du réseau par le client.

Le protocole établit une couche applicative au-dessus de la couche de transport UDP. Les ports utilisés seront :

- ***1812*** pour recevoir les requêtes d'authentification et d'autorisation.
- ***1813*** pour recevoir les requêtes de comptabilité.

Le protocole RADIUS repose principalement sur un serveur (freeRADIUS, OpenRadius,…), relié à une base d'identification (base de données, annuaire LDAP, ...) et un client RADIUS, appelé ***NAS (Network Access Server)***, faisant office d'intermédiaire entre l'utilisateur final et le serveur. L'ensemble des transactions entre le client RADIUS et le serveur RADIUS est chiffrée et authentifiée grâce à un secret partagé.
Le protocole est basé sur des échanges requêtes/réponses avec les clients Radius, c'est-à-dire les NAS. Il n'y a jamais de communication entre le poste de travail et le serveur mais il faut passer plutôt par l'intermédiaire un commutateur ou borne qu'on a appelé le NAS.

4.1.3 Fonctionnement

- Un utilisateur fait une demande de connexion à distance par l'envoie d'une requête au NAS.
- NAS ou client Radius porte l'information de la demande au serveur Radius.
- Serveur Radius réagit à cette demande d'identification par l'envoie une réponse de ses 4 réponses possibles lors de sa consultation à la base de donnée pour savoir le type de scénario d'identification demandé pour l'utilisateur :
  - ***ACCEPT*** : identification accepté par l'utilisateur.
  - ***REJECT*** : identification a échoué pour cet utilisateur.
  - ***CHALLANGE*** : Serveur Radius demande plus d'informations de la part de l'utilisateur et propose un « défi ».
  - ***CHANGE PASSWORD*** : Serveur Radius demande à l'utilisateur un nouveau mot de passe.

Cette image et le principe fonctionnement du protocole sont pris depuis la source [3]. Mais la rédaction est faite par moi-même.

Voir le schéma de la figure 2 pour présenter les différents acteurs dans le fonctionnement :

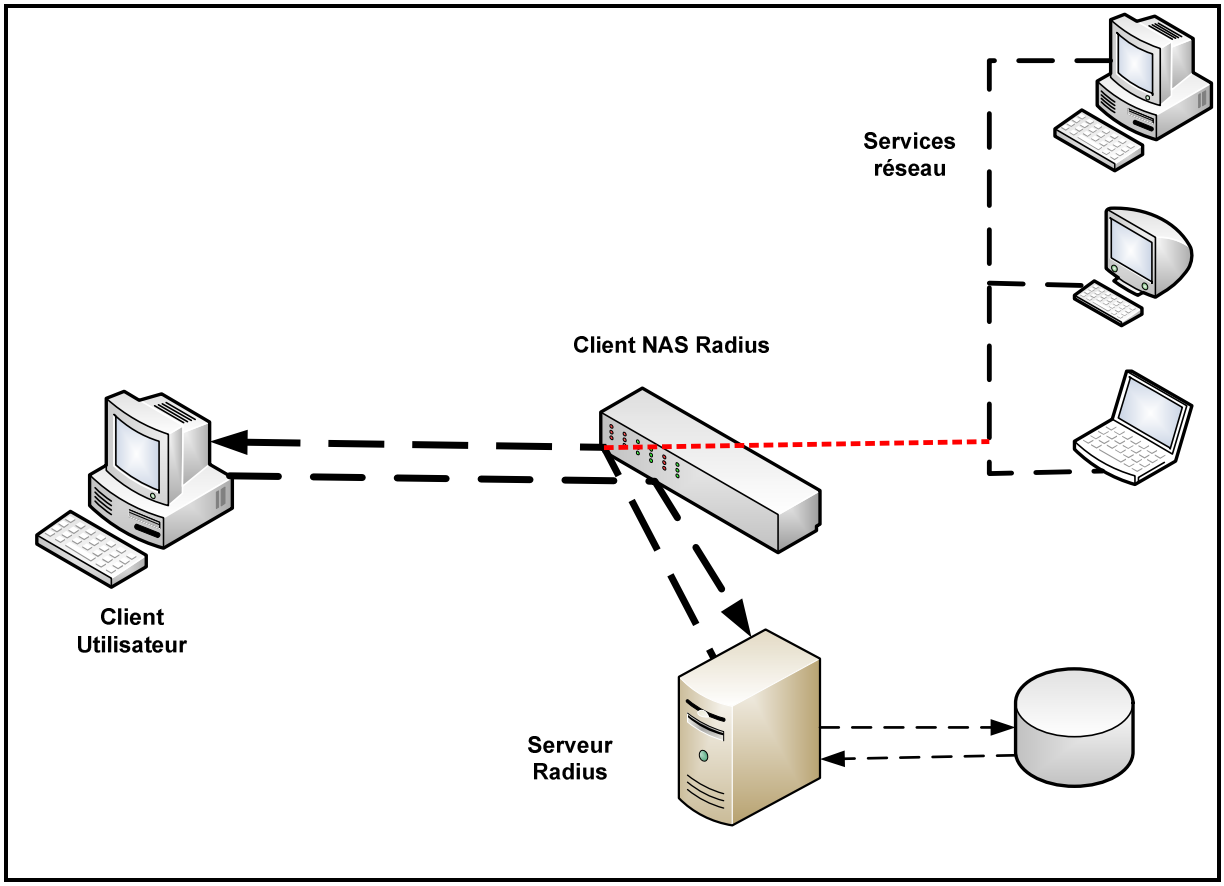


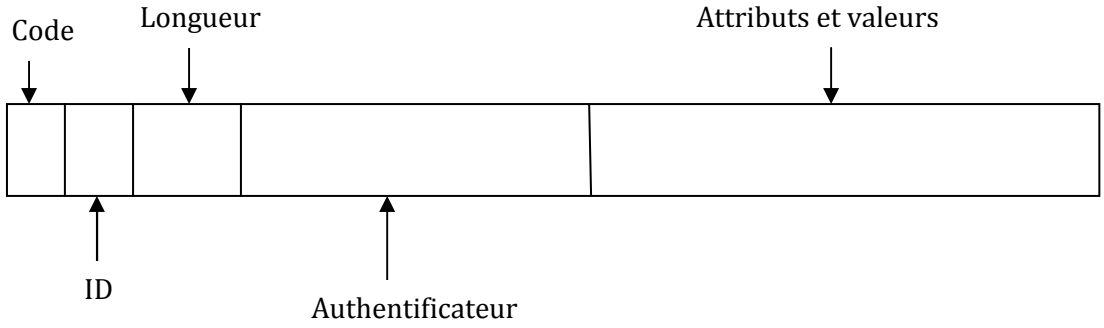
Figure 2: Schéma de fonctionnement de Radius

4.1.4 Format général des paquets

Il est inspiré depuis la source [4] et la source [5] qui est la traduction en français des RFC 2865 et RFC 2866. Il est écrit par mes mots.

Radius utilise 4 types de paquets pour assurer les transactions d'authentification. Et il existe 3 autres types de paquets (***Accounting-Request***, ***Accounting-Response***, ***Accounting-Status***) pour la comptabilité ou la facturation.

La forme générale des paquets :



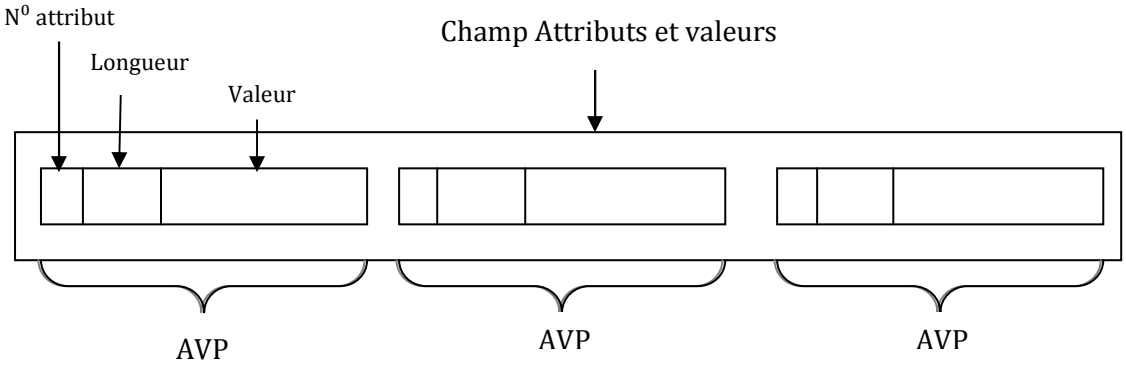
- ***Code***
  Ce champ d'un seul octet contient une valeur qui identifie le type du paquet. Il y a 4 dont les noms sont ***Access-Request*** (code=1), ***Access-Accept*** (code=2), ***Access-Rejet*** (code=3), ***Access-Challenge*** (code=11).
- ***ID***
  Ce champ d'un seul octet contient une valeur permettant au client Radius d'associer les requêtes et les réponses.
- ***Longueur***
  Champ de 16 octets contenant la longueur totale du paquet.
- ***Authentificateur***
  Ce champ de 16 octets a pour but de vérifier l'intégrité des paquets. Il y a 2 types d'authentificateur de requête et de réponse.
  Celle de type requête est inclus dans les paquets de type Access-Request ou Accounting-Request envoyé par le NAS. Sa valeur est calculée de façon aléatoire.
  L'autre type réponse se trouve dans les paquets de type Access-Accept, Access-Challenge ou Access-Reject. Sa valeur est calculée par le serveur à partir d'une formule de hachage MD5 sur une chaîne de caractère. Cette dernière est composé des champs code, ID, longueur, authentificateur de requête et attributs ainsi que d'un secret partagé (un mot de passe connu par le serveur et le NAS).
- ***Attributs et valeurs***
  Ce champ est de longueur variable et contient la charge utile. Donc c'est les attributs et leur valeur qui seront envoyés soit par le NAS en requête ou soit par le serveur en réponse.

4.1.5 Champ Attributs du paquet Radius

Les attributs constituent la partie importante à détailler pour comprendre la richesse du protocole de Radius. Ces champs sont le fondement du protocole.
Chaque attribut possède un numéro d'attribut, au quel est attribué un nom. Les différentes types où quel peut correspondre un numéro sont :

- ✓ ***Adresse IP*** (4 octets)
- ✓ ***Date*** (4 octets)
- ✓ ***Chaîne de caractères*** (max 255 octets)
- ✓ ***Entier*** (4 octets)
- ✓ ***Valeur binaire*** (1 octet)
- ✓ ***Valeur parmi une liste de valeur*** (4 octets)

Ces attributs et leur valeur porte le nom ***Attributes Value-Pair (AVP)***. Ce champ peut contenir plusieurs couples AVP comme la montre la figure en bas :



Le nom de l'attribut ne sera jamais présent dans les paquets mais il y aura à sa place un numéro d'attribut, qui va correspondre à ce nom grâce à un dictionnaire.

Il y a beaucoup d'attributs dans le protocole de Radius mais on va s'intéresser seulement qui sont utile pour l'authentification sur un réseau local. Ce qui est le but dans notre projet. Voilà le détail de quelques attributs les plus importants :

- ***Callback-Number*** : sa valeur est un numéro de téléphone à rappeler. On l'utilise dans le cas d'une connexion par modem. Le scénario est que l'utilisateur appelle son fournisseur, il est authentifié par le serveur qui demande à l'équipement réseau de couper la communication, et de couper la communication et rappeler automatiquement le client sur le numéro stocké dans l'attribut Callback-Number.
- ***User-Name*** : est envoyé par le NAS et contient l'identifiant qui va servir de point d'entrée dans la base du serveur d'authentification. Par exemple User-Name contient la valeur de l'adresse MAC du poste de travail qui se connecte au réseau.
- ***User-Password*** : c'est un mot de passe associé à User-Name transmis par le NAS. Et cette valeur sera valider par le serveur d'authentification seulement lors de la vérification dans la base de données. Dans le cas d'une connexion par MAC, le mot de passe est toujours l'adresse MAC elle-même.

- ***NAS-IP-Adresse*** : c'est une adresse IP du NAS qui communique avec le serveur. Cet attribut est transmis par le NAS lui-même. Il est utile pour permettre un poste de travail de s'authentifier seulement s'il est connecté sur un NAS qui possède cette adresse IP.
- ***NAS-Port*** : c'est un numéro de port du NAS sur le quel est connecté le poste de travail. Cet attribut est transmis par le NAS. Son utilisation sert à authentifier un poste de travail s'il est connecté sur ce numéro de port.
- ***Called-Station-Id*** : c'est l'adresse MAC du NAS. Cet attribut est transmis par le NAS et permet d'authentifier un poste en fonction de l'équipement sur lequel est connecté.
- ***Calling-Station-Id*** : c'est l'adresse MAC du poste de travail qui se connecte au réseau. Il permet d'authentifier un poste par la méthode de connexion MAC. Et il sert aussi à vérifier que cette authentification s'effectue depuis un poste qui a pour adresse MAC la valeur de l'attribut.

4.1.5.1 Attributs vendor

Ces attributs sont renseignés soit par un commutateur ou borne sans fil, soit par un serveur d'authentification lorsqu'il répond aux requêtes. Or Radius propose un attribut ***Vendor Specific Attribute (AVP)*** qui permet d'utiliser la plus part des capacités du matériel. C'est un attribut dont le code est toujours 26 et il encapsule les attributs spécifiques du constructeur du matériel. Sa valeur est calculé sur cette forme (N^0 attribut+ longueur+valeur) et on ajoute le code international du constructeur.

4.1.5.2 Dictionnaires d'attributs

Le dictionnaire est juste une table qui permet de faire la correspondance entre le numéro et leur nom respectif. Cette table contient 3 champs par attribut (le numéro d'attribut, son nom et son type). Il y a au moins un dictionnaires des attributs standards et dictionnaire par constructeur.

4.1.6 Les différentes types de paquets

L'authentification de Radius se fait suivant un dialogue entre le NAS et le serveur. Et dés le début, on a vu qu'il y a 4 types d'échanges qu'on va les décrire.

- ***Access-Request*** : le dialogue commence par un envoie d'un paquet Access-Request de la part du NAS. Il contient au moins l'attribut User-Name, Calling-Station-Id, Nas-Identifier.
- ***Access-Accept*** : ce paquet est renvoyé au NAS par le serveur d'authentification si l'identification transmise par l'access-Request a été correctement validée. Il contient des attributs spécifiques au NAS comme les autorisations accordées par le serveur.
- ***Access-Reject*** : envoyé par le serveur Radius au NAS si l'authentification a échoué.
- ***Access-Challange*** : après d'une réception d'un paquet Access-Request, le serveur peut renvoyer un paquet Access-Challange pour demander plus d'informations comme le mot de passe au poste de travail ou un certificat.

4.2 Critères d'authentification ou d'identification

Avant de traiter les différents types d'authentification, il faut savoir si on veut faire une authentification suivant des utilisateurs ou des machines.
Dans le cas de machine, la machine possède une identité propre et unique lors du sa démarrage ou au moment de la connexion de l'utilisateur.
Dans le cas d'utilisateurs, l'authentification se fait lorsque chaque utilisateur se connecte dans sa session, donc c'est l'identité d'un seul utilisateur qui est considérée au moment de démarrage de la machine. Par exemple, si un utilisateur se connecte sur une machine, elle sera placée sur le VLAN 3, et si un autre utilisateur se connecte sur la même machine, elle sera placée sur VLAN 4.

4.2.1 Authentification par adresse Ethernet (adresse MAC)

L'adresse MAC de la carte Ethernet d'un poste de travail permet d'identifier chaque machine de façon unique et fiable sur un réseau filaire.
Dans le cas d'un réseau sans fil, cette méthode d'adresse MAC n'est pas utilisé car on peut facilement capter cette adresse, puis ce qu'elle circule en clair et on n'a pas le contrôle sur tout le périmètre du réseau sans fil.
Ce type d'authentification est appelé ***Radius-MAC***.

4.2.2 Authentification par identifiant et mot de passe

C'est une authentification par utilisateur sur la base d'un identifiant et un mot de passe. Pour plus de sécurité, il ne faut pas circuler ce mot de passe en clair ou la stocker dans une base de donnée en clair.

4.2.3 Authentification par certificat électronique

Un certificat électronique est une carte d'identité électronique délivrée par une autorité de certification et conforme à la norme X509.
Cette technique d'authentification consiste à faire présenter le client un certificat dont la validité pourra être vérifié par le serveur. Si elle appartient à un utilisateur, c'est une authentification par utilisateur. Et l'autre cas, il peut s'agir d'un certificat d'une machine.

4.3 Matériel nécessaire pour l'authentification

Le matériel nécessaire est :

- Equipements réseau
- Serveur d'authentification ou d'identification
- Postes clients

4.3.1 Equipements réseau

Les éléments réseaux dont le plus principal est le commutateur ou une borne sans fil ou un serveur qui fait office de ***client Radius comme Asterisk (pam_radius / radiusclient-ng)***. Ce qu'on a appelé NAS ou clients Radius, c'est lui qui soumet des requêtes au serveur Radius.

Ce NAS doit impérativement supporter certains standards comme ***le protocole Radius*** et les protocoles IEEE 802.1X. Et dans le cas d'utilisation des réseaux virtuels VLAN, le NAS doit être compatible avec le protocole 802.1Q.

4.3.2 Serveur d'authentification

Le serveur d'authentification a le rôle d'authentifier les requêtes qui lui parviennent d'un poste client. Donc la mission de faire des échanges avec lui pour obtenir la preuve qu'il est bien qui il prétend être. Il y a 2 preuves : une par mot de passe ou un certificat électronique.
Pour ce faire, on a besoin d'un seul serveur d'authentification qui est Radius. Il y a plusieurs sortes commerciales et libres :

- ACS (Cisco sous Windows)
- Aegis (sous Linux)
- IAS (sous Windows)
- ***OpenRadius*** (libre sous Linux)
- ***FreeRadius*** (libre sous Linux, BSD, Windows)

4.3.3 Postes clients

Ça peut être des ordinateurs ayant des logiciels VOIP ou des téléphones IP ou analogiques. Puis que dans notre projet, on s'intéresse à une infrastructure VOIP.

4.4 Principe d'authentification

On a vu qu'il existe 3 types d'authentification : Une authentification par adresse MAC à l'aide du protocole Radius, par identifiant et mot de passe, et l'autre à l'aide du protocole 802.1X et EAP (Extensible Authentification Protocol).
Dans notre cas, on va s'intéressait juste au type d'identification par un nom d'utilisateur et son mot de passe.
L'authentification dans la société Switzernet se fait selon un compte de chaque utilisateur possédant un contrat chez eux. Donc chaque utilisateur possédera un identifiant et un mot de passe dans son compte, ce qui lui permet de téléphoner en utilisant l'infrastructure VOIP de la société. Cette figure 3 ci-dessous explique les étapes d'authentification suivant cette technique :

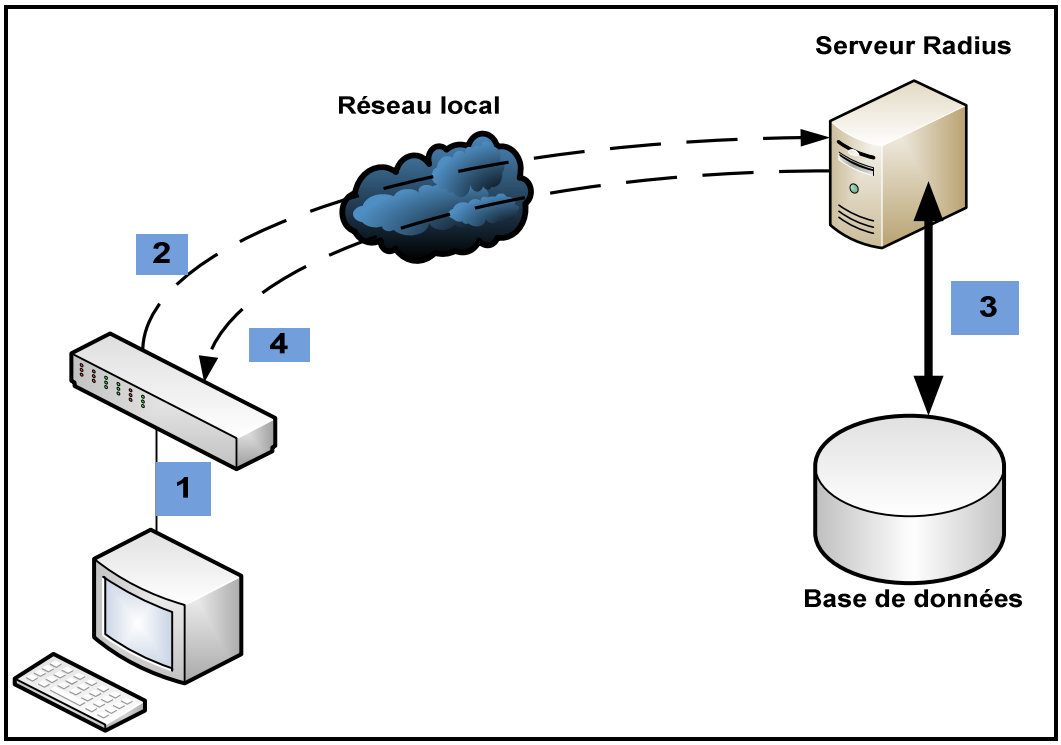


Figure 3: Authentification Radius-MAC

1) Le poste de travail ou un téléphone IP se branche sur un des ports des commutateurs.
2) Le commutateur ou NAS détecte cette connexion et envoie une requête d'authentification (***Access-Request***) au serveur Radius. L'adresse MAC est contenu dans cette requête et elle est utilisée comme un identifiant.
3) Le serveur radius reçoit ce paquet, ensuite il va chercher si cette adresse MAC est enregistrée dans la base donnée, pour trouver le VLAN au quel le poste de travail sera connecté.
4) Le serveur envoie la réponse de la recherche au commutateur. Si la ***réponse est négatif***, il envoie un paquet de type ***Access-Reject*** et le port du commutateur reste fermé ainsi que le poste de travail n'est pas connecté au réseau. Dans le cas d'une ***réponse positive***, il envoie une requête de type ***Access-Accept***, elle contient le numéro de VLAN et le poste peut se connecter.

4.5 Principe de facturation

La facturation sous Asterisk consiste à enregistrer un certain nombre de paramètres lors de l'établissement d'une connexion via le serveur Radius. Ces informations portent par exemple sur le numéro de l'appelant, l'appelé, la durée d'appel, etc…
Ensuite Asterisk envoie ces paramètres de chaque client à ce serveur Radius, qu'on peut les stocker dans une base de données pour créer des factures.

5 CONCEPTION D'UNE SOLUTION D'AUTHENTIFICATION ET D'ACCOUNTING VIA RADIUS

Après avoir effectué une étude sur les protocoles utilisés, ainsi que le principe d'authentification et de facturation, on va commencer la réalisation de la première étape du projet. Dans cette partie, il s'agit de trouver une solution d'intégrer Radius dans Asterisk pour mettre en œuvre le principe d'authentification et de facturation. Cette partie consiste à faire des recherches sur les possibilités d'implémenter cette technique avec ces 2 composants (Asterisk et Radius), et par la suite choisir une solution, soit interne ou externe à Asterisk. Selon le choix pris sur une solution trouvée, on va essayer d'implémenter et citer les détails de la réalisation de cette application VOIP.

5.1 Solutions possibles d'identification\facturation avec Asterisk et Radius

Toute cette partie est prise depuis la source [6] et source [7] qui est en anglais. Cette méthode est bien expliquée. C'est pour cela je l'ai traduit et je l'écris en gros avec ma traduction.

Il y a trois méthodes possibles dont deux externes et une interne :

- ***Externes :***

  - ✓ ***PortaOne's Radius*** client pour Asterisk comme module externe.
  - ✓ ***CW Radius*** pour Asterisk comme module externe.

- ***Interne***

  - ✓ Asterisk contient un fichier contenant des informations sur le numéro appelant, numéro appelé, durée de l'appel, date de l'appel, heure d'appel… ce qui est très intéressante pour la facturation. Et l'authentification se fait par l'intégration interne d'un client Radius à Asterisk. Ce dernier peut communiquer avec ce serveur Radius par l'intermédiaire de son RadiusClient interne. Cette méthode utilise ***CDR (CALL Detail Record) pour la facturation*** et un client radius ***RadiusClient-ng pour l'authentification*** interne à Asterisk.

Voici le schéma descriptif de cette solution interne à Asterisk par le client RadiusClient-ng. Voir figure 4 en bas :

Les étapes 1à 6 pour l'authentification et la dernière pour la facturation.

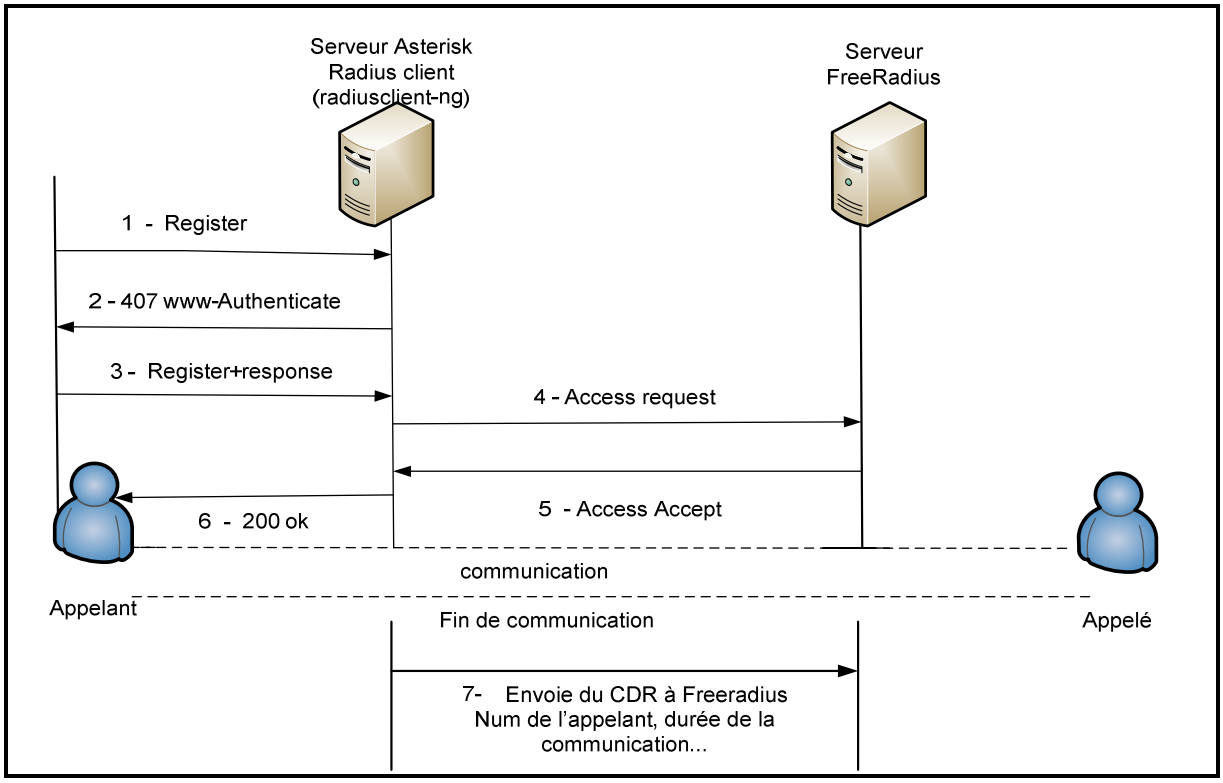


Figure 4: Solution interne à Asterisk (CDR et RadiusClient-ng)

5.2 Solution CDR (CALL Detail Record) par le client radiusclient-ng dans Asterisk

Cette section est prise de la source [8] en anglais et je l'ai traduit par moi-même, de la page 18 à 19 avec mes mots.

5.2.1 C'est quoi un CDR

Lorsqu'une connexion est établie, Asterisk effectue des enregistrements concernant la communication. Ces enregistrements sont appelés : ***CDR (Call Detail Record)***. On les trouve sous 2 formes différentes : soit un fichier texte en clair, soit dans une base de données (mysql, postgress…). Voici quelques données CDR enregistrées dans Asterisk :

- Le numéro de compte
- Le numéro source
- Le numéro de destination
- ID de l'appelant
- Le canal utilisé…
- Le canal de destination

5.2.2 Champs de CDR

- ***Auccountcode*** (numéro de compte à utiliser pour la facturation ou l'analyse, lien avec l'autocountcode de sip.conf)
- ***Src*** (numéro de l'appel)
- ***Dst*** (l'extension de la destination)
- ***Dcontext*** (contexte de destination du plan de numérotation utilisé)
- ***Clid*** (informations concernant l'appelant, caller ID)
- ***Channel*** (canal utilisé pour le routage de l'appel vers l'appelant)
- ***Dstchannel*** (canal utilisé pour le routage de l'appel vers l'appelé)
- ***Lastdata*** (paramétres utilisés par l'application)
- ***Start*** (heure d'appel)
- ***Answer*** (heure de la réponse de l'appelé)
- ***End*** (heure de fin de l'appel)
- ***Duration*** (durée de l'appel en secondes)
- ***Billsec*** (durée utilisée pour la facturation en secondes)
- ***Disposition*** (dernier statut de l'appel)
- ***User field*** (un champ défini par l'utilisateur, maximum 255 caractères)

5.2.3 Les principaux fonctions du CDR

- ***SetAccount (account)***: permet de définir un compte pour le CDR.
- ***SetAMAFlags (flags)*** : permet de positionner les flags AMA.
- ***NoCDR ()***: permet de désactiver le CDR pour l'appel en cours.
- ***ResetCDR()*** : permet de remettre le CDR à vierge.
- ***ForkCDR ()***: permet de créer un nouveau CDR pour l'appel en cours.
- ***Authenticate*** (): authentification et définir un compte pour le CDR.

5.2.4 Format du CDR dans Asterisk

La facturation de chaque appel effectué correspond à une ligne avec les différentes données enregistrées dans ce fichier, et séparées par des virgules. La figure 5 montre le format général du fichier Master.csv: [6]

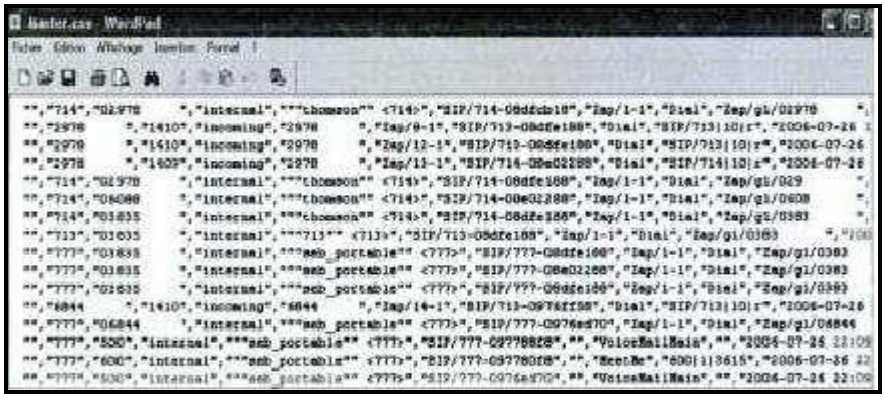


Figure 5: Format du fichier Master.csv du CDR

5.3 Solution radiusclient-ng dans Asterisk

Radiusclient-ng est une bibliothèque pour la rédaction d'un client Radius. Cette librairie est écrite entièrement en C. Il a été adopté par le projet FreeRadius.
Il contient un programme de ligne de commandes pour envoyer les enregistrements de facturations à Radius, et ainsi que des requêtes utiles pour interroger le statut du serveur Radius.

C'est une librairie téléchargeable, gratuite et portable (Linux, Solaris…). Ensuite, on doit configurer cette interface de communication entre Asterisk et FreeRadius.

5.4 Intégration de la solution CDR dans Asterisk et Radius

On va citer les principaux environnements nécessaires pour mettre en œuvre cette solution interne détaillé auparavant. Ceci nous amène à installer et configurer certains composants nécessaires au mécanisme d'authentification et de facturation via radius. Cette partie contient un petit guide sur les étapes de configuration de cette solution et l'explication de certains fichiers à configurer sous Asterisk et Radius.

Cette solution est tirée de la source [7].

5.4.1 Pré requis

- Serveur ***FreeRadius*** : est un serveur d'authentification.
- Librairie ***Radiusclient-ng*** : est une interface de communication entre les deux serveurs et elle permet d'envoyer les paramètres de facturation CDR à FreeRadius.
- Serveur ***Asterisk*** : est un PBX qui a le rôle d'un registre ou un relais.

5.4.2 Installation et configuration de Radiusclient-ng

Installation

- Télécharger la source depuis ce lien :

  http://developer.berlios.de/projects/radiusclient-ng/
- Décompresser la source

```
root@localhost:/usr/local/src# tar xvfz radiusclient-ng-0.5.2.tar.gz
```

- Compilation et installation de la librairie

```
root@localhost:/usr/local/src# cd radiusclient-ng-0.5.2
root@localhost:/usr/local/src/radiusclient-ng-0.5.2# ./configure
root@localhost:/usr/local/src/radiusclient-ng-0.5.2# make
root@localhost:/usr/local/src/radiusclient-ng-0.5.2# make install
```

Configuration

La configuration de la librairie ***radiusclient-ng*** est installée sur ce chemin ***usr/local/etc/radiusclient-ng***

- Fichier ***radiusclient.conf***

Il faut ouvrir ce fichier et chercher ces lignes qui contiennent :

- ✓ ***authserver localhost =>*** C'est le nom d'hôte ou l'adresse IP du serveur RADIUS utilisé pour l'authentification. Il faut changer, sauf si le serveur est en cours d'exécution sur la même machine que le PBX Asterisk.
- ✓ ***Acctserver localhost =>*** C'est le nom d'hôte ou l'adresse IP du serveur RADIUS utilisé pour comptabilité. Il faut changer, sauf si le serveur est en cours d'exécution sur la même machine que le PBX Asterisk.

- Fichier ***servers***

Protocole RADIUS utilise un mécanisme simple de contrôle d'accès basé sur le partage des secrets. Le serveur RADIUS permet de limiter l'accès aux clients RADIUS. Il utilise le principe de secret partagé pour pouvoir accepté que les clients radius ayant le même secret que lui.
On a besoin de configurer un secret partagé pour chaque serveur qu'on va utiliser.
Le partage des secrets sont stockés dans ce fichier ***/ usr / local / etc / radiusclient-ng / server***.
Chaque ligne contient le nom d'hôte d'un serveur RADIUS et son secret partagé utilisé pour identifier le client radius. Les deux valeurs sont séparées par des espaces.

- Fichier ***dictionary***

Asterisk utilise des attributs qui ne sont pas inclus dans le dictionnaire de radiusclient, donc il est nécessaire d'ajouter ces attributs d'asterisk. Il y a un fichier portant le nom ***dictionary.asterisk*** qui contient la liste des attributs utilisés par Asterisk. On doit l'inclure dans ce fichier dictionnaire du client ***/ usr / local / etc / radiusclient-ng / dictionnaire***.

```
$INCLUDE /path/to/dictionary.asterisk
```

5.4.3 Installation et configuration de FreeRadius

Installation

- Télécharger la source depuis ce lien :

http://freeradius.org/

- Décompresser la source

```
root@localhost:/usr/local/src# tar xvfz freeradius-1.1.1.tar.gz
```

- Compilation et installation de la librairie

```
root@localhost:/usr/local/src# cd freeradius-1.1.1
root@localhost"/usr/local/src/freeradius-1.1.1# ./configure
root@localhost"/usr/local/src/freeradius-1.1.1# make
root@localhost"/usr/local/src/freeradius-1.1.1# make install
```

- Toutes les fichiers de configuration de FreeRadius se trouvent dans ***usr/local/etc/raddb***.

Configuration

- Fichier ***clients.conf***

  Ce Fichier contient la description des clients radius qui sont autorisés à utiliser le serveur Radius. Pour chaque client, on doit indiquer son nom d'hôte ou l'adresse IP avec son secret partagé. Ce dernier doit être le même que celui défini dans le fichier de configuration du radiusclient -ng.
  Cela revient à déclarer quels matériels (NAS ou client Radius) peuvent soumettre des requêtes au serveur FreeRadius. Tout autre matériel sera refusé et le message suivant sera émis dans le fichier de journalisation : ***Ignoring request from unknown client adresse-ip-du-NAS***.

  Par Exemple pour chaque NAS, on utilise cette syntaxe:

```
client  adresse-ip {
        secret = monsecret
        shortname = toto
}
```

  - ✓ ***adresse-ip*** : est l'adresse IP du client Radius.
  - ✓ ***secret*** : est le secret partagé entre le serveur radius et le client NAS.
  - ✓ ***shortname*** : est un alias que l'on donne à ce client NAS.

  Le fichier contient déjà une entrée pour localhost (127.0.0.1) par défaut, donc si on veut exécuter le serveur RADIUS sur la même machine que notre serveur Asterisk, il faut modifier le point d'entrée en vigueur, en remplaçant le mot de passe par défaut.

- Fichier ***dictionnary***

  Le dictionnaire d'attributs est une simple table qui permet de faire la correspondance entre leur numéro et leur nom respectif. Cette table contient trois champs par attribut : le numéro de l'attribut, son nom et son type.

  Ce Fichier ***/ usr / local / etc / raddb / dictionnaire*** contient le dictionnaire du serveur FreeRadius. On doit ajouter le même fichier dictionnaire (***dictionary.asterisk***), qu'on a ajouté précédemment au dictionnaire de ***radiusclient-ng***. On peut l'inclure dans le fichier principal de son dictionnaire en ajoutant cette ligne suivante à la fin du fichier ***/ usr / local / etc / raddb / dictionnaire***:

  ```
  $ Include / chemin / vers / dictionary.asterisk
  ```

  => Il inclura les mêmes attributs définis dans le dictionnaire de ***radiusclient-ng*** de sorte que le client et le serveur communique en utilisant les mêmes attributs et arrive à se compredre.

- Fichier ***radiusd.conf***

  Ce fichier regroupe l'ensemble des paramètres nécessaires pour décrire le type de fonctions souhaitées. Il existe plusieurs d'options et de modules. On va citer les plus intéressants :

  - ✓ ***Paramétre du service Radiusd*** :

    On trouve ici les paramètres directement liés au fonctionnement de la base du processus Radiusd :

    - Déclaration des chemins d'accès où se trouvent les programmes (prefix), les fichiers de configuration (sysconfig).
    - Déclaration du port d'écoute : normalement 1812 ou 0 pour utiliser le port défini dans le fichier etc/services.
    - Déclaration du nombre minimal de processus lancés au démarrage et du nombre maximal possible.
    - Inclusion du fichier clients.conf.

  - ✓ ***Déclaration des modules :***

    Cette section commence par le mot-clé ***Modules*** et contient la déclaration de tous les modules qui seront utilisés pendant l'exploitation.

    ```
    Modules {
      Nom-module [nom-instance] {
        Config-item = valeur}
    }
    ```

Le nom du module correspond à un fichier nommé ***rlm_nom_module*** dans le répertoire ***/usr/local/lib***. Et entre l'accolade, on trouve un certain nombre d'options particulières à chaque module. Et dans radiusd.conf existe aussi un grand nombre de module prédéfinis.

✓ ***Section Authorize :***

Cette section contient la liste des modules qui doivent être exécutés afin de constituer la liste des ***reply-items*** et de préparer le terrain avant la section Authenticate.

✓ ***Scetion Authenticate***

On trouve ici la liste des modules correspondant aux méthodes d'authentification que le serveur accepte. Ces modules sont les mêmes que ceux qui ont pu être utilisés dans la section Authorize, mais ici, c'est la tâche d'authentification est accomplie. Ces déclarations des modules constituent la liste des protocoles que reconnaît le serveur.

Après le petit résumé du contenu de ce fichier, la configuration à faire dans celui-ci, est d'activer ***l'authentification Digest*** qui est par défaut désactivée. Donc il ya deux sections, "***Authorize***" et "***Authenticate***". Les deux sections contiennent une ligne contenant le mot «***digest***». On doit décommenter pour avoir la permission d'utiliser l'authentification Digest.

- Fichier ***users***

Le fichier users est la base de données locale. Il est utilisé soit comme base d'autorisation, soit comme base d'authentification ou les deux à la fois. Il s'agit d'un simple fichier texte. Il faut connaître le format de ce fichier, donc il est constitué d'une liste d'entrées, chacune correspondant à un utilisateur ou à une machine. Le format de ces entrées est :

Identifiant <config-items>, <check-items>,<check-items>,…<check-items>
Reply-item,…

Chaque entrée est composé de deux parties : ***Identifiant*** est l'identité véhiculé par l'attribut User-Name dans un paquet ***Acess-Request***. FreeRadius balaye le fichier avec cette identité comme clé de recherche. ***Les config-items*** sont toujours écrits sur la première ligne et sont séparés par une virgule. ***Les check-items*** sont eux aussi toujours sur la première ligne, séparés par une virgule et la ligne se termine sans virgule. Voilà la structure générale :

Identifiant config-item check-item

Test ***Auth-Type: = Digest,*** ***User-Password == "test"***

Reply-Message = "Hello world" reply-item

Test du serveur FreeRadius

La configuration de base de notre serveur FreeRadius est faite. Il faut aller sur le chemin d'installation du serveur (***"/ usr / local / src/***), et lancer son démarrage par cette commande :

```
Root @ / usr / local / src # radiusd-X
```

Pour gérer les demandes d'authentification de type ***Digest*** dans FreeRadius, on a besoin de décommenter Digest dans les 2 sections "***Authorize***" et " ***Authenticate*** ". Créer un fichier digest et mettre ce texte suivant sur une seule ligne:

```
User-Name="test",
Digest-Response = "631d6d73147add2f9e437f59bbc3aeb7",
Digest-Realm = "testrealm", Digest-Nonce = "1234abcd»,
Digest-Method = "INVITE", Digest-URI ="sip:5555551212@example.com",
Digest-Algorithm = "MD5", Digest-User-Name = "test", Message-Authenticator=""
```

Pour exécuter ***radclient*** et tester le serveur:

```
root@/usr/local/src# radclient -f digest localhost auth <shared_secret>
```

On doit autoriser l'accès au client localhost dans le fichier ***clients.conf***. Et de ne pas oublier de remplacer ***<shared_secret>*** avec le secret partagé défini pour le client localhost dans ce fichier client.conf.

Si le serveur fonctionne correctement, on doit avoir cette réponse d'après la configuration faite ci-dessus:

```
Reçu réponse ID 224, code 2, longueur = 45
Reply-message = "Hello world"
```

5.4.4 *Installation et configuration d'Asterisk*

Installation

- Télécharger la dernière version Asterisk à partir de ce lien :

  http://ftp.digium.com/pub/asterisk

- Déplacer et compiler les sources dans le répertoire ***/usr/src/***

```
cd /usr/src/
tar zxvf /asterisk-1.2.0.tar.gz
chown –R root :root asterisk-1.2.0
cd usr/src/asterisk-1.2.0
```

- Compilation d'Asterisk

```
make clean
make
make mpg123
make samples
make config
```

make mpg 123 => elle permet à Asterisk d'utiliser le logiciel mpg123 pour la diffusion de fichiers au format MP3. Cela est utile lors de la diffusion de musique d'attente.

make samples => permet l'installation des fichiers de configuration par défaut comme ***/etc/asterisk/sip.conf***, ***/etc/aterisk/extensions.conf***…

- Démarrer le serveur Asterisk pour le tester :

```
Cd /usr/sbin/asterisk/ asterisk –vvvc
Cd /etc/init.d/asterisk start
```

Configuration

Asterisk enregistre automatiquement et conserve les renseignements CDR de nos appels (identité des interlocuteurs, la durée de la conversation, …). Il crée un fichier CSV et il se trouve dans le répertoire ***/var/log/asterisk/cdr-csv***.

Pour activer les demandes d'enregistrement d'Asterisk via FreeRadius, on doit charger le module ***cdr_radius.so***. Pour ce faire, il faut l'inclure en ajoutant cette ligne dans le fichier ***etc/asterisk/ modules.conf : load =>cdr_radius.so***. Il y a une autre méthode pour ajouter ce module, c'est de mettre le fichier cdr_radius.so dans ce chemin ***/usr/lib/asterisk /modules*** car Asterisk stocke tous les modules dans ce répertoire. On redémarre le serveur Asterisk pour qu'il prenne en compte.

Par défaut, le serveur FreeRadius enregistre toutes les demandes de facturation dans ce répertoire ***/ usr / local / var / log / radius / radacct*** sous forme de fichiers texte brut. Et le serveur va créer un fichier dynamiquement pour chaque nom d'hôte faisant office d'un client radius dans ce répertoire.

6 REALISATION DE L'AUTHENTIFICATION/ACCOUNTING A L'AIDE DE RADIUSCLIENT-NG

A ce stade, on va essayer d'implémenter une infrastructure de test utilusant ce radiusclient-ng pour authentifier et enregistrer les CDRs via FreeRadius.Pour ce faire, on va simuler cette solution d'authentification et de facturation sur des machines virtuelles avant de l'appliquer dans l'infrastructure VOIP de la société Switzernet.

Le principe de cette manipulation est d'installé un serveur ***Vmware*** sur une machine de l'école. Ce serveur permet de créer des machines virtuelles Linux ou Windows ou Mac… Dans notre projet, on a besoin d'utilisé des machines Linux. Un post Asterisk et un post Radius qui communique entre eux à l'aide d'une interface, qui est le NAS (client Radius). Et pour les tests, on aura aussi besoin de deux téléphones SIP qui essayeront de s'enregistrer sur le serveur Asterisk, ensuite ils s'authentifieront via le serveur Radius pour pouvoir émettre des appels téléphoniques entre eux.

Premièrement, il faut installer un serveur Vmware depuis ce lien http://www.vmware.com/download/server/.
Ici, on ne va pas parler de l'installation de ce serveur car ce n'est pas le but du projet. Mais l'installation est facile, elle se fait en quelques minutes et il faut juste s'enregistrer pour avoir un numéro de série.

Deuxièmement, il reste les machines Linux. Le choix de distribution de la machine linux est fait sur la ***Fedora Core 9***. Pour ce faire, on doit juste téléchargé cette machine à partir de ce lien http://www.vmware.com/appliances/directory/1220 et ouvrir un fichier .vmx dans le Vmware pour avoir une machine Fedora sur notre poste.

Maintenant, l'implémentation de l'infrastructure peut commencer en installant et configurant le serveur Asterisk et FreeRadius sur une machine différente Fedora. Cette partie mérite d'être détaillée par la suite de façon claire dans notre rapport et de donner brièvement une démonstration de leurs fonctionnements corrects.

6.1 Installation et configuration d'Asterisk sur Fedora 9

Avant de commencer l'installation d'Asterisk, on aura besoin d'installer certains environnements et librairies utiles pour son fonctionnement de façon correcte. Donc voici l'énumération des environnements installés avant le PBX.

Sous Fedora, on peut avoir recours à l'utilisation d'une interface graphique pour installer des environnements par un simple clic. Cette interface est ***Yumex*** mais il existe bien d'autre. Celle-ci s'installe juste en exécutant cette commande ***yum install yumex*** dans le Shell.

La procédure de l'installation et de configuration des environnements indispensables à Asterisk est tirée depuis le livre en anglais [6] et de la source [12].

Installation de PERL

L'installation de perl est faite par yumex et le module CPAN comme le montre la figure 6 en bas :

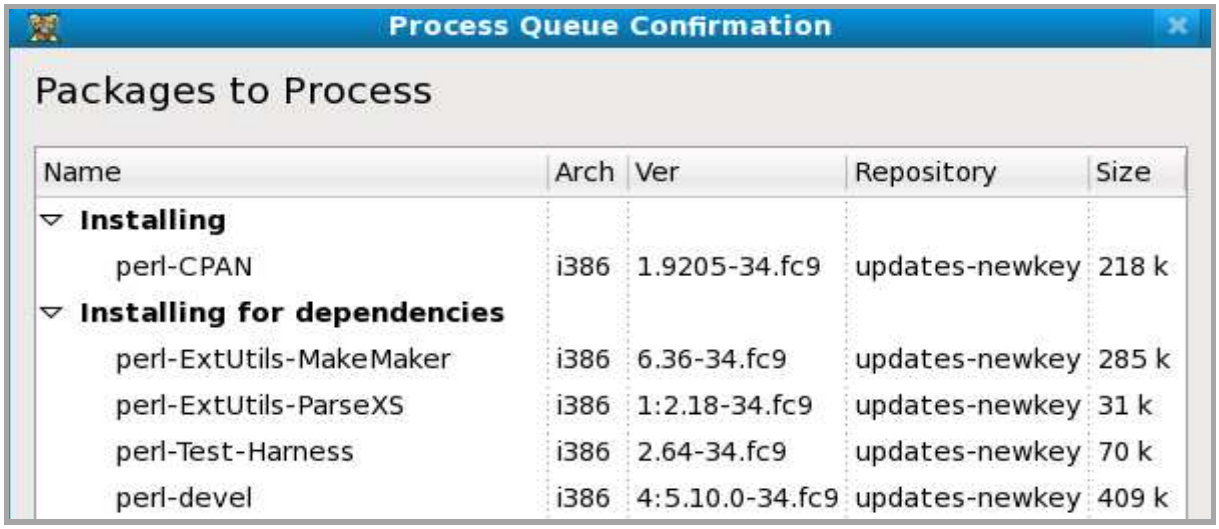


Figure 6: Installation de perl-CPAN et ses dépendances

Il faut ouvrir un Shell et taper cette commande :

- ✓ ***# perl – MCPAN –e shell***

Avant son utilisation, on doit configurer MCPAN. Donc il faut répondre par No pour appliquer la configuration par défaut, sinon on doit configurer tout par nous-mêmes et ça prend beaucoup du temps, ce n'est pas le but dans notre projet.

Quand tout est installé correctement, on ouvre la console de CPAN pour qu'on puisse installer ses modules nécessaires pour le fonctionnement d'Asterisk :

```
cpan> install Bundle ::CPAN
cpan> install Net ::Telnet
cpan> install Proc ::WaitStat
cpan> install IPC ::Signal
cpan> install IO::Socket
cpan> install Getopt::Long
cpan> install Pod::Usage
```

```
cpan[4]> install IPC::Signal
Running install for module 'IPC::Signal'
Running make for R/RO/ROSCH/IPC-Signal-1.00.tar.gz
Fetching with LWP:
  http://www.perl.org/CPAN/authors/id/R/RO/ROSCH/IPC-Signal-1.00.tar.gz
Fetching with LWP:
  http://www.perl.org/CPAN/authors/id/R/RO/ROSCH/CHECKSUMS
Checksum for /root/.cpan/sources/authors/id/R/RO/ROSCH/IPC-Signal-1.00.tar.gz ok
```

```
cpan[7]> install IO::Socket
IO::Socket is up to date (1.30_01).

cpan[8]> install Getopt::Long
Getopt::Long is up to date (2.37).

cpan[9]> install Pod::Usage
Pod::Usage is up to date (1.35).
```

Installation du compilateur gcc

Asterisk a besoin que ces paquets soit installés : ***gcc, glibc-kern headers, cpp, binutils, glibc-headers, glibc-devel***. Si ces librairies manquent donc lors de la compilation d'Asterisk génère des erreurs. Voir figure 7 en bas :

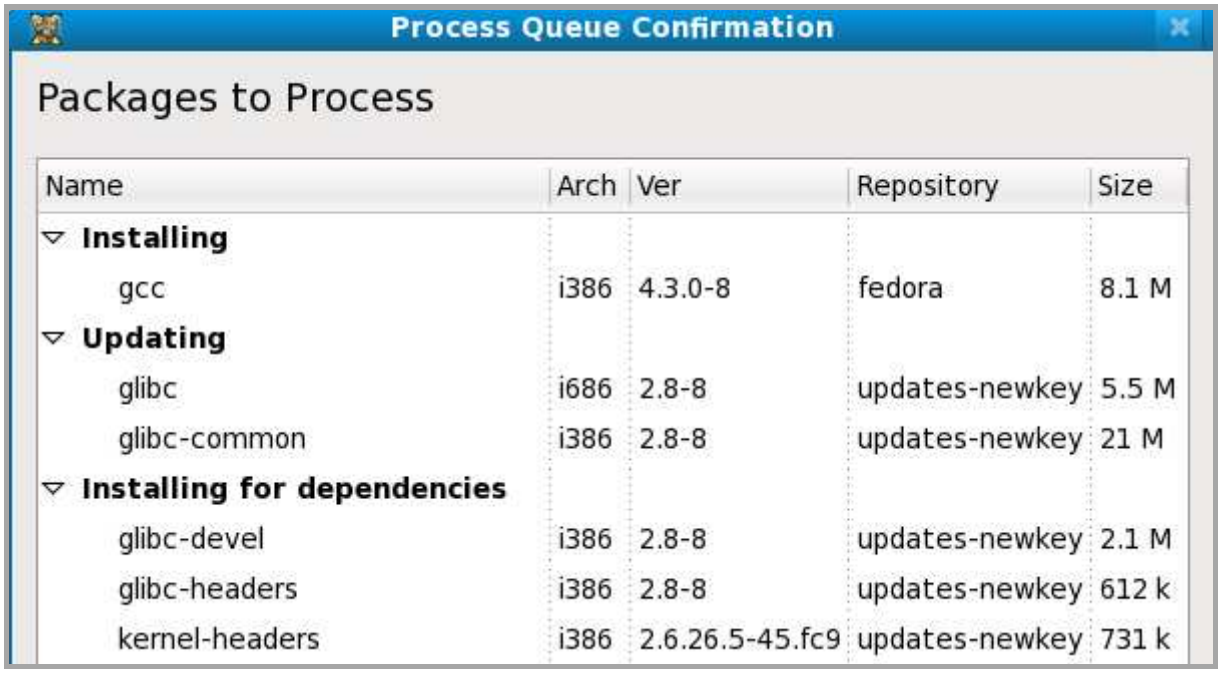


Figure 7: Installation du compilateur GCC et ses dépendances

Installation de bison et m4
Ils sont utiles pour l'installation d'Asterisk.

Installation de la librairie openssl et ses dépendances
Asterisk a besoin de ces librairies : ***openssl-devel, e2fsprogs-devel, zlib-devel, krb5-devel, krb5-libs***.

Installation de la librairie ncurse
Il faut installer la librairie ***ncurse***, ***ncurse-devel***.

Installation de la librairie valgrind
Il faut installer ***valgrind et valgrind-devel***.

Installation de MYSQL

L'installation de MYSQL n'est pas obligatoire mais elle est utile, par exemple si on veut enregistrer les champs de CDR dans une table MySQL au lieu du format texte d'Asterisk. L'installation de MYSQL se fait en tapant cette ligne de commande :

```
[root@localhost admin]# yum install mysql mysql-server
Loaded plugins: refresh-packagekit
updates-newkey                                           | 2.3 kB     00:00
fedora                                                   | 2.4 kB     00:00
updates                                                  | 2.6 kB     00:00
Setting up Install Process
```

L'avantage de l'installateur yum est de détecter automatiquement les librairies nécessaires au fonctionnement correct de MYSQL. La figure8 en bas, nous donne une idée de ces librairies :

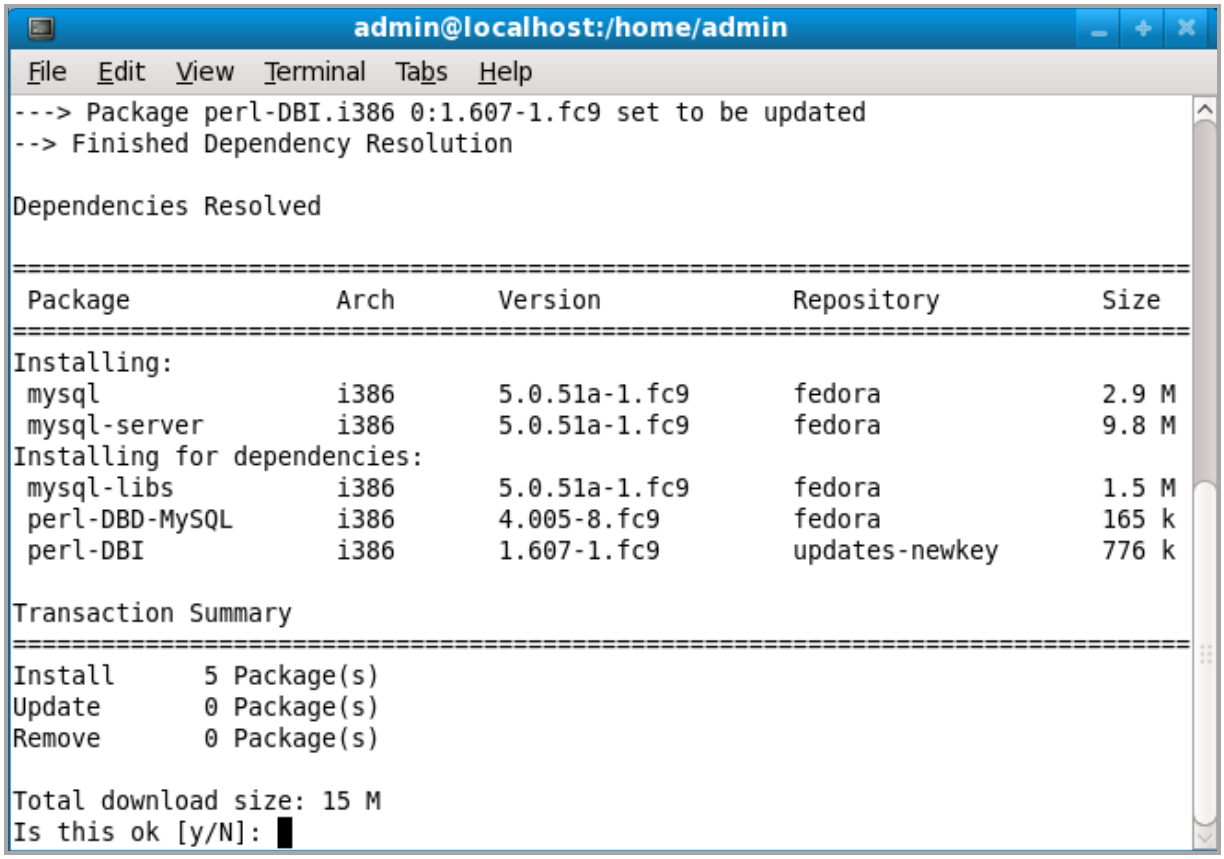


Figure 8: Installation de MYSQL et ses dépendances

Il suffit de répondre par oui et l'installation se fait automatiquement.

6.1.1 *Installation d'Asterisk*

L'installation d'Asterisk a été faite en téléchargeant le fichier compressé selon notre distribution linux. Ensuite, on décompresse ce fichier d'extension tar.gz et on le met dans un endroit quelconque de notre système, par exemple ***/usr/src***. Le lien de téléchargement est : http://downloads.digium.com/pub/asterisk/releases/

Le téléchargement et le placement des fichiers d'Asterisk décompressés se trouvent dans le chemin spécifié en haut, on tape ces commandes :

```
root@/usr/src# cd asterisk-1.4.21.2
make clean
./configure
make
make install
make samples
```

L'installation d'Asterisk est complète comme le montre la figure 9. Il reste plus qu'à le configurer et tester son fonctionnement.

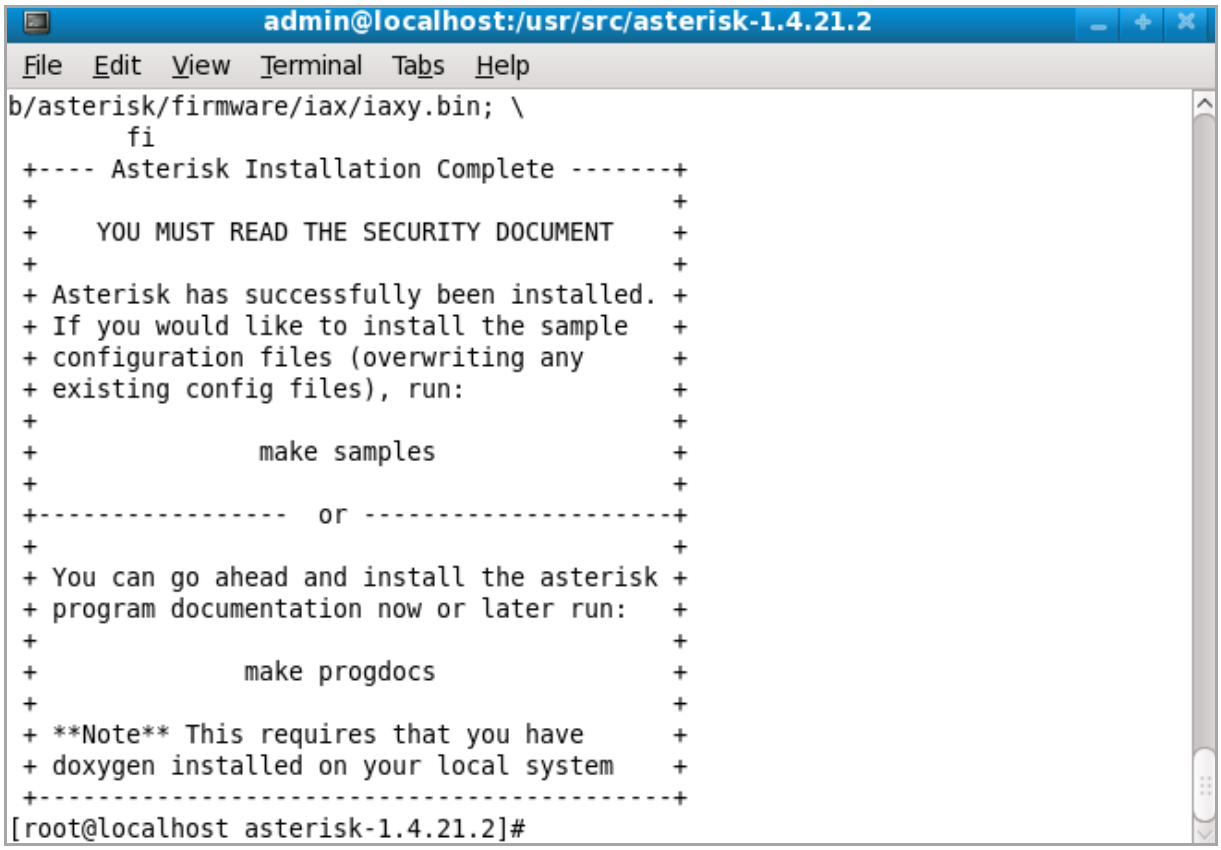


Figure 9: Installation d'Asterisk complète

Voici l'architecture des fichiers installés d'Asterisk sur notre système avec une brève explication de la fonction de chacun d'eux :

- ***etc/asterisk*** : contient tous les fichiers nécessaires à le configurer.
- ***etc/asterisk/sip.conf*** : ici, on définit les options du protocole SIP à Asterisk. Ce dernier contient l'identification des entités (utilisateur, secret, numéro téléphone…).
- ***etc/asterisk/extensions.conf*** : constitue le moyen par lequel, on indique à Asterisk comment gérer les appels. Donc on définit le plan de numérotation.
- ***etc/asterisk/voicemail.conf*** : contrôle le système de boîte vocale d'Asterisk.
- ***etc/asterisk/cdr.conf*** : sert à rendre possible d'enregistrer les détails d'un appel(CDR). Permet d'activer le système de facturation dans Asterisk.
- ***usr/lib/asterisk/modules*** : contient tous les modules chargeables d'Asterisk. Par défaut, Asterisk charge tous ces modules lors de son démarrage.
- ***/var/lib/asterisk*** : contient le fichier ***astdb*** et des sous-répertoires. Cet astdb contient la base de données d'informations locale d'Asterisk. Les sous –répertoires sont ***agi-bin/*** contient des scripts qui s'interfacent avec Asterisk par les divers applications AGI intégrées. ***Firmware/*** contient les firmwares de divers appareils compatibles Asterisk. ***Keys/*** peut contenir les clés privées/publiques pour authentifier les utilisateurs. ***Sounds/*** contient les annonces vocales pour Asterisk. Les contenus des annonces de base incluses avec Asterisk sont dans le fichier ***sound.txt***.
- ***Var/log/asterisk*** : c'est dans cet endroit, Asterisk enregistre les erreurs concernant son fonctionnement dans le fichier messages et d'autres fichiers de bugs.
- ***Var/log/asterisk/cdr_csv*** : c'est ici, on trouve le stockage des CDR dans un fichier de format texte portant le nom ***Master.csv***.

6.1.2 Configuration d'Asterisk

Cette configuration nécessite l'intégration de 3 modules importants à Asterisk après son installation. L'ajout de ces modules doit se faire impérativement après son installation. Ces modules sont : module ***perl***, module ***addons*** et module ***sounds***.

- ***Module Asterisk-perl*** : est une bibliothèque qui permet de développer des applications de type AGI et Manager.

- ***Module Asterisk-sounds*** : permet d'utiliser des annonces.

- ***Module Asterisk-addons*** : permet de stocker les enregistrements d'accounting CDR dans une base de données MYSQL.

6.1.2.1 Module asterisk- perl-0.10

On peut télécharger la source depuis ce lien : http://asterisk.gnuinter.net/files/asterisk.perl-0.10.tar.gz

```
root@/usr/src# tar zxvf root/download/asterisk-perl-0.10.tar.gz
cd asterisk-perl-0.10
perl Makefile.PL
make all
make install
```

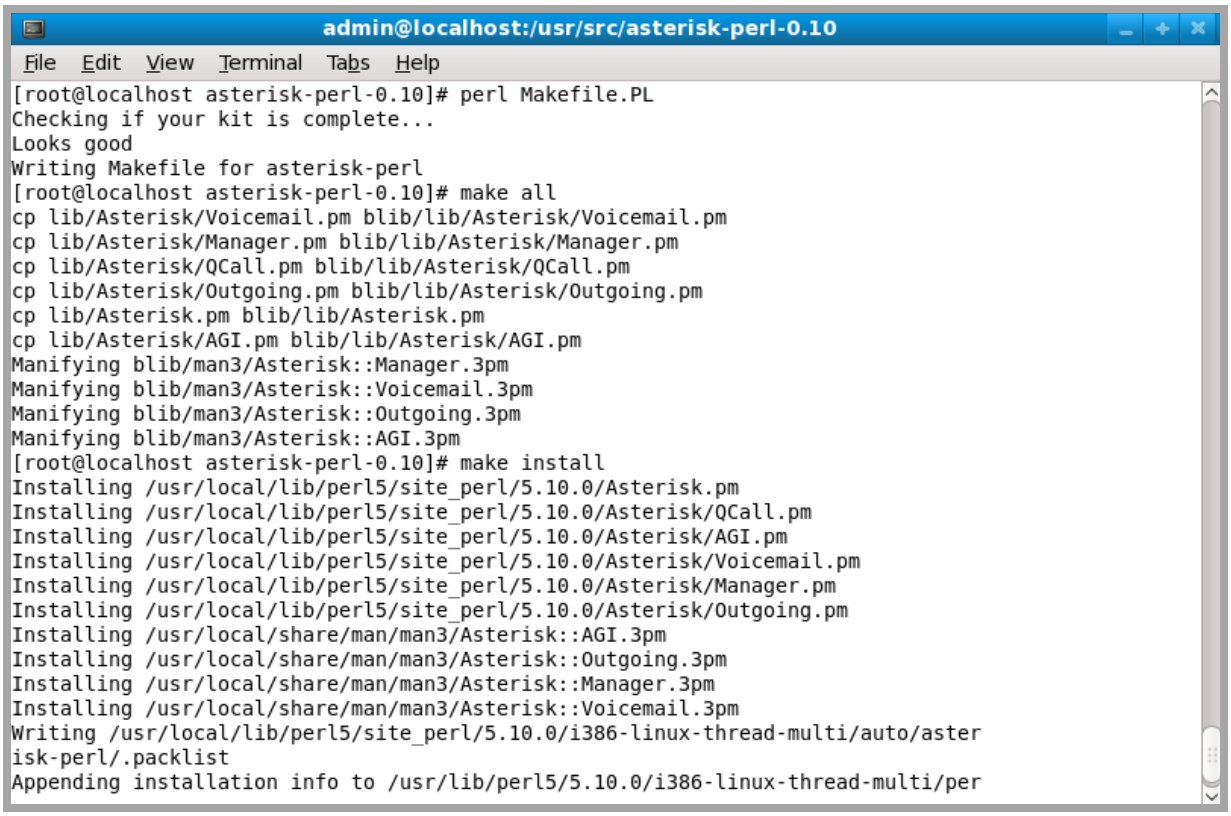


Figure 10: Installation du module asterisk-perl-0.10

6.1.2.2 Module asterisk- sounds-1.2.1

La source se trouve depuis ce lien : http://ftp.digium.com/pub/releases/asterisk-sounds-1.2.1

```
root@/usr/src# tar zxvf root/download/asterisk-sounds-1.2.1
cd asterisk-sounds-1.2.1
make install
```

```
[root@localhost src]# ls
asterisk-1.4.21.2             asterisk-1.4.21.2.tar.gz  asterisk-sounds-1.2.1
asterisk-1.4.21.2-1.fc8.i386  asterisk-addons-1.4.2
[root@localhost src]# cd  asterisk-sounds-1.2.1
[root@localhost asterisk-sounds-1.2.1]# make install
mkdir -p /var/lib/asterisk/sounds
mkdir -p /var/lib/asterisk/sounds/letters
mkdir -p /var/lib/asterisk/sounds/phonetic
mkdir -p /var/lib/asterisk/sounds/silence
mkdir -p /var/lib/asterisk/sounds/wx
mkdir -p /var/lib/asterisk/sounds/ha
for x in sounds/letters/*.gsm; do \
                if grep -q "^%`basename $x`%" sounds-extra.txt; then \
                        install -m 644 $x /var/lib/asterisk/sounds/letters ; \
                else \
                        echo "No description for $x"; \
                        exit 1; \
                fi; \
```

Figure 11: Installation du module asterisk-sounds-1.2.1

6.1.2.3 Module asterisk- addons-.1.4.2

La source se trouve depuis ce lien : http://ftp.digium.com/pub/releases/asterisk-addons-1.4.12

root@/usr/src# tar zxvf root/download/asterisk-addons-1.4.2
cd asterisk-addons-1.4.2
make install

```
make[2]: Nothing to be done for `all'.
make[2]: Leaving directory `/usr/src/asterisk-addons-1.4.2/format_mp3'
make[1]: Leaving directory `/usr/src/asterisk-addons-1.4.2'
for x in app_addon_sql_mysql.so app_saycountpl.so cdr_addon_mysql.so res_config_
mysql.so; do /usr/bin/install -c -m 755 $x /usr/lib/asterisk/modules ; done
make[1]: Entering directory `/usr/src/asterisk-addons-1.4.2/format_mp3'
/usr/bin/install -c -m 755 format_mp3.so /usr/lib/asterisk/modules
make[1]: Leaving directory `/usr/src/asterisk-addons-1.4.2/format_mp3'
make[1]: Entering directory `/usr/src/asterisk-addons-1.4.2/asterisk-ooh323c'
cp .libs/libchan_h323.so.1.0.1 /usr/lib/asterisk/modules/chan_ooh323.so
make[1]: Leaving directory `/usr/src/asterisk-addons-1.4.2/asterisk-ooh323c'
 +---- Asterisk-Addons Installation Complete ----+
 +                                               +
 +    Addons has successfully been installed.    +
 +    If you would like to install the sample    +
 +    configuration files (overwriting any       +
 +    existing config files), run:               +
 +                                               +
 +                make samples                   +
 +-----------------------------------------------+
```

Figure 12: Installation du module asterisk-addons-1.4.2

6.1.3 Test du fonctionnement d'Asterisk

En premier, nous allons démarrer le serveur Asterisk en ligne de commande. On ouvre un Shell et on tape cette commande dans ce chemin :

```
# cd etc/rc.d/init.d/asterisk start
```

Et pour le stopper, on met juste ***asterisk stop***.

Le binaire d'Asterisk est installé par défaut dans ***/usr/sbin/asterisk***. On va lancer la console ***CLI*** d'Asterisk qui permet d'interroger notre serveur et de voir les messages d'erreurs d'Asterisk. Le binaire d'Asterisk se lance avec différentes options possibles. Comme le serveur est lancé donc il est conseillé d'appeler la console CLI en tapant ***asterisk –r***. Voir figure 13 ci-dessous :

```
[root@localhost ~]# asterisk -r
Asterisk 1.4.21.2, Copyright (C) 1999 - 2008 Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for detail
s.
This is free software, with components licensed under the GNU General Public
License version 2 and other licenses; you are welcome to redistribute it under
certain conditions. Type 'core show license' for details.
=========================================================================
Connected to Asterisk 1.4.21.2 currently running on localhost (pid = 2018)
Verbosity is at least 3
localhost*CLI>
```

Figure 13: Démarrage d'Asterisk en mode console

Le dernier test du serveur se fait en changement le fichier de configuration ***sip.conf***. Cette modification consiste à ajouter un utilisateur de type SIP. Donc on définit un nom d'utilisateur, un numéro de téléphone, un mot de passe…Voir figure 14 ci-dessous:

sip.conf

```
[user1]

type=friend
username=user1
secret=password
quality=yes
;regexten=1234                  ; When they register, create extension 1234
callerid="user1" <1234>
host=dynamic                    ; This device needs to register
nat=no                          ; X-Lite is behind a NAT router
canreinvite=no
dtfmode=rfc2833                 ; Typically set to NO if behind NAT
context=internal
;disallow=all
;allow=gsm                      ; GSM consumes far less bandwidth than ulaw
allow=ulaw
;allow=alaw
;mailbox=1234@default,1233@default      ; Subscribe to status of multiple mailboxes
```

Figure 14: Comment définir un utilisateur SIP sous Asterisk

Par la suite, on envoie une requête au serveur Asterisk en tapant cette commande CLI :

- ✓ ***sip show users*** : pour voir tous les utilisateurs ajoutés.
- ✓ ***sip show users user1*** : pour voir l'utilisateur SIP user1 enregistré dans le serveur.

Donc si le serveur fonctionne correctement, on doit voir l'affichage des données des utilisateurs SIP définies dans sip.conf dans la console CLI. Voir figure 15 ci-dessous :

```
localhost*CLI> sip show user user1
localhost*CLI>

  * Name       : user1
  Secret       : <Set>
  MD5Secret    : <Not set>
  Context      : internal
  Language     :
  AMA flags    : Unknown
  Transfer mode: open
  MaxCallBR    : 384 kbps
  CallingPres  : Presentation Allowed, Not Screened
  Call limit   : 0
  Callgroup    :
  Pickupgroup  :
  Callerid     : "user1" <1234>
  ACL          : No
  Codec Order  : (ulaw:20)
  Auto-Framing:  No

localhost*CLI>
```

Figure 15: Affichage de la sortie des données SIP dans la console CLI

6.2 Installation et configuration de FreeRadius sur Fedora 9

6.2.1 *Installation de FreeRadius*

Pour l'installer, on a téléchargé le fichier compressé de FreeRadius à partir de cette source : http://freeradius.org/download.html. Le choix de la version est fait sur ***FreeRadius-1.1.7.tar.gz***. Comme d'habitude, on décompresse ce fichier téléchargé et on tape ces commandes [13]:

```
root@/usr/src# cd freeradius-1.1.7
make clean
./configure
make
make install
```

6.2.2 *Configuration de FreeRadius*

Les fichiers de configuration du serveur se trouvent dans le répertoire ***usr/local/etc/raddb***. On va citer seulement les fichiers les plus importants, et surtout ceux qu'on va utiliser dans notre projet.

- ***users.conf*** : constitue une liste d'entrées, chacune correspondant à un utilisateur (nom d'utilisateur, mot de passe…). C'est une base de données locale pour l'authentification et l'autorisation.
- ***radiusd.conf*** : regroupe les paramètres globaux et les fonctions configurées par l'utilisateur comme l'authentification, autorisation et facturation dans FreeRadius.
- ***clients.conf*** : a le rôle de définir les secrets partagés avec chaque équipement réseau. Ceci veut dire de déclarer les NAS qui peuvent communiquer avec le serveur FreeRadius.
- ***naslist.conf*** : on met la liste des clients radius NAS où on spécifie l'adresse IP du NAS, le type d'équipement.
- ***sql.conf*** : est utile si on veut authentifier ou autoriser les utilisateurs, qui sont seulement enregistrées dans une base de données MYSQL.

6.2.3 *Test du fonctionnement FreeRadius*

Pour le tester, on doit lancer le service radius par l'interface des services de Fedora ou par la ligne de commande. On va lancer le serveur en ***mode debug*** pour qu'on puisse voir la configuration faite dans notre serveur FreeRadius. Ce mode permet l'affichage des erreurs lors de la présence d'un problème et une indication sur l'origine du bug. La commande pour le mode debug est : ***radiusd –X***.

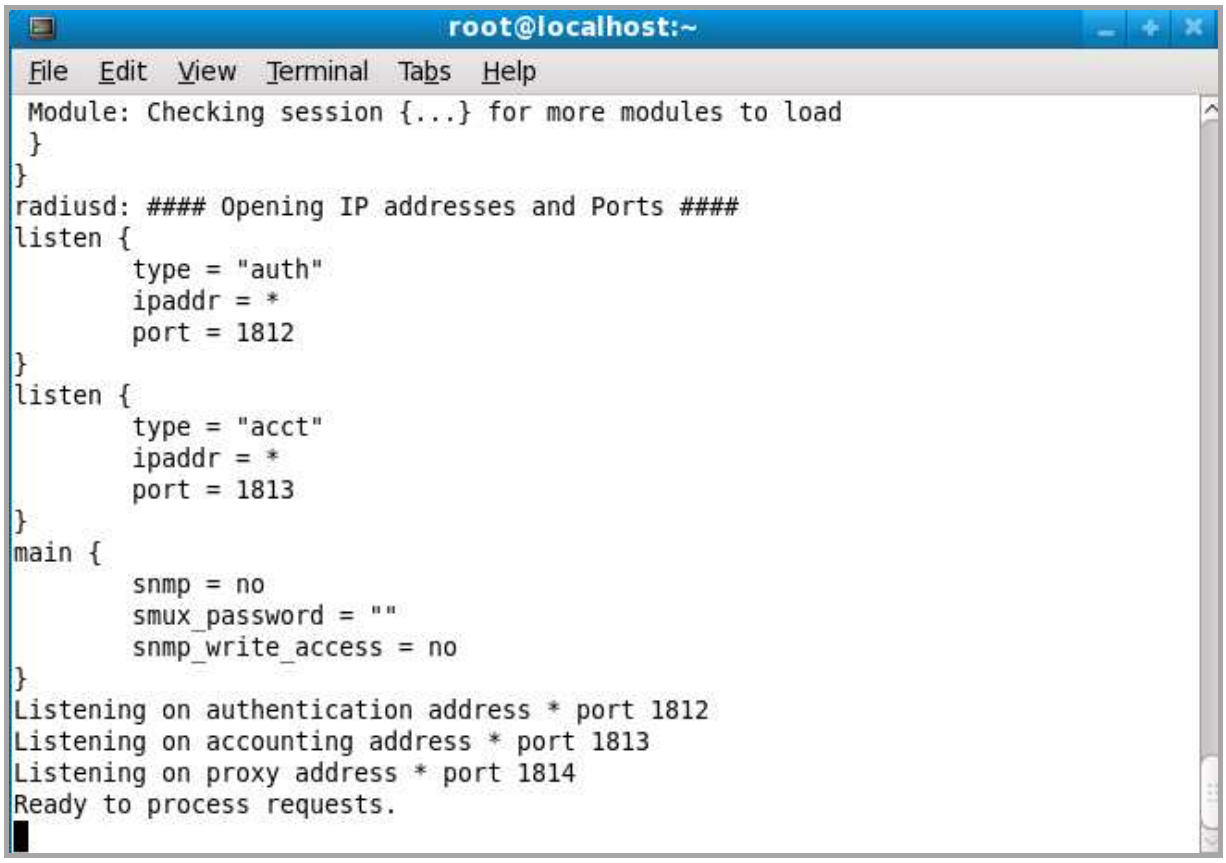


Figure 16: Serveur FreeRadius démarré

D'après la figure 16 ci-dessus, le serveur est fonctionnel et il est en train d'écouter sur les bons ports ***1812*** pour ***authentification*** et ***1813*** pour ***l'accounting***.

Si on veut le tester correctement, on doit éditer le fichier ***users.conf*** et ajouter un utilisateur avec un mot de passe. Il faut mettre ce nouvel utilisateur suivant cette structure :

Selim	***Cleartext-Password := "test"*** ***Reply-Message := "Compte, OK !!!! "***

Ensuite, on redémarre le serveur pour qu'il prenne en considération ce changement. On ouvre un autre Shell à part et on tape cette commande en respectant la structure décrite en bas :

 # radtest user password IP du serveur freeradius :port port nas secret de freeraduis

Si le serveur fonctionne correctement, nous devrions recevoir un paquet de type ***Acces-Accept***. Si cette authentification de cet utilisateur est acceptée par FreeRadius. Ce dernier affichera une réponse positive avec un message du replay défini dans le fichier users.conf. Voir figure 17 ci-dessous :

- ✓ ***radtest selim selim localhost :1812 0 testing123***

```
[root@localhost etc]# radtest selim selim 127.0.0.1:1812 0 testing123
Sending Access-Request of id 165 to 127.0.0.1 port 1812
        User-Name = "selim"
        User-Password = "selim"
        NAS-IP-Address = 127.0.0.1
        NAS-Port = 0
rad_recv: Access-Accept packet from host 127.0.0.1 port 1812, id=165, length=37
        Reply-Message = "compte ok!!!!!!"
[root@localhost etc]#
```

Figure 17: Test du FreeRadius avec la commande radtest

6.3 Test d'enregistrement de deux utilisateurs SIP sous Asterisk

Pour enregistrer deux clients SIP dans Asterisk, on doit éditer le fichier ***sip.conf*** comme c'est cité au paravent dans la section test du serveur. Ceci dans le but de définir 2 clients selon les attributs SIP conventionnels d'Asterisk.

6.3.1 Structure général du fichier sip.conf

Voici la structure générale de notre fichier sip.conf. Voir la figure 18 ci-dessous :

```
[general]
context=default                 ; Default context for incoming calls
;allowguest=no                  ; Allow or reject guest calls (default is yes)
allowoverlap=no                 ; Disable overlap dialing support. (Default is yes)
;allowtransfer=no               ; Disable all transfers (unless enabled in peers or users)
                                ; Default is enabled
;realm=mydomain.tld             ; Realm for digest authentication
                                ; defaults to "asterisk". If you set a system name in
                                ; asterisk.conf, it defaults to that system name
                                ; Realms MUST be globally unique according to RFC 3261
                                ; Set this to your host name or domain name
bindport=5060                   ; UDP Port to bind to (SIP standard port is 5060)
                                ; bindport is the local UDP port that Asterisk will listen on
bindaddr=0.0.0.0                ; IP address to bind to (0.0.0.0 binds to all)
srvlookup=no                    ; Enable DNS SRV lookups on outbound calls
                                ; Note: Asterisk only uses the first host
                                ; in SRV records
                                ; Disabling DNS SRV lookups disables the
                                ; ability to place SIP calls based on domain
                                ; names to some other SIP users on the Internet
```

Figure 18: Format général de sip.conf

Le seule changement est de mettre le ***srvlookup=no*** pour supprimer la possibilité de faire des appels SIP basés sur les noms de domaine. Et toutes les autres options, nous avons laissés leurs valeurs par défaut.

Ensuite, l'enregistrement des deux clients SIP se fait comme suit dans ***sip.conf*** :

[user1]
Type = friend ou peer ou user

- ✓ ***User*** : authentifier les appels entrants.
- ✓ ***Peer*** : authentifier les appels sortants.
- ✓ ***Friend*** : authentifier les appels entrants et sortants.

Secret = "mot de passe" : est un mot de passe utilisé pour l'authentification
Nat = no ou yes : si l'extension est derrière un appareil faisant du NAT(Network Adress translation).
Host = dynamic : demande à l'extension de s'enregistre pour qu'Asterisk sache comment joindre le téléphone.
Context=internal : on spécifie l'emplacement des instructions utilisées pour contrôler ce que le téléphone est autorisé à faire et de quoi faire des appels à destination de cette extension.
Canreinvite = no or yes : Asterisk tente ou pas de rediriger les appels

Voici comment sont enregistrés nos deux utilisateurs SIP dans notre fichier ***sip.conf*** sur le serveur :

```
[user1]

type=friend
username=user1
;secret=password
quality=yes
;regexten=1234                    ;
callerid="user1" <4321>
host=dynamic
auth=md5
auth_type=radius
nat=yes                           ;
canreinvite=no
dtfmode=rfc2833                   ;
context=internal
disallow=all
allow=gsm                         ;
allow=ulaw
allow=alaw
;mailbox=1234@default,1233@default
```

```
[user2]

type=friend
username=user2
;secret=test
quality=yes
;regexten=1234                    ;
callerid="user2" <5678>
host=dynamic                      ;
nat=yes                           ;
auth=md5
auth_type=radius
canreinvite=no
dtfmode=rfc2833                   ;
context=internal
;disallow=all
allow=gsm                         ;
allow=ulaw
allow=alaw
;mailbox=5678@default,1233@default
```

Pour tester si ces deux utilisateurs ont été pris en compte et enregistrés dans notre serveur Asterisk, on démarre le serveur et la console CLI par la commande ***asterisk –r*** et on obtient ceci :

```
=========================================================================
Connected to Asterisk 1.4.21.2 currently running on localhost (pid = 21583)
Verbosity is at least 3
localhost*CLI> sip show users
Username                  Secret           Accountcode      Def.Context      AC
L  NAT
user2                     test                              internal         No
   RFC3581
user1                     password                          internal         No
   RFC3581
localhost*CLI>
```

On voit bien que ces deux clients ont été ajoutés avec leurs options comme le mot de passe, le contexte et le nat activé. Ce résultat est obtenu en tapant cette commande ***sip show users*** dans la console CLI.

6.3.2 Test d'un appel entre deux utilisateurs SIP avec x-lite via Asterisk

Pour ce faire, on a besoin d'installer ***x-lite*** et de le configurer pour les deux clients SIP d'asterisk. Et par la suite, on crée un plan de numérotation à l'aide du fichier ***extensions.conf*** dans Asterisk.

6.3.2.1 Installation de x-lite

Cette source est téléchargeable à partir de ce lien
http://www.counterpath.com/index.php?menu=download

Pour lancer ce logiciel, on doit aller dans le dossier de xten-xlite et taper cette commande :

- ✓ ***./xtensoftphone***

Par la suite, on aura une fenêtre de x-lite qui s'ouvre et en parallèle une autre fenêtre de configuration Audio Tuning Wizard. Le but est de configurer le son du micro. Voir figure 19 en bas :

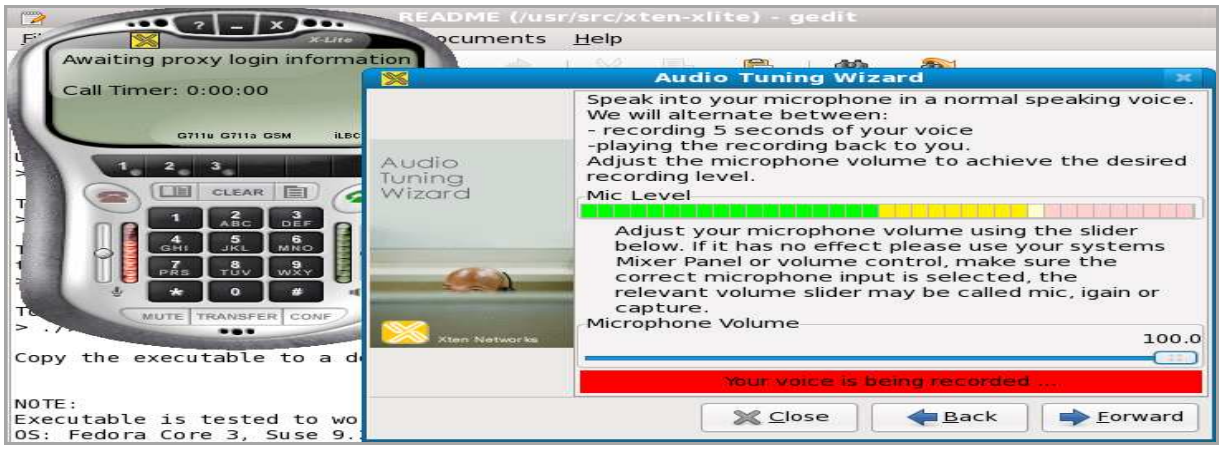


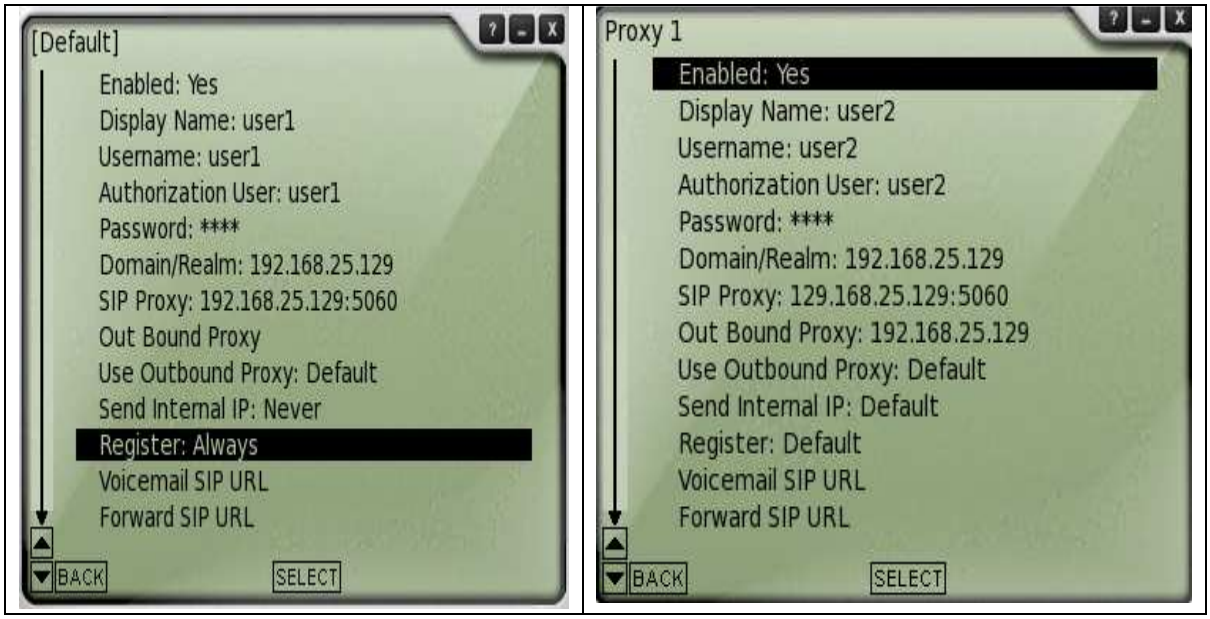
Figure 19: Fenêtre x-lite

La prochaine étape est de configurer le menu de x-lite pour que l'appel entre les 2 clients fonctionne correctement.

6.3.2.2 Configuration de x-lite pour les deux utilisateurs SIP

Il y a 2 principaux changements à les faire dans le menu de x-lite [10]:

- Aller dans ***Menu>advanced System settings>Audio Settings> Transmit Silence*** et mettre ce paramètre à ***yes*** au lieu de no.
- Aller dans ***Menu>System Settings> SIP Proxy> Default*** pour user1 et ***Menu>System Settings> SIP Proxy> Proxy1*** pour user2 et entrer les paramètres nécessaires pour chacun d'eux.



Les changements consistent à introduire ces paramètres dans le menu SIP Proxy comme c'est indiqué par le tableau ci-dessus :

- ✓ ***Enabled*** : toujours à la valeur yes au lieu de no.
- ✓ ***Display Name, Username, Authorization User*** : prennent la valeur de ***username*** de notre fichier sip.conf.
- ✓ ***Password*** : prend la valeur ***secret*** de notre fichier sip.conf pour chaque utilisateur enregistré.
- ✓ ***Domain/Realm*** : on met l'adresse IP de notre serveur Asterisk.
- ✓ ***SIP Proxy*** : on met aussi l'adresse IP de notre serveur Asterisk suivi de son numéro se port.

Si tous ces paramètres ont été introduits correctement et se concordent avec les paramètres SIP de notre fichier sip.conf. On aura un message ***logging in*** dans la fenêtre de x-lite. Ceci implique que l'authentification via asterisk pour les deux utlisateurs est réussie.

La procédure d'authentification d'un utilisateur SIP dans X-lite via Asterisk se déroule selon ces échanges du diagramme en flèche ci-dessous [11] :

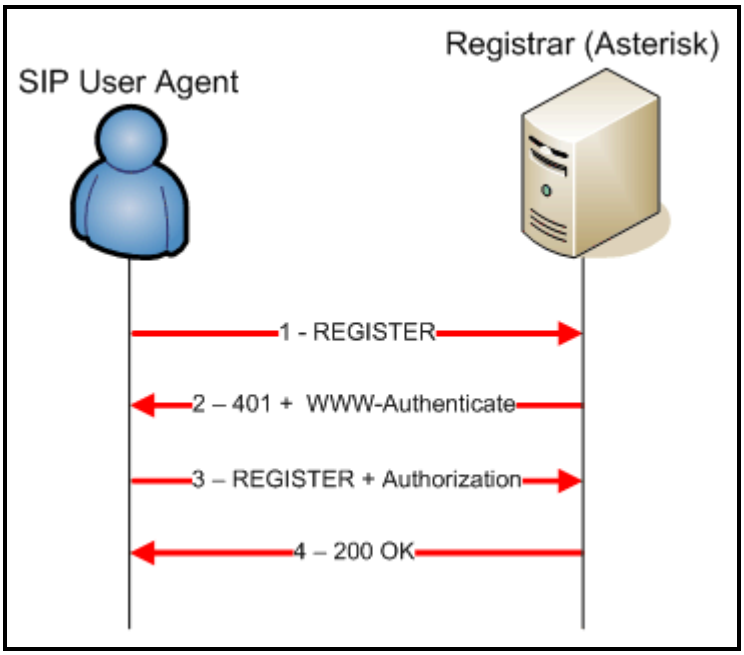


Figure 20: Diagramme en flèche d'un enregistrement SIP sous X-lite

Si les échanges de la figure 20 ci-dessus pour l'authentification ont été respectés, l'utilisateur peut effectuer des appels dans X-lite depuis cet instant.

6.3.2.3 *Résultat du Test d'un appel effectué dans x-lite*

Avant de faire les appels, nous devons configurer un plan de numérotation dans Asterisk à l'aide du fichier ***extensions.conf***. Dans ce fichier, nous avons définit le contexte enregistré dans sip.conf avec les instructions autorisés à être appliqués par nos deux clients.

Dans extensions.conf, on trouve le contexte qui permet de définir une portion d'appel. Et ce contexte contient une ou plusieurs extensions. La syntaxe générale est de cette forme :
[contexte]
exten => numéro de l'appel, numéro de priorité, application ()

- ✓ ***numéro de l'appel*** : est l'extension
- ✓ ***numéro de priorité*** : car chaque extension peut contenir plusieurs étapes donc il faut les classer selon un ordre croissant.
- ✓ ***application()*** : c'est la fonction qui va être déclenchée lorsque l'extension est appelée. Et Asterisk plusieurs types de ces applications.

Par exemple : ***exten => 100, 1, Dial (SIP/100)*** signifie que si un poste compose le numéro 100 donc c'est le poste SIP 100 va sonner.

Dans le cas de nos 2 clients SIP, le fichier ***extensions.conf*** aura ces lignes ajoutées :

```
[internal]
exten => 1234,1,Dial(SIP/user1)
exten => 5678,1,Dial(SIP/user2)
```

Comme les téléphones SIP sont des téléphones multi-ligne, ça veut dire qu'ils peuvent accepter plusieurs appels entrants en même temps. Donc dans notre x-lite, on va essayer de faire un appel de SIP 4321 à 5678, il suffit de le répondre sur la ligne 3 du soft phone et essayer de lui parler pendant un petit laps de temps et vice versa.
On sait que notre Asterisk enregistre par défaut les traces de tous les appels effectués dans un fichier ***Master.csv*** sur ce chemin ***var/log/asterisk/cdr_csv***. Ceci donne des informations sur la destination, source, la durée de l'appel, à quel moment l'appel a été fait, etc. Pour plus de détail sur ces paramètres enregistrés dans ce fichier CDR d'Asterisk, il faut aller à la partie 4.2.2 du rapport. Voici un aperçu de ce fichier Master.csv après le test de l'appel entre ces 2 clients SIP :

```
"","4321","5678","internal","""user1"" <4321>","SIP/user1-08c3b988","SIP/
user2-08c3f900","Dial","SIP/user2","2008-10-26 20:48:57","2008-10-26 20:49:19","2008-10-26
20:51:25",148,126,"ANSWERED","DOCUMENTATION","1225054137.0",""
"","5678","4321","internal","""user2"" <5678>","SIP/user2-08c3b988","SIP/
user1-08c3f900","Dial","SIP/user1","2008-10-26 20:52:59","2008-10-26 20:53:06","2008-10-26
20:53:16",17,10,"ANSWERED","DOCUMENTATION","1225054379.2",""
```

6.4 FreeRadius et base de donnée MYSQL

L'intégration d'une base de données à FreeRadius nécessite de configurer les fichiers du serveur et de taper des commandes SQL. On peut l'intégrer de deux façons : interne ou externe.
Mais il n'y a pas un grand changement entre ces deux principes à part quelques modifications minimes mais le principe reste le même. Dans notre cas, le choix est fait sur l'intégration interne de MySQL à FreeRadius. Il ne faut pas oublier qu'on doit avoir un serveur MYSQL installé sur notre machine [13] et [14].

6.4.1 Configuration de FreeRadius avec MySQL

6.4.1.1 Création de la BD radius

1. Créer une base de données radius
   - ✓ ***Mysql –u root***
   - ✓ ***Create database radius***

2. Importer les schémas des tables de FreeRadius dans la nouvelle base radius
   - ✓ ***Mysql –u root radius < /usr/local/share/doc/freeradius/examples/mysql.sql***

3. Vérifier si toutes les tables de FreeRadius ont été importés la nouvelle BD radius.

```
mysql> show tables;
+-----------------+
| Tables_in_radius |
+-----------------+
| nas             |
| radacct         |
| radcheck        |
| radgroupcheck   |
| radgroupreply   |
| radippool       |
| radpostauth     |
| radreply        |
| usergroup       |
+-----------------+
9 rows in set (0.00 sec)
```

- ✓ La ***table nas*** contient les adresses IP des clients Radius ou Switch ou routeur …
- ✓ La ***table radacheck*** contient les utilisateurs avec leurs mots de passe pour l'authentification.
- ✓ La ***table radacct*** contient les valeurs de l'accounting ou facturation comme le numéro de source, destination, durée d'appel… Donc les attributs CDR.
- ✓ La table ***radgroupcheck*** permet de définir un groupe avec un type d'authentification spécifique.
- ✓ La table ***usergroup*** permet d'introduire les utilisateurs à un groupe de radgroupcheck.
- ✓ Les autres tables n'ont pas d'intérêt dans le cas de notre projet.

4. Créer un nouvel utilisateur par exemple selim pour administrer la BD radius.

```
mysql> insert into user (Host, User, Password) values ('%', 'selim', '');
Query OK, 1 row affected, 3 warnings (0.00 sec)

mysql> update user set Password = password ('selim') where user = 'selim';
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

5. Donner tous les privilèges à ce nouvel utilisateur pour l'administration la BD radius.

```
[root@localhost ~]# mysqladmin reload -u root
[root@localhost ~]# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13
Server version: 5.0.51a Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> grant all on radius.* to selim IDENTIFIED BY "selim";
Query OK, 0 rows affected (0.01 sec)

mysql> exit
Bye
```

6.4.1.2 *Comment Modifier les fichiers de configuration de FreeRadius pour MySQL*

L'utilisation de MYSQL avec FreeRadius nécessite à faire quelques modifications dans les deux fichiers de configuration : ***sql.conf*** et ***radiusd.conf***. Ces fichiers se trouvent sur notre machine dans ce chemin ***usr/local/etc/raddb***.

- ✓ ***sql.conf*** contient les informations d'identification de la base de données et des requêtes SQL prête à être utiliser par Freeradius.

- ✓ ***radiusd.conf*** est le fichier de configuration générale de FreeRadius.

- Dans ***sql.conf***, il faut spécifier le nom de la machine contenant le serveur MYSQL. Par exemple si on veut mettre le serveur MYSQL sur une autre machine, on donne le nom de la machine distante. On donne aussi le nom et le mot de passe de l'administrateur du serveur MySQL pour se connecter. Le dernier paramètre est introduire le nom de la base de données ***radius*** à utiliser avec FreeRadius. Dans notre projet, on a utilisé ces paramètres de configurations d'après la capture ci-dessous :

```
sql {
        # Database type
        # Current supported are: rlm_sql_mysql, rlm_sql_postgresql,
        # rlm_sql_iodbc, rlm_sql_oracle, rlm_sql_unixodbc, rlm_sql_freetds
        driver = "rlm_sql_mysql"

        # Connect info
        server = "localhost"
        login = "root"
        password = "rootpass"

        # Database table configuration
        radius_db = "radius"
```

Ensuite, il faut décommenter une ligne à la fin du fichier ***sql.conf*** pour donner le droit de chercher les clients radius depuis la table ***nas*** de la base de données radius.

- ✓ ***readclients = yes***

- Dans ***radiusd.conf***, on doit décommenter ces lignes qui sont :

- ✓ ***$INCLUDE ${confdir}/sql.conf*** pour la version FreeRadius-1.1.7
- ✓ ***$INLCLUDE sql.conf*** pour les versions 2.0 de FreeRadius

- ✓ ***$INLCUDE sql/mysql/counter.sql*** pour les versions 2.0 de FreeRadius.
- ✓ Décommenter les sections ***counter pour les modules sql*** .Pour la version FreeRadius-1.1.7, elles ne sont pas commentées par défaut.

- ✓ Décommenter ***sql*** et commenter ***files*** dans la section ***authorize***.
- ✓ Décommenter ***sql*** dans la section ***accounting***.
- ✓ Décommenter ***sql*** dans la section ***session***.
- ✓ Décommenter ***sql*** dans la section ***post-auth***.

6.4.2 *Test de FreeRadius avec MySQL*

Avant de faire le test, il faut comprendre la structures des tables de FreeRadius dans la base de données radius.

6.4.2.1 *Table nas*

Cette figure ci-dessous montre les différents champs de cette table et quel champ ne doit pas être vide par exemple le nom du nas et le secret partagé du client radius doivent être introduits.

```
mysql> describe nas;
+-------------+--------------+------+-----+---------------+----------------+
| Field       | Type         | Null | Key | Default       | Extra          |
+-------------+--------------+------+-----+---------------+----------------+
| id          | int(10)      | NO   | PRI | NULL          | auto_increment |
| nasname     | varchar(128) | NO   | MUL | NULL          |                |
| shortname   | varchar(32)  | YES  |     | NULL          |                |
| type        | varchar(30)  | YES  |     | other         |                |
| ports       | int(5)       | YES  |     | NULL          |                |
| secret      | varchar(60)  | NO   |     | secret        |                |
| community   | varchar(50)  | YES  |     | NULL          |                |
| description | varchar(200) | YES  |     | RADIUS Client |                |
+-------------+--------------+------+-----+---------------+----------------+
8 rows in set (0.03 sec)
```

Figure 21: table nas de la BD radius

Voici la commande pour ajouter un client radius locale dans cette table :

```
mysql> insert into nas(nasname, shortname,secret) Values ('127.0.0.1','localhost
','testing123');
Query OK, 1 row affected (0.04 sec)

mysql> select * from nas;
+----+-----------+-----------+-------+-------+------------+-----------+---------
------+
| id | nasname   | shortname | type  | ports | secret     | community | descript
ion   |
+----+-----------+-----------+-------+-------+------------+-----------+---------
------+
|  1 | 127.0.0.1 | localhost | other |  NULL | testing123 | NULL      | RADIUS C
lient |
+----+-----------+-----------+-------+-------+------------+-----------+---------
------+
1 row in set (0.01 sec)
```

Figure 22: Insertion d'un client radius dans la table nas

6.4.2.2 *Table radcheck*

Cette table permet d'ajouter des clients pour les authentifier dans FreeRadius. Dans cette partie, on va ajouter deux type de clients, dont un utilise un type ***d'authentification PAP*** et un autre ***CHAP***. Avant d'expliquer la différence entre ces 2 types d'authentification et comment les insérer dans cette table. On va vous monter la structure de la table ***radcheck***. Voir figure 23:

```
mysql> describe radchek;
ERROR 1146 (42S02): Table 'radius.radchek' doesn't exist
mysql> describe radcheck;
+-----------+------------------+------+-----+---------+----------------+
| Field     | Type             | Null | Key | Default | Extra          |
+-----------+------------------+------+-----+---------+----------------+
| id        | int(11) unsigned | NO   | PRI | NULL    | auto_increment |
| UserName  | varchar(64)      | NO   | MUL |         |                |
| Attribute | varchar(32)      | NO   |     |         |                |
| op        | char(2)          | NO   |     | ==      |                |
| Value     | varchar(253)     | NO   |     |         |                |
+-----------+------------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

Figure 23: table radcheck dans la BD radius

L'insertion d'un utilisateur de type ***CHAP*** se fait de cette manière :

```
mysql> insert into radcheck (Username, Attribute,op,Value) values ('test','Passw
ord','==','test');
Query OK, 1 row affected (0.00 sec)

mysql> insert into radcheck (Username, Attribute,op,Value) values ('test','Auth-
type','==','local');
Query OK, 1 row affected (0.00 sec)
```

L'insertion d'un utilisateur de type ***PAP*** se fait comme suit :

```
mysql> insert into radcheck (Username, Attribute,op,Value) values ('test1','Cryp
t-Password','==',ENCRYPT('test'));
Query OK, 1 row affected (0.01 sec)

mysql> insert into radcheck (Username, Attribute,op,Value) values ('test1','Auth
-type','==','Crypt-local');
Query OK, 1 row affected (0.00 sec)
```

D'après les deux insertions ci-dessus, on voit bien qu'on peut enregistrer le mot de passe de deux manières différentes : une en ***texte claire (CHAP)*** et l'autre ***crypté (PAP)***. La preuve est donnée par la capture ci-dessous du contenu de la table ***radcheck*** après ces deux insertions effectuées auparavant.

```
mysql> select * from radcheck;
+----+----------+----------------+----+---------------+
| id | UserName | Attribute      | op | Value         |
+----+----------+----------------+----+---------------+
|  2 | test     | Password       | == | test          |
|  3 | test     | Auth-type      | == | local         |
|  4 | test1    | Crypt-Password | == | AOEaujPvYmy/c |
|  5 | test1    | Auth-type      | == | Crypt-local   |
+----+----------+----------------+----+---------------+
4 rows in set (0.00 sec)
```

CHAP permet de stocker les mots de passe en clair :

- ✓ ***Avantage*** : les mots de passes ne circulent pas en clair sur la ligne téléphonique entre l'utilisateur et le serveur.
- ✓ ***Désavantage*** : le stockage des mots de passes doit se faire en clair sur le serveur.

PAP permet de stocker les mots de passe cryptés :

- ✓ ***Avantage*** : les mots de passes ne sont pas en clair sur le serveur.
- ✓ ***Désavantage*** : les mots de passes circulent en clair sur la ligne téléphonique entre l'utilisateur et le serveur.

Cette différenciation nous donne une penchée sur l'authentification de type PAP car un pirate peut sniffer qu'un seul mot de passe sur la ligne téléphonique. Et dans le cas de CHAP, le serveur peut être piraté donc tous les mots de passes des utilisateurs sont découverts.

6.4.2.3 Test d'authentification avec la commande radtest

Nous avons testé l'utilisateur ***test*** qui utilise l'authentification ***CHAP***. On voit bien qu'on reçoit un paquet de type ***Access-Accept*** avec les données de l'utilisateur pour confirmer l'identification réussie.

```
[root@localhost ~]# radtest test test localhost:1812 0 testing123
Sending Access-Request of id 144 to 127.0.0.1 port 1812
        User-Name = "test"
        User-Password = "test"
        NAS-IP-Address = 255.255.255.255
        NAS-Port = 0
rad_recv: Access-Accept packet from host 127.0.0.1:1812, id=144, length=20
```

Nous avons aussi testé l'utilisateur ***test1*** qui utilise l'authentification PAP. On reçoit aussi le même type de paquet pour confirmer l'identification réussie.

```
[root@localhost ~]# radtest test1 test localhost:1812 0 testing123
Sending Access-Request of id 138 to 127.0.0.1 port 1812
        User-Name = "test1"
        User-Password = "test"
        NAS-IP-Address = 255.255.255.255
        NAS-Port = 0
rad_recv: Access-Accept packet from host 127.0.0.1:1812, id=138, length=20
```

6.5 Implémentation du clientradius-ng sous Asterisk et FreeRadius

Après avoir installé les deux serveurs Asterisk et FreeRadius et testé leurs fonctionnements, nous allons maintenant mettre en place leur interface de communication ***radiusclient-ng***. Ce dernier servira d'intermédiaire entre eux pour pouvoir échanger des messages de type radius. C'est lui qui va permettre d'authentifier les utilisateurs SIP d'Asterisk via FreeRadius et qui va garantir l'envoie des paramètres de l'accounting des appels à FreeRadius.

Pour ce faire, nous avons construit une machine local contenant un PBX Asterisk, le client radius (***clientradius-ng***) et un serveur FreeRadius. Cette plateforme va nous permettre de contrôler et de tester l'authentification des utilisateurs via radius ainsi que le principe de facturation.

Cette partie va nous montrée comment installer et configurer ce client radius dans Asterisk et FreeRadius correctement. Elle traitera une partie du test de son fonctionnement pour l'authentification et l'accounting [15] et [7].

6.5.1 Installation et configuration du radiusclient-ng sur Fedora9

6.5.1.1 Installation du radiusclient-ng

La source de cette librairie se trouve à partir de ce lien :
http://prdownload.berlios.de/radiusclient-ng/radiusclient-ng-0.5.5.1.tar.gz

Les commandes d'installation sont :

```
root@/usr/src# tar xvfz radiusclient-ng-0.5.5.1.tar.gz
root@/usr/src# cd radiusclient-ng-0.5.5.1
./configure
make
make install
```

6.5.1.2 Configuration du radiusclient-ng-0.5.5.1

Après l'installation, le dossier contenant les fichiers de configurations se trouve par défaut sur ce chemin ***/usr/local/etc/radiusclient-ng***.

Il y a trois fichiers de configurations à modifier : ***radiusclient.conf***, ***servers*** et ***dictionary***.

- ***radiusclient.conf*** : est le fichier de configuration générale du client radius. Nous avons modifié ces lignes comme ceci :
  - ✓ ***authserver*** localhost
  - ✓ ***acctserver*** localhost

- ***servers*** : sert à spécifier l'adresse IP du serveur Radius avec son secret partagé. Seulement les NAS ayant le même secret que lui peuvent communiquer avec ce serveur. On ajoute cette ligne car notre serveur FreeRadius est local à la machine.

  ✓ localhost testing123

- ***dictionary*** : contient tous les attributs de différents types de dictionnaires. Mais il ne contient pas le dictionnaire ***dictionary.digium d'Asterisk***. il faut inclure ce dictionnaire d'Asterisk dans ce fichier par cette ligne :

  ✓ ***$INCLUDE /usr/src/asterisk-1.4.21.2/contrib/dictionary.digium***

6.5.2 Configuration de FreeRadius

La configuration du serveur nécessite la modification de quelques fichiers. Ces derniers se trouvent sur ce chemin ***usr/local/etc/raddb*** :

- ***clients.conf*** : permet de décrire les clients radius qui ont la permission de contacter le serveur seulement s'ils ont le même secret que FreeRadius. Le client local est déjà défini par défaut, donc il suffit juste de vérifier s'il n'est pas commenté.

- ***radius.conf*** : contient la configuration générale du serveur, il faut commenter la partie ***listen***. Ensuite, on doit vérifier si ***port*** est mis à ***0*** par défaut. Ceci nous permettra de démarrer le serveur sur les bons ports ***1812*** et ***1813***, et d'utiliser l'authentification et l'accounting sur notre serveur.

- ***naslist*** : sert aussi à définir les clients radius d'autre façon que clients.conf. Il faut vérifier si le client localhost est décommenté. Le serveur a besoin d'utiliser les deux fichiers pour contrôler l'accès aux clients radius.

- ***dictionary*** : contient des attributs du dictionnaire de FreeRadius. il faut ajouter les attributs du dictionnaire d'Asterisk pour que le client radius puisse comprendre FreeRadius lors de l'échange des CDR. Ce dictionnaire existe déjà par défaut avec les nouvelles versions de FreeRadius. En tout cas, la version 1.7 de notre projet contient ce dictionnaire d'Asterisk par défaut.

- ***users*** : enregistre les utilisateurs qui vont être utilisés lors de l'authentification via FreeRadius. L'identification est faite à l'aide d'un nom d'utilisateur et un mot de passe.

6.5.3 Test du radiusclient-ng avec FreeRadius

Ce test nous permettra de vérifier et de corriger les problèmes de configuration avant de continuer la configuration du client dans Asterisk. Cela nous facilite la tâche de détecter l'origine des erreurs lors de son utilisation avec les deux serveurs ensembles.

Pour ce faire, on ajoute un utilisateur de type digest dans le fichier ***users*** de FreeRadius comme suit :

```
Test        Auth-type  := digest, CleartextPassword := "test"
            Reply-Message  := "Authentification digest, OK!!!!!"
```

Ensuite, on crée un fichier texte contenant les paramètres digest nécessaires pour ce type d'authentification. Ils doivent être introduits sur une seule ligne. Ce fichier porte le nom digest et il est comme ceci :

```
User-Name="test", Digest-Response = "438b8240d5f2a52d1612cd93a6fd539a", Digest-Realm=
"testrealm", Digest-Nonce = "1234abcd", Digest-Method= "INVITE", Digest-
URI="sip:5555551212@exemple.com", Digest-Algorithm="MD5", Digest-User-Name="test"
```

6.5.3.1 *Test d'authentification du client radius*

On lance le client radius en tapant cette commande dans notre Shell :

```
#  radclient –f digest  localhost :1812 auth testing123
```

- ✓ ***digest***: est le nom du fichier contenant les paramètres pour l'authentification digest.
- ✓ ***localhost*** : est l'adresse IP ou nom du serveur FreeRadius.
- ✓ ***1812*** : est le port d'authentification
- ✓ ***auth*** : on veut faire un test d'authentification
- ✓ ***testing123*** : est le secret partagé entre le client radius et le serveur FreeRadius

Si tout vas bien, on doit voir une réponse comme celle qui suit :

```
[root@localhost ~]# radclient -f digest localhost:48416 auth testing123
Received response ID 115, code 3, length = 54
        Reply-Message = "Authentification Digest, ok!!!!."
[root@localhost ~]#
```

6.5.3.2 *Test d'accounting du client radius*

Pour l'accounting, la commande est la même. La seule différence est de mettre ***acct*** au lieu d' ***auth*** et de modifier le port avec celui de l'accounting ***1813***.

```
#   radclient –f digest  localhost :1813 acct  testing123
```

```
[root@localhost ~]# radclient -f digest localhost:1813 acct testing123
Received response ID 108, code 5, length = 20
```

Si l'accounting fonctionne correctement, le serveur FreeRadius crée automatiquement un dossier qui porte le nom de la machine contenant un fichier texte d'accounting. Ce dernier va

contenir ces paramètres de digest envoyé par le client radius lors de l'accounting. Il se trouve sur ce chemin ***/usr/local/var/log/radius/radacct/127.0.0.1***.

```
Wed Nov  5 23:59:14 2008
        User-Name = "test"
        Digest-Response = "472ba8ee634d289dfdb4635bc7008b79"
        Digest-Attributes = 0x010b746573747265616c6d
        Digest-Attributes = 0x020a3132333461626364
        Digest-Attributes = 0x0308494e56495445
        Digest-Attributes = 0x041c7369703a353535353535313231324065786564706c652e636f6d
        Digest-Attributes = 0x06054d4435
        Digest-Attributes = 0x0a0674657374
        NAS-IP-Address = 127.0.0.1
        Client-IP-Address = 127.0.0.1
        Acct-Unique-Session-Id = "5604c7e27eabd96b"
        Timestamp = 1225947554
```

Figure 24: fichier de l'accounting dans FreeRadius

Comme la première étape du test fonctionne parfaitement, on passe à la configuration d'Asterisk pour qu'il puisse envoyer les paramètres CDR de facturation au serveur FreeRadius.

6.5.4 Configuration d'Asterisk

Si Asterisk est déjà installé alors il faut le recompiler pour qu'il prenne en compte de l'existence du package clientradius-ng. On doit taper ces commandes pour la recompilation :

```
root@/usr/src# cd asterisk-1.4.21.2
make clean
make update
make
make install
```

Dans le fichier ***cdr.conf*** se trouvant dans ce chemin ***/etc/asterisk***, nous avons décommenté l'en-tête radius pour qu'Asterisk sait où trouver la librairie client radius grâce au path.

```
[radius]
;usegmtime=yes    ; log date/time in GMT
;loguniqueid=yes  ; log uniqueid
;loguserfield=yes ; log user field
; Set this to the location of the radiusclient-ng configuration file
; The default is /etc/radiusclient-ng/radiusclient.conf
radiuscfg => /usr/local/etc/radiusclient-ng/radiusclient.conf
```

Dans le fichier ***cdr_custom.conf***, on doit décommenter la partie ***mappings*** pour envoyer tous les paramètres du CDR. Ces variables sont définies dans ce fichier.

Dans le fichier ***cdr_manager.conf***, on active l'envoie des CDR en mettant ***enabled = yes***.
La dernière configuration à faire est de copier le fichier ***cdr_radius.so*** dans le répertoire module d'Asterisk. Ce répertoire se trouve dans ***/usr/lib/asterisk/modules***. Après un redémarrage, Asterisk charge automatiquement tous les modules présents dans ce répertoire.

6.5.5 *Test de l'authentification à l'aide des appels SIP avec X-lite*

En utilisant les mêmes clients SIP enregistrés dans le fichier ***sip.conf*** d'Asterisk, on va réaliser le test d'authentification à l'aide des appels dans X-lite. On doit décommenter l'option ***secret*** pour user1 et user2, et ajouter un paramètre ***auth-type=radius*** dans sip.conf. Ces changements signifient qu'on va utiliser l'authentification de type digest, et de chercher le mot de passe dans FreeRadius en utilisant le cryptage ***md5***.

Le test a été réalisé en lançant le logiciel X-lite pour authentifier user1 et user2. S'il y a un message ***logging in*** dans X-lite, donc l'authentification est accepté. Maintenant, on peut émettre des appels entre les deux utilisateurs SIP. Cette procédure d'authentification est caractérisée par ses échanges de messages d'après le digramme en flèche ci-dessous :

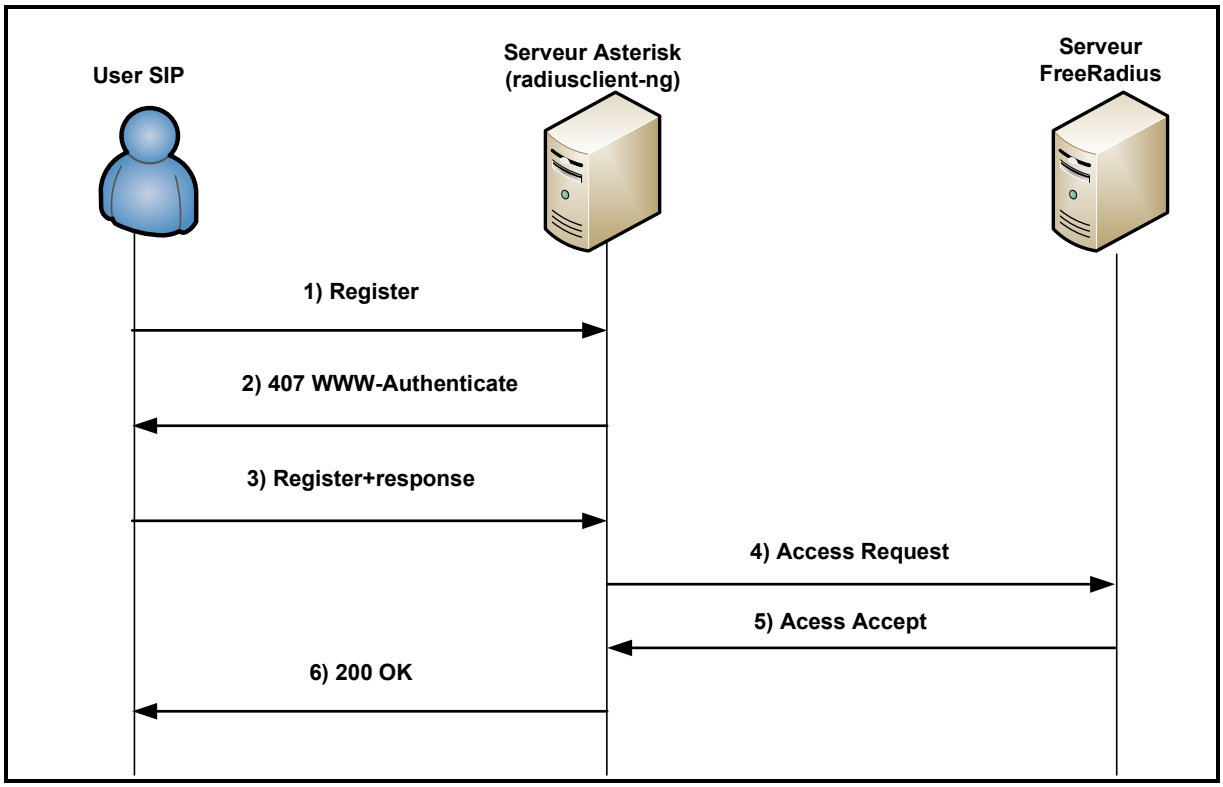


Figure 25: Diagramme en flèche de l'Authentification par radiusclient-ng

6. L'user agent va envoyer une demande d'enregistrement dans Asterisk.
7. Le serveur Asterisk répond par ***407*** pour lui indiquer qu'il faut passer un proxy d'authentification.
8. L'user agent renvoie un enregistrement avec une réponse à la ***requête 407***. C'est-à-dire il a su maintenant qu'il doit passer un proxy.
9. Le client radiusclient-ng envoie un message ***Access Request*** contenant le nom d'utilisateur, son mot de passe et l'adresse IP du NAS pour l'identifier.
10. Le serveur FreeRadius répond par ***Access Accept*** dans le cas d'une identification correcte, et par un ***Access Reject*** dans le cas d'un nom d'utilisateur faux, ou mot de passe erroné, ou adresse IP du NAS incorrecte.

11. Si l'authentification est réussie, Asterisk envoie un message OK pour pouvoir faire un appel.

Pour tester notre application, nous avons lancé un sniffer Wireshark pour pouvoir visualiser les paquets radius échangés entre les différents composants de notre plateforme de test. Mais malheureusement, la capture des trames ne donne aucun paquet radius transmis. On voyait dans la capture que des paquets SIP. Le doute s'est installé sur le bon fonctionnement de l'authentification par ce ***clientradius-ng***.

6.5.6 *Test de l'accounting à l'aide des appels SIP avec X-lite*

La capture de la figure 26 illustre les différents champs CDR enregistrés dans le fichier détail crée automatiquement dans FreeRadius. Ce fichier donne beaucoup d'informations importantes sur l'appel effectué lors de notre test comme le numéro de l'appelant et de l'appelé, la durée de la communication, à quel moment a été effectué l'appel, à quel moment l'utilisateur a répondu, la durée de la session de connexion, l'adresse IP du client…

```
Acct-Status-Type = Stop
Asterisk-Acc-Code = ""
Asterisk-Src = "4321"
Asterisk-Dst = "5678"
Asterisk-Dst-Ctx = "internal"
Asterisk-Clid = ""user1" <4321>"
Asterisk-Chan = "SIP/user1-088d1558"
Asterisk-Dst-Chan = "SIP/user2-088dae90"
Asterisk-Last-App = "Dial"
Asterisk-Last-Data = "SIP/user2"
Asterisk-Start-Time = "2008-11-06 01:08:04 -0500"
Asterisk-Answer-Time = "2008-11-06 01:08:06 -0500"
Asterisk-End-Time = "2008-11-06 01:08:16 -0500"
Asterisk-Duration = 12
Asterisk-Bill-Sec = 10
Asterisk-Disposition = "ANSWERED"
Asterisk-AMA-Flags = "DOCUMENTATION"
User-Name = "SIP/user1-088d1558"
Acct-Session-Id = "1225951684.4"
NAS-Port = 0
Acct-Delay-Time = 0
NAS-IP-Address = 127.0.0.1
Client-IP-Address = 127.0.0.1
```

Figure 26: Fichier CDR des appels reçus dans FreeRadius

On trouve ces mêmes informations que celui du fichier ci-dessus concernant les paramètres CDR de facturation des appels dans la fenêtre debug du FreeRadius. Voir figure 27 ci-dessous :

```
        Acct-Status-Type = Stop
        Asterisk-Acc-Code = ""
        Asterisk-Src = "4321"
        Asterisk-Dst = "5678"
        Asterisk-Dst-Ctx = "internal"
        Asterisk-Clid = ""user1" <4321>"
        Asterisk-Chan = "SIP/user1-088d1558"
        Asterisk-Dst-Chan = "SIP/user2-088dae90"
        Asterisk-Last-App = "Dial"
        Asterisk-Last-Data = "SIP/user2"
        Asterisk-Start-Time = "2008-11-06 01:08:04 -0500"
        Asterisk-Answer-Time = "2008-11-06 01:08:06 -0500"
        Asterisk-End-Time = "2008-11-06 01:08:16 -0500"
        Asterisk-Duration = 12
        Asterisk-Bill-Sec = 10
        Asterisk-Disposition = "ANSWERED"
        Asterisk-AMA-Flags = "DOCUMENTATION"
        User-Name = "SIP/user1-088d1558"
        Acct-Session-Id = "1225951684.4"
        NAS-Port = 0
        Acct-Delay-Time = 0
        NAS-IP-Address = 127.0.0.1
  Processing the preacct section of radiusd.conf
modcall: entering group preacct for request 0
```

Figure 27: Affichage des CDR dans FreeRadius en mode debug

Pour tester l'accounting, on a eu recours une autre fois au sniffer pour capter les trames radius échangés entre notre serveur Asterisk et FreeRadius à l'aide de l'interface radiusclient-ng. D'après le principe de l'accounting, on devait avoir ces types de paquets radius dans notre capture comme c'est indiqué dans ce digramme en flèche ci-dessus :

Cette figure est prise depuis la source [16].

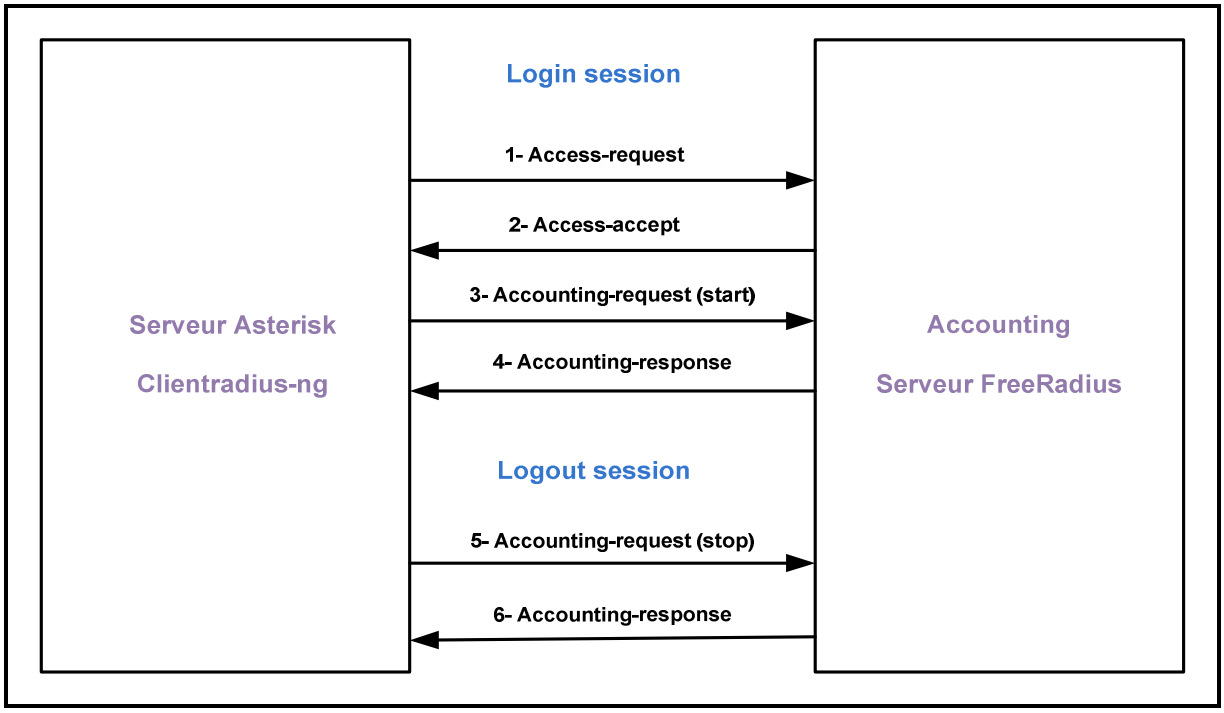


Figure 28: Diagramme en flèche de l'Accounting

1. Le client radius récupère le nom d'utilisateur et le mot de passe du client SIP, il crypte ces infos à l'aide du secret partagé et il envoie A***ccess-request*** pour le FreeRadius (A***uthentification***).

2. Si le nom d'utilisateur et le mot de passe sont validés par FreeRadius, alors il envoie un ***Access-accept*** en ajoutant d'autres informations (adresse IP, masque de réseau, temps de session autorisé, etc.) au client radius (***Autorisation***).

3. Le client envoie un ***Accounting-request start*** pour indiquer que l'utilisateur est connecté à Asterisk (***Accounting***).

4. Le serveur FreeRadius répond par un ***Accounting-response*** quand les informations d'accounting sont enregistrées.

5. Quand l'utilisateur se déconnecte d'Asterisk, le client radius envoie un ***Accounting-request stop*** avec ces infos :

   - ✓ ***Delay time*** : le temps qu'il tente d'envoyer ce message.
   - ✓ ***Input octets*** : le nombre des octets reçus par l'utilisateur.
   - ✓ ***Output octets*** : le nombre des octets envoyés par l'utilisateur.
   - ✓ ***Session time*** : le nombre de seconde que l'utilisateur est connecté.
   - ✓ ***Input packets*** : le nombre de paquets reçus par l'utilisateur.
   - ✓ ***Output packets*** : le nombre de paquets envoyés par l'utilisateur.
   - ✓ ***Reason*** : la raison pour laquelle l'utilisateur est déconnecté.

6. Serveur FreeRadius répond par un ***Accounting-response*** lorsque tous les informations de facturation sont stockés.

La capture des trames dans le sniffer montre qu'aucun paquet Accounting a été transmis entre le radiusclient-ng et FreeRadius. Ceci nous a rendus perplexe face à cette solution malgré que nous ayons reçu comme même des traces concrètes de l'accounting dans le fichier détail de FreeRadius stockant les infos de facturation. D'après les figures 26 et 27 ci-dessus, nous avons la preuve que les infos de l'accounting sous forme de fichier texte ont été bel bien enregistrées correctement dans notre serveur d'authentification.

6.6 Faiblesse de la solution CDR à l'aide du client radiusclient-ng

Les tests des captures sont faits sur notre machine localhost où le serveur Asterisk et FreeRadius ont été installé en local. Cette installation nous a laissé douter sur l'origine de l'absence des paquets radius échangés entre eux. Pour remédier à ce problème, nous avons essayé d'installer une autre plateforme de test, cette fois-ci les deux serveurs seront mis sur des machines différentes. Le résultat du test est le même que celui en local donc toujours réception des paramètres CDR de l'accounting dans le fichier détail de facturation de FreeRadius et absence de paquets radius échangés.

En consultant la documentation publiée sur le net de cette solution, il s'avère que l'authentification peut se faire aussi à l'aide d'un autre client radius ***pam-radius***.
Notre choix est fait sur le deuxième client radiusclient-ng car il propose une solution d'envoi de l'accounting à FreeRadius en parallèle de son mécanisme d'authentification.
Alors que pam-radius permet seulement d'authentifier les utilisateurs via radius.

Mais ce radiusclient-ng est rarement utilisé avec Asterisk et FreeRadius. Sur le net, on ne trouve pas de la documentation concernant sa configuration pour l'authentification via radius. Alors que la documentation de son utilisation avec un autre serveur OpenSER (serveur SIP) est facilement trouvable. Le seule guide de notre projet est la solution proposé par le développeur Mr Philippe Sultan qui est le rédacteur du document SIP peers external authentication in Asterisk - OpenPBX. Dans ce lien, on parle un tout petit peu de cette solution mais il a mis en détail la solution pam-radius.
Nous avons remarqué qu'un package manque pour faire fonctionner l'authentification correctement. Ce paquet est téléchargeable depuis cette source (svn://svn.openpbx.org/openpbx/branches/psultan/experimental/trunk) à partir de ce lien de la solution de Mr Philippe. Or il s'est avéré que ce package utile à notre projet n'est plus possible de le downloader à partir de ce lien et n'est plus disponible même sur internet.

Face à ce type de problème, nous avons écarté cette solution pour l'instant. Nous avons pris la décision d'utiliser un autre client radius pour faire fonctionner l'authentification correctement via FreeRadius. Par chance, le développeur de la solution précédente Mr Philippe Sultan, nous a envoyé un e-mail pour nous confirmer que ce client ***radiusclient-ng n'est plus fonctionnel***. Ce message reçu de sa part a été mis dans notre annexe F à titre d'information.

La nouvelle solution est le ***client radius Portaone's*** dans Asterisk. C'est un client radius écrit entièrement en Perl, portable et facile à utiliser. Il permet de faire de l'authentification de type digest et de l'accounting via radius. Par la suite, nous allons détailler les procédures de l'installation et de configuration de la solution externe portaOne dans Asterisk. Nous allons aussi crée une nouvelle plateforme de test avec deux nouveaux machines Asterisk et FreeRadius. Et nous terminerons cette partie par un test de son fonctionnement avec notre serveur FreeRadius et Asterisk en essayant cette fois-ci d'avoir une captures des paquets radius échangés.

6.7 Implémentation de la nouvelle solution Portaone dans Asterisk pour l'utilisation de l'authentification et l'accounting

6.7.1 Installation du client radius Portaone

Avant de commencer l'installation de ce client, on a eu recours à réinstaller une nouvelle machine virtuelle Asterisk avec les modules accessoires utiles pour son fonctionnement correcte. Les étapes de l'installation sont : [17]

1. Télécharger les scripts perl qui feront office de client radius dans Asterisk. Ce dossier downloadé contient aussi des patches à être appliqués. Ces patches sont des corrections aux fichiers d'Asterisk comme chan_sip.c, channel.h, channel.c…
   Ces patches correspondent à la version 1.4.11 d'Asterisk. Il reste plus qu'à les appliquer comme suit :

```
#  cp patch-channels ::chan_sip.c asterisk-1.4.11
#  cd asterisk-1.4.11
#  patch –p0 < patch-channel::chan_sip.c
```

2. Installer le module CPAN, ceci a été détaillé ci-dessus dans la partie installation d''Asterisk. L'utilité de son installation est de pouvoir installer des modules appartenant à CPAN qui seront utile pour l'authentification à l'aide de ce client :

```
Install Config ::Inifiles
Install Crypt ::CBC
Install Crypt ::DES
Install Authen ::Radius
```

   En installant le paquet radius avec l'installateur CPAN, il nous demande d'entrer :

   - ✓ L'adresse IP du serveur FreeRadius
   - ✓ Le port de l'authentification (1812)
   - ✓ Le secret partagé
   - ✓ Le nom d'utilisateur et le mot de passe de l'administrateur du serveur par défaut qui est nobody et nobody.

   Voir la capture ci-dessous pour voir plus de détail concernant ces paramètres Introduites dans le ***module Authen ::Radius*** de CPAN :

```
Make sure this machine is in your Radius clients file!
Enter hostname[:port] of your Radius server: 10.192.51.54:0
Enter shared-secret of your Radius server: testing123
Enter a username to be validated: nobody
Enter this user's password: nobody
Using Radius server 10.192.51.54:1812
ok 2
Adding attribute 1 (1) with value 'nobody'
Adding attribute 2 (2) with value 'nobody'
Adding attribute 4 (4) with value '127.0.0.1'
Sending request:
          00 01 02 03 04 05 06 07 - 08 09 0A 0B 0C 0D 0E 0F  0123456789ABCDEF

00000000  01 86 00 34 B5 5A 96 C8 - 4E 81 B6 50 BB 98 1D 49  ...4.Z..N..P...I
00000010  B1 38 EA 66 01 08 6E 6F - 62 6F 64 79 02 12 9C 77  .8.f..nobody...w
00000020  17 7F 27 E9 2E 52 5A 20 - 50 2C 48 55 D1 DC 04 06  ..'..RZ P,HU....
00000030  7F 00 00 01                                       ....
not ok 3
ok 4
  MANOWAR/RadiusPerl-0.13.tar.gz
  /usr/bin/make test -- OK
```

3. Installation des applications ***Asterisk::AGI*** et ***Asterisk ::Manager*** pour Asterisk. Nous l'avons réalisé en installant le module ***asterisk-perl*** qui est une bibliothèque permettant de développer ces deux applications. Il ne peut pas être installé par le CPAN. Cette installation est déjà citée dans la partie installation d'Asterisk page 35 du rapport.

4. Editer le fichier ***/etc/asterisk/modules.conf*** d'Asterisk pour inclure le chargement du nouveau module concernant l'application AGI. Nous devons ajouter une ligne comme ceci :
   - ✓ ***Load => res_agi.so***

6.7.2 Configuration du client radius Portaone dans Asterisk

Dans le premier point de la partie installation, nous avons téléchargé deux scripts perl du client portaone. Il y a un fichier ***agi-rad-auth.pl*** pour l'authentification et un fichier ***ast-rad-acc.pl*** pour l'accounting. Il faut copier le fichier ***agi-rad-auth.pl*** dans le répertoire ***var/lib/asterisk /agi-bin*** d'Asterisk. Et le fichier ***ast-rad-acc.pl*** dans le répertoire du fichier système ***usr/sbin***.

Asterisk Manager fournit une API permettant à des programmes externes de se connecter à Asterisk, ainsi que de pouvoir lancer des commandes et de lire des événements d'Asterisk à travers d'un socket TCP/IP. Notre ***client Portaone*** est un programme externe qui va se connecter à Asterisk et exécuter des commandes d'authentification via cette API. Avant de pouvoir utiliser les fonctionnalités du Manager, nous devons configurer le fichier ***/etc/asterisk /manager.conf*** qui contient les informations relatives à la connexion de cette l'interface.

Dans notre fichier ***manager.conf***, nous allons définir un utilisateur qui a le droit de se connecter à cette interface. Nous allons spécifier un nom d'utilisateur, un mot de passe et l'adresse IP de la machine Asterisk. Ces paramètres à définir dans ce fichier doivent correspondre à l'utilisateur de manager déclaré dans le fichier ***ast-rad-acc.pl***. L'utilisateur du Manager définit dans notre script perl d'accounting est user ***test*** avec mot de passe ***test*** et host ***localhost***.

Voici un aperçu de notre fichier ***manager.conf*** configuré dans notre projet :

```
;
; Asterisk Call Management support
;
[general]
enabled =yes
port = 5038
bindaddr = 127.0.0.1
secret = test
permit= 127.0.0.1

[test]
secret = test
permit= 127.0.0.1
read = system,call,log,verbose,command,agent,user
write = system,call,log,verbose,command,agent,user
```

Figure 29: Fichier manager.conf d'Asterisk

Pour tester le fonctionnement d'Asterisk Manager, nous devons arrêter le serveur et le redémarrer afin de lancer la connectivité au manager. Ensuite, nous lançons cette commande dans le Shell de notre machine Asterisk :

```
#  telnet localhost 5038
```

Le test consiste à faire un essai d'accès au manager avec l'utilisateur définit auparavant dans notre fichier manager.conf. Il va donc devoir créer une connexion en établissant un accès via le port ***TCP 5038*** et se connecter en utilisant la commande ***Login***. Le résultat de la connexion de cet utilisateur ***test*** à notre Manager est donné par cette capture ci-dessous :

```
[root@localhost ~]# telnet localhost 5038
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Asterisk Call Manager/1.0
Action: login
Username: test
Secret: test

Response: Success
Message: Authentication accepted
```

Nous avons remarqué la réception d'un message de connexion acceptée dans cette capture ci-dessus. Dans le cas d'une erreur dans le nom d'utilisateur ou le mot de passe, on recevra une réponse de ce type ***Erreur*** et un message : ***authentification requis***.

La prochaine étape est de définir deux utilisateurs SIP dans le fichier ***etc/asterisk/sip.conf*** pour effectuer des appels entre eux. Le seul changement par rapport à la déclaration d'un utilisateur SIP classique est d'ajouter un paramètre important pour spécifier que l'authentification est extérieure à Asterisk. Ce paramètre ajouté est ***externalauth*** qui prend la valeur ***yes***. Voici un exemple d'un utilisateur SIP configuré dans notre projet pour cette solution où le paramètre ajouté est gras :

```
[77777]
type=friend
username=77777
secret=test
regexten=77777
context= sip_auth                    ; When they register, create extension 1234
callerid= <77777>
host=dynamic                  ; This device needs to register
externalauth=yes
```

Figure 30: Fichier sip.conf d'Asterisk pour la solution Portaone

L'autre fichier important à modifier pour réaliser le test des appels des deux clients SIP, est le fichier ***etc/asterisk/extensions.conf.*** Dans ce fichier, nous allons déclarer dans ces paramètres globaux des indications sur notre serveur d'authentification FreeRadius à utiliser pour identifier les utilisateurs. Ces paramètres sont :

- ✓ ***RADIUS_Server*** : L'adresse IP de notre serveur FreeRadius.
- ✓ ***RADIUS_Secret*** : Le mot secret partagé qui a été déclaré dans FreeRadius.
- ✓ ***RADIUS_Auth_Port*** : Le numéro de port d'authentification radius (***1812***).
- ✓ ***RADIUS_Acct_Port*** : Le numéro de port d'accounting radius (***1813***).
- ✓ ***Acct_Update_Timeout*** : Le temps de l'update d'account (***60***)
- ✓ ***NAS_IP_Adress*** : L'adresse IP de la machine hébergeant le client radius (IP de la machine Asterisk dans le cas de notre projet).

Et bien sûr, définir un contexte pour les deux utilisateurs SIP dans le même fichier ***extensions.conf***. Ce contexte va contenir un groupe d'extensions qui sont des instructions à être appliquées par Asterisk dans un ordre précis. Ces extensions servent à spécifier ce qui arrive aux appels des deux utilisateurs lorsqu'ils suivent le plan de numérotation. Elles sont déclenchées par un appel entrant ou par des chiffres composés sur le canal.

Dans le cas d'utilisation de notre client portaone, on doit impérativement utiliser ces extensions avec les fonctions décrites ci-dessous :

```
[sip-auth]
; SIP Authorization headers;
exten => _X.,1,SIPGetHeader(SIP_Authorization=Proxy-Authorization)
;
; RADIUS Authorize;
exten => _X.,n,agi,agi-rad-auth.pl|Routing=SIP&AuthorizeBy=SIP

; SIP username from Digest is returned in channel variable SIP_Username
; Set correct ACCOUNTCODE for the accounting;
exten => _X.,n,Set(CDR(accountcode)=${SIP_Username})
;
; Routing information is returned in channel variable Dial_Info
; PortaBilling100 routing and authentication information returned as follows:
; SIP/number_to_dial:password:authuser@ip1/SIP/number_to_dial@ip2/;;
exten => _X.,n,NoOp(${Dial_Info})
;
; Number to dial is returned in channel variable DNID;
exten => _X.,n,NoOp(${DNID})
exten => _X.,n,Dial(${Dial_Info})
exten => _X.,n,Hangup
```

Figure 31: Fichier extensions.conf d'Asterisk pour la solution Portaone

On sait bien que la forme de l'extension est divisée en 3 parties :

- ✓ ***exten => numéro de l'appel, numéro de priorité, application ()***

Dans notre exemple, ***le numéro de l'appel*** est représenté par ***_X.*** . Le ***X*** représente un chiffre entre 0 et 9 et le ***.*** représente un ou plusieurs chiffres sans aucune restriction

La priorité est représentée par ***n***. Ceci signifie qu'Asterisk attribuera un chiffre de priorité automatiquement selon l'ordre des lignes dans le fichier.

Applications() est différente dans chaque extensions comme ***NoOp()*** , ***Dial()*** …

Après avoir détaillé la structure générale des extensions de notre exemple, nous allons citer la signification et la fonction de chaque application appelle dans les extensions de notre fichier extensions.conf. Ils seront décrits et énumérées selon l'ordre des priorités attribués par Asterisk.

1. ***SIPGetHeader(SIP_Authorization = Proxy-Authorization)*** : permet de récupérer l'en-tête depuis un appel entrant et d'attribuer à la variable de canal ***SIP_Authorization*** le contenu de l'en-tête ***Proxy-Authorization***.

2. ***agi, agi-rad-auth.pl¦Routing=SIP&AuthorizedBy=SIP***: ceci permet d'exécuter une application AGI qui permet de donner le contrôle du canal à notre programme

extérieur ***agi-rad-auth.pl*** placé dans notre répertoire ***/var/lib/asterisk/agi-bin***. Et nous avons la possibilité de spécifier autant de paramètre à passer à notre programme perl. Ici dans notre exemple, nous avons spécifié 2 paramètres dont un ***routing de type SIP*** et ***une autorisation de type SIP***, car il y a aussi une autorisation de type ***Account*** utilisé pour ***les cartes prépaye à l'aide du pin***.

3. ***Set (CDR(accountcode)=${SIP_Username})*** : permet d'attribuer à la variable CDR(accountcode) la valeur SIP_Username retournée dans le canal par ***user SIP digest***.

4. ***Noop(${Dial_Info})*** : elle ne fait rien de spécial et elle permet juste d'afficher le contenu de cette variable dans la console d'Asterisk, elle est utilisée pour des fins de débogage.

5. ***Dia l (${Dial_Info})*** : permet une tentative de connexion du canal passé en paramètre.

6. ***Hangup()*** : permet de raccrocher sans condition le canal en cours.

6.7.3 Test du client radius Portaone avec Asterisk et FreeRadius

L'infrastructure générale de la solution a été mise en place dans Asterisk. Comme précédemment, nous avons définit 2 utilisateurs SIP dans sip.conf. On lance X-lite pour paramétrer les 2 utilisateurs SIP définies dans Asterisk.

L'étape suivante est d'aller dans ce répertoire ***/usr/sbin*** et de lancer le script de l'accounting en tapant cette commande :

```
#   ./ast-rad-acc.pl
```

Or en lançant ce script, on obtient une erreur dans le Shell. Le message est : ***"Can't call method "val" on an undefined value at /usr/sbin/ast-rad-acc.pl line 302."***
La capture ci-dessous, nous montre une idée sur le fonctionnement de la fonction val et ses paramètres.

```
sub load_config {
        my $conf=Config::IniFiles->new(-file => "$config_dir/extensions.conf");
        syslog('crit', "Config file error!") if !defined $conf;

        $rad_serv = $conf->val('globals','RADIUS_Server');
        $rad_sec = $conf->val('globals','RADIUS_Secret');
        $nas_ip = $conf->val('globals','NAS_IP_Address');
        syslog('notice', "extensions.conf loaded");
}
```

Si le script ne s'exécute pas, nous ne pouvons pas tester le fonctionnement de notre client radius Portaone car ce script doit être lancé quand on démarre Asterisk pour avoir un fonctionnement correcte de cette solution. En faisant des recherches sur le net, nous ne trouvons aucune trace sur ce genre de bug. Et d'après le code perl fourni ci-dessus, nous pouvons douter des paramètres du serveur FreeRadius., alors que ces variables ont été correctement configurées.

6.7.4 Correction du bug de notre script perl du client portaone

La seule cause du bug est le script en lui même. Donc il fallait trouver une nouvelle solution Portaone ou une autre solution pour implémenter l'authentification et l'accounting via Radius.

Comme la documentation sur notre client Portaone est quasi inexistante. Après beaucoup de recherches, on a trouvé un site russe qui propose un autre client radius Portaone. Ce client est écrit en perl, utilise la même configuration précédente dans Asterisk et a le même principe d'authentification et d'accounting que l'autre. Ce nouveau client s'appelle ***radiusclient*** et il est téléchargeable depuis cette source [17] : http://www.lanbilling.ru/asterisk_solution.html

Ce lien propose 2 versions du client : une version ancienne ***radiusclient*** et une autre version ***radiusclinet-1.4***. Pour notre projet, nous avons utilisé la nouvelle version 1.4 du client.
Les seules différences entre le client Portaone et radiusclient 1.4 sont : des changements dans les 2 deux codes script perl (***ast-rad-acc.pl et agi-rad-auth.pl***) et l'utilisation d'une autre méthode d'authentification dans le fichier ***/etc/asterisk/extensions.conf*** d'Asterisk. Notre fichier de configuration d'Asterisk ***extensions.conf*** est devenu comme ceci :

```
exten => 77777,1,Set(SIP_Authorization=${SIP_HEADER(Proxy-Authorization)})
exten => 77777,n,AGI(agi-rad-auth.agi|Mode=SIP)
exten => 77777,n,Goto(default,${EXTENSION},1)
exten => 77777,n,Hangup()
exten => 88888,1,Set(SIP_Authorization=${SIP_HEADER(Proxy-Authorization)})
exten => 88888,n,AGI(agi-rad-auth.agi|Mode=SIP)
exten => 88888,n,Goto(default,${EXTENSION},1)
exten => 88888,n,Hangup()
```

Figure 32: Fichier extensions.conf pour la solution radiusclient-1.4

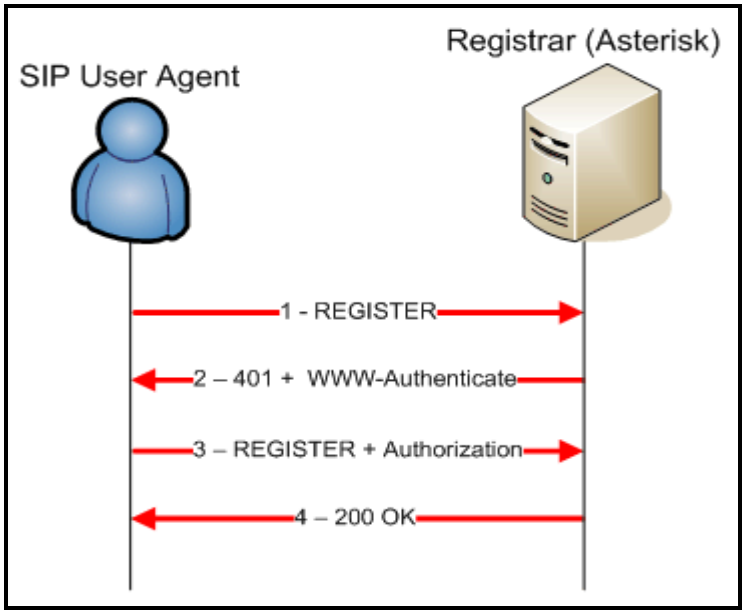
Par la suite, nous avons remplacé les anciens fichiers perl de Portaone par les nouveaux scripts téléchargés du ***radiusclient-1.4*** dans Asterisk. En lançant le script ***ast-rad-acc.pl*** d'accounting avant de redémarrer Asterisk, il s'exécute sans aucun message d'erreur. Maintenant, on peut commencer à tester le fonctionnement de l'identification et l'accounting via radius à l'aide de notre radiusclient-1.4.

6.7.5 Test du radiusclient-1.4 avec Asterisk et FreeRadius

Etant donné que toute la configuration a été faite dans Asterisk pour notre client radius et que le script d'accounting s'est lancé correctement, nous allons pouvoir tester l'authentification de ses deux utilisateurs SIP avec X-lite.

6.7.5.1 *Test du fonctionnement de l'authentification avec radiusclient-1.4*

Le test d'authentification via radius consiste à capter des paquets radius échangés entre le client radius et FreeRadius. Les deux utilisateurs s'enregistrent correctement à partir d'Asterisk avec l'envoie des messages standards de SIP. Comme c'est montré par le diagramme en flèche en bas :



Les captures SIP dans notre machine ***Asterisk d'adresse IP : 10.192.51.55*** pour l'enregistrement des deux utilisateurs SIP (***77777 et 88888***).

3 6.829383	10.192.51.55	10.192.51.55	SIP	Request: REGISTER sip:10.192.51.55
4 7.038870	10.192.51.55	10.192.51.55	SIP	Status: 100 Trying (1 bindings)
5 7.048649	10.192.51.55	10.192.51.55	SIP	Status: 401 Unauthorized (0 bindings)
6 7.136614	10.192.51.55	10.192.51.55	SIP	Request: REGISTER sip:10.192.51.55
7 7.146965	10.192.51.55	10.192.51.55	SIP	Status: 100 Trying (1 bindings)
8 7.148075	10.192.51.55	10.192.51.55	SIP	Status: 401 Unauthorized (0 bindings)
9 7.236310	10.192.51.55	10.192.51.55	SIP	Request: REGISTER sip:10.192.51.55
10 7.239744	10.192.51.55	10.192.51.55	SIP	Status: 100 Trying (1 bindings)
11 7.328796	10.192.51.55	10.192.51.55	SIP	Status: 200 OK (1 bindings)
12 7.380855	10.192.51.55	10.192.51.55	SIP	Request: REGISTER sip:10.192.51.55
13 7.388182	10.192.51.55	10.192.51.55	SIP	Status: 100 Trying (1 bindings)
14 7.396983	10.192.51.55	10.192.51.55	SIP	Status: 200 OK (1 bindings)

Figure 33: Capture des paquets SIP de l'enregistrement des utilisateurs 77777 et 88888

Lorsque l'on veut émettre un appel par un des utilisateurs SIP dans X-lite, nous devons capter une trame de ***type 407*** pour dire que l'autorisation n'est plus donné par notre serveur Asterisk, mais plutôt par un autre serveur. Dans notre cas, le serveur d'autorisation est notre serveur ***FreeRadius d'adresse IP*** : ***10.192.51.62***.

43 136.178658	10.192.51.55	10.192.51.55	SIP/SDP	Request: INVITE sip:88888@10.192.51.55, with session descriptio
44 136.236925	10.192.51.55	10.192.51.55	SIP	Status: 407 Proxy Authentication Required
45 136.319911	10.192.51.55	10.192.51.55	SIP	Request: ACK sip:88888@10.192.51.55
46 136.513528	10.192.51.55	10.192.51.55	SIP/SDP	Request: INVITE sip:88888@10.192.51.55, with session descriptio
47 136.557504	10.192.51.55	10.192.51.55	SIP	Status: 100 Trying

Figure 34 : Capture des paquets SIP indiquant l'obligation de passer par un proxy

Après que l'utilisateur SIP a compris qu'il doit passer par un autre serveur d'autorisation, il reformule une autre demande d'invitation et il attend la réponse du serveur FreeRadius. Donc il y aura des échanges de ***paquets radius de type Accept*** entre notre client radius intégrée et notre FreeRadius. L'adresse IP ***10.192.51.55*** est celle de notre client radius dans Asterisk et l'adresse IP ***10.192.51.62*** est celle de notre serveur FreeRadius.

383 39.341491	10.192.51.55	10.192.51.62	RADIUS	Access-Request(1) (id=119, l=262)
393 40.857645	10.192.51.62	10.192.51.55	RADIUS	Access-Accept(2) (id=119, l=20)
501 56.467276	10.192.51.55	10.192.51.62	RADIUS	Accounting-Request(4) (id=83, l=419)
503 56.691874	10.192.51.62	10.192.51.55	RADIUS	Accounting-Response(5) (id=83, l=20)

Figure 35: Capture des paquets radius d'authentification sur la machine FreeRadius

Si l'utilisateur SIP est enregistré dans notre serveur d'autorisation avec un nom d'utilisateur et un mot de passe correcte, FreeRadius transmettra un message de type ***Access-Accept*** à notre client. Ce dernier envoie un message ***200 ok*** à l'utilisateur authentifié. Par la suite, le téléphone sonnera chez le destinataire. Voir la capture en bas :

205 841.395915	10.192.51.55	10.192.51.55	SIP	Status: 180 Ringing
206 841.413354	10.192.51.55	10.192.51.55	SIP	Status: 180 Ringing
213 854.206748	10.192.51.55	10.192.51.55	SIP/SDP	Status: 200 Ok. with session description

6.7.5.1 Test de fonctionnement de l'accounting avec radiusclient-1.4

Le test de l'accounting consiste à capter des paquets radius mais de type ***accounting***. Il y a deux types de paquets d'accounting :

- ***Accounting Request*** : est envoyé par le client radius au serveur FreeRadius. Ce paquet a la valeur du code ***4***. Il peut contenir des ***attributs Vendor (AVP)*** concernant le constructeur du client (car le client peut être un routeur cisco par exemple), le nom d'utilisateur, l'id de la session.
- ***Accounting Response*** : désigne l'accusé de la réception de l'accounting par FreeRadius et donne une réponse au client qu'il a bien reçu le paquet accounting. Le code du paquet a la valeur ***5***.

Le paquet ***d'Accounting Request*** se caractérise par ***Acct-specific Attributes*** pour indiquer la phase de l'accounting. Cet ***Acct-specific Attributes*** se trouve dans les attributs ***AVP*** du paquet radius. On trouve un champ ***Acct-Status-type*** qui donne des informations sur l'état de l'accounting. Ce champ est spécifié par un numéro d'attribut fixe de valeur ***40***, une longueur et une valeur. Cette valeur est l'origine de la détermination du statut de l'accounting. Il peut prendre que ces valeurs dans le tableau ci-dessous :

Valeur	Statut de l'accounting
1	Start
2	Stop
3	Interim-Update
7	Accounting-On

8	Accounting-Off
9-14	Réservé pour le tunnel accounting
15	Réservé pour un défaut d'accounting

Durant notre test de l'accounting, on a réussie à capter ces paquets d'accounting juste après que l'authentification a été validée par FreeRadius. Voici la figure 36 ci-dessous de la capture des paquets accounting enregistrés dans le sniffer lors de notre test (clientradius-1.4 a l'adresse IP ***10.192.51.55*** et le serveur FreeRadius a l'adresse ***IP 10.192.51.57***) :

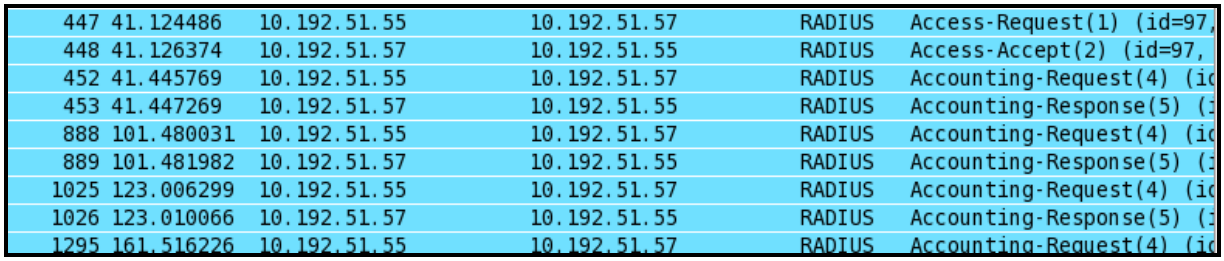


Figure 36: capture des paquets Accounting avec radiusclient-1.4 dans FreeRadius

A première vue, on peut croire que le fonctionnement de l'accounting dans FreeRadius est correct. Hélas, en analysant le paquet ***Accounting-Request***, nous avons rendu compte que le statut de notre accounting est de type ***Interim-Update*** (donc l'attribut ***Acct-Status-type*** contient la valeur ***3***). Tous ces paquets captés sont des updates car l'accounting n'arrive pas à passer au statut ***stop*** quand l'utilisateur se connecte d'Asterisk. Voir figure 37 ci-dessous :

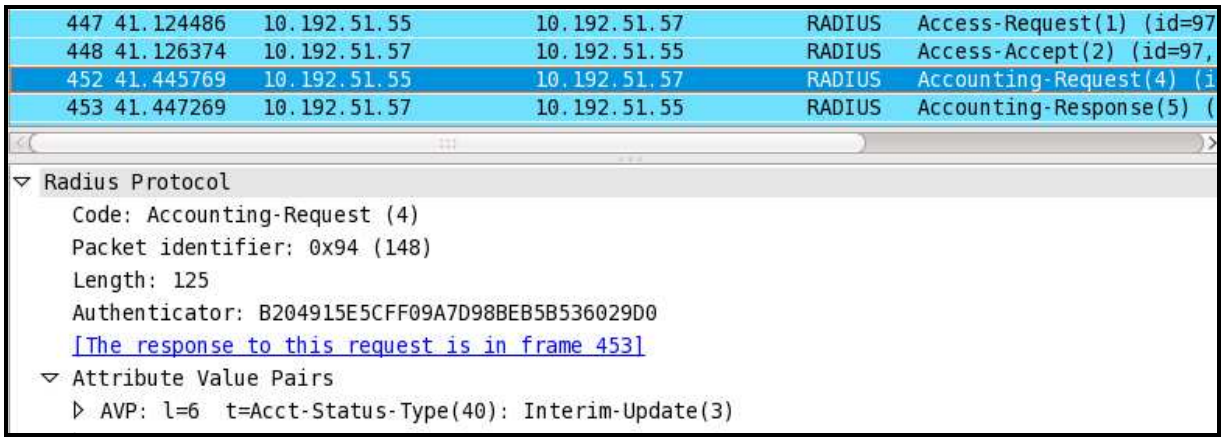


Figure 37: Paquet Accounting avec le statut Interim-Update

On a expliqué auparavant que l'accounting se base sur une variable de session comme le protocole http d'internet. Dans le cas de notre projet, on parle de session d'appel. Ici le problème que le client n'arrive pas envoyer la fin de la session de l'appel à FreeRadius. Si c'est le cas, FreeRadius ne réussira pas à enregistrer les bonnes valeurs de l'accounting comme la durée de la communication, la durée de la session d'appel…

En allant regardant le fichier de facturation dans FreeRadius de cet appel effectué se trouvant sur ce chemin sur notre machine ***/usr/local/var/log/radius/radacct/10.192.51.55/detail-date-du-jour*** pour vérifier cette hypothèse, notre supposition s'est avérée positive.

La figure 38 ci-dessous justifie la réponse à notre hypothèse sur l'enregistrement incorrect des paramètres de facturation sur FreeRadius à l'aide du script de notre client radiusclient-1.4 :

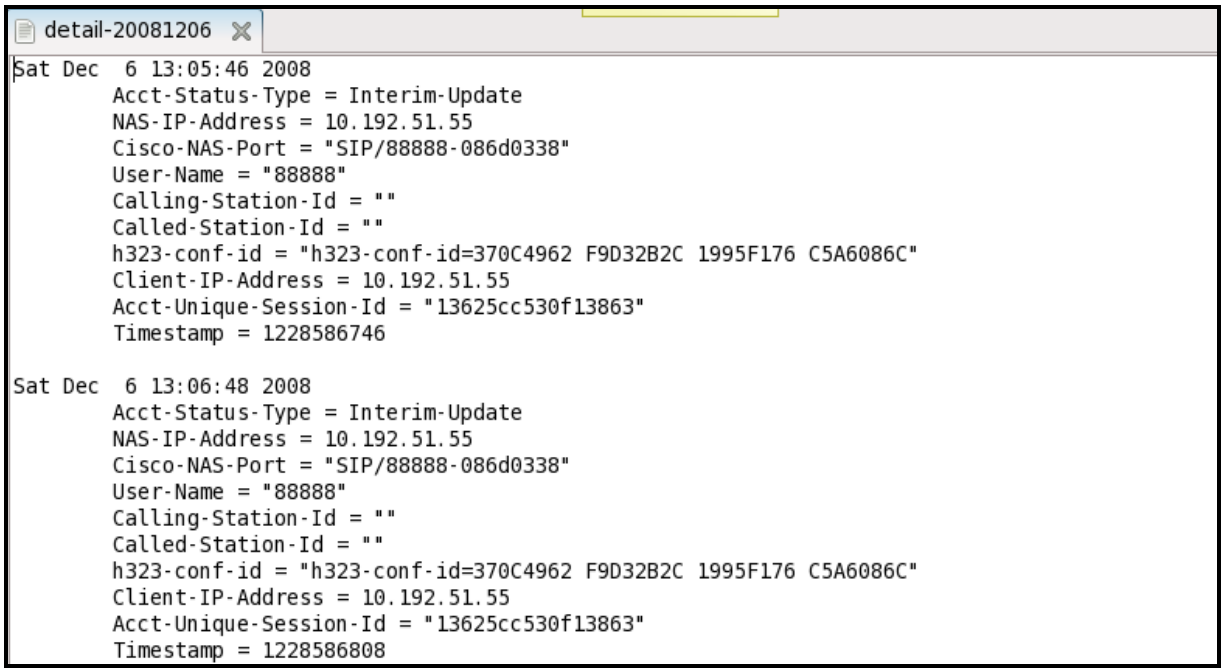


Figure 38: Résultat de l'enregistrement de l'accounting avec le statut Interim-Update

Pour remédier ce problème, la solution est de ne plus utiliser ce script d'accounting ***ast-rad-acc.pl*** avec Asterisk donc ne plus l'exécuter depuis la console. Comme l'ancien ***clientradius-ng*** fonctionne correctement et même parfaitement avec le principe d'accounting sous FreeRadius.

Le principe pour la résolution de notre bug est d'utiliser les deux clients ***radiusclient-1.4*** et l'autre ancien client ***radiusclient-ng ensemble***. C'est-à-dire d'utiliser le radiusclient-1.4 pour l'authentification des utilisateurs via radius, et l'autre sera utiliser pour l'accounting afin d'envoyer les bonnes paramètres CDR de facturation à notre serveur FreeRadius.

Nous avons été amenées à réinstaller et reconfigurer l'ancien client ***clientradius-ng*** sous Asterisk et FreeRadius.

Lors de la terminaison de cette implémentation de ces deux clients ensembles, nous avons commencé notre test d'authentification et d'accounting avec cette nouvelle plateforme. Le résultat s'est avéré positive car les utilisateurs s'authentifient correctement via FreeRadius. Et concernant l'accounting, nous avons réussis à capturer un paquet radius de ***type Accounting Request de type Stop*** (donc la valeur de ***l'attribut Acct-Status-Type est de 2 et n'est plus 3***) et un enregistrement des champs CDR correctement dans le fichier de facturation détail de FreeRadius.

Voir la figure 39 ci-dessous qui montre le résultat de l'accounting avec radiusclient-ng :

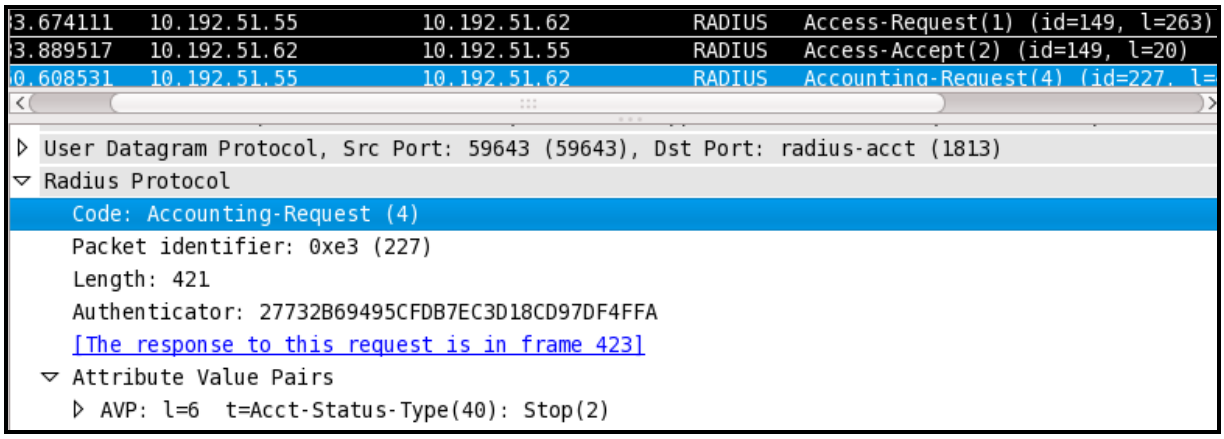


Figure 39: Capture des paquets accounting avec radiuclient-ng dans FreeRadius

Dans notre fichier texte détail de facturation dans FreeRadius concernant l'accounting, nous avons eu ces enregistrements de CDR de l'appel effectué lors de notre test avec la nouvelle plateforme réalisée. Voir figure 40 ci-dessous qui montre la réception des bonnes valeurs CDR de l'accounting grâce au client radiusclient-ng :

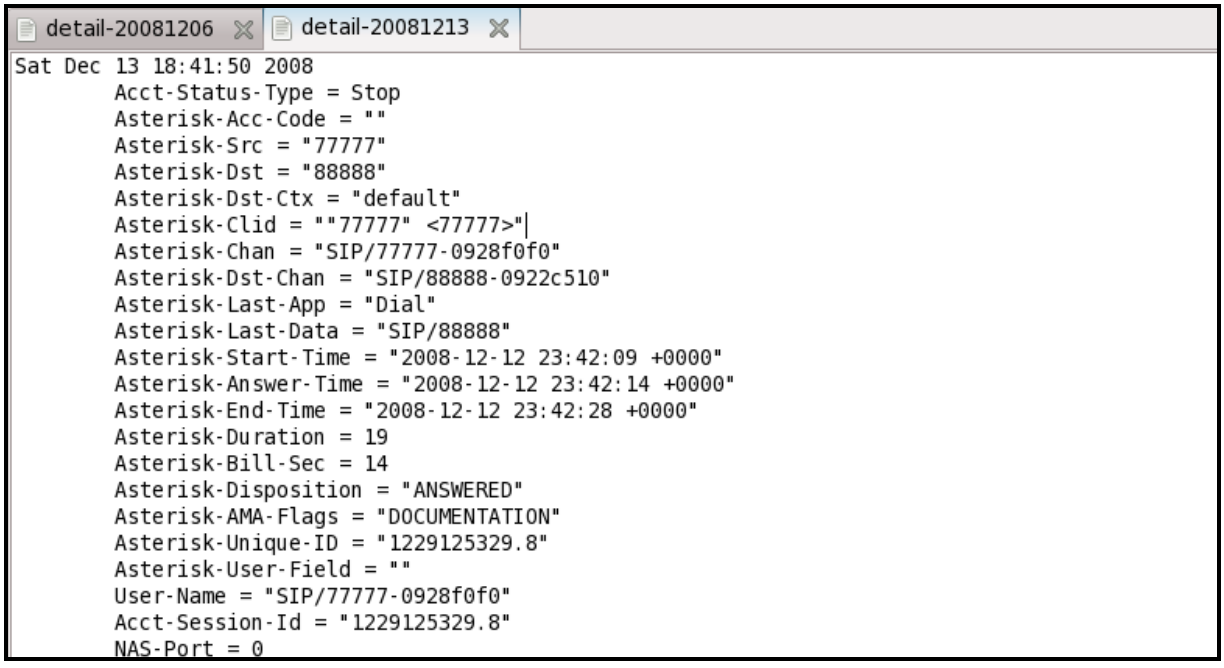


Figure 40: Enregistrement correcte des données de facturation à l'aide de clientradius-ng

7 AMLIORATIONS IMPORTES

7.1 Enregistrement du CDR (Accounting) dans une BD MYSQL

On a vu auparavant qu'Asterisk et FreeRadius stockent les informations de l'accounting sous forme de fichier texte. Ce format peut être nuisible à la lecture de ces informations surtout dans le fichier Master.csv dans Asterisk, qui est beaucoup mieux visible dans FreeRadius. Dans cette partie, nous allons essayer de centraliser ces données de façon efficace à l'aide des bases de données. Ceci dans le but de rendre l'exploitation et la gérance de ces données de facturation plus facile à l'utilisation et la manipulation.

7.1.1 CDR dans la BD MYSQL de FreeRadius

L'accounting a été enregistré au paravent dans un format de fichier sous FreeRadius. Et dans le chapitre MySQL et FreeRadius, nous avons parlé de la possibilité d'intégrer la BD de donnée dans notre serveur de facturation.

L'activation de l'enregistrement accounting se fait en décommettant ***sql*** dans le module accounting du fichier ***radius.conf*** de FreeRadius. Il faut vérifier si les requêtes SQL de l'accounting sont présentes dans le fichier ***sql.conf***.

Voici un aperçu des requêtes sql présentes dans le fichier de configuration sql.conf :

```
#######################################################################
#  Accounting Queries
#######################################################################
# accounting_onoff_query        - query for Accounting On/Off packets
# accounting_update_query       - query for Accounting update packets
# accounting_update_query_alt   - query for Accounting update packets
#                                (alternate in case first query fails)
# accounting_start_query        - query for Accounting start packets
# accounting_start_query_alt    - query for Accounting start packets
#                                (alternate in case first query fails)
# accounting_stop_query         - query for Accounting stop packets
# accounting_stop_query_alt     - query for Accounting start packets
#                                (alternate in case first query doesn't
#                                 affect any existing rows in the table)
#######################################################################
```

Figure 41: Requêtes SQL de l'accounting présentes dans le fichier sql.conf de FreeRadius

Pour la partie SQL, nous avions déjà importé ce fichier ***/usr/local/share/doc/freeradius/examples /mysql.sql***. Dans cette importation, on trouve la table ***radacct*** pour stocker tous les détails de l'accounting. Elle contient des informations sur la session de connexion et de la déconnexion d'un client radius. Nous enregistrons donc principalement à quel moment a eu la connexion du client radius, une variable de session, le numéro du client, l'adresse IP du client (NAS), son port, le temps de la déconnexion, la durée de la session…

Voir figure ci-dessous pour avoir une idée générale sur les informations de l'accounting :

```
mysql> describe radacct;
+----------------------+-------------+------+-----+---------------------+-------
---------+
| Field                | Type        | Null | Key | Default             | Extra
        |
+----------------------+-------------+------+-----+---------------------+-------
---------+
| RadAcctId            | bigint(21)  | NO   | PRI | NULL                | auto_i
ncrement |
| AcctSessionId        | varchar(32) | NO   | MUL |                     |
        |
| AcctUniqueId         | varchar(32) | NO   | MUL |                     |
        |
| UserName             | varchar(64) | NO   | MUL |                     |
        |
| Realm                | varchar(64) | YES  |     |                     |
        |
| NASIPAddress         | varchar(15) | NO   | MUL |                     |
        |
| NASPortId            | varchar(15) | YES  |     | NULL                |
        |
| NASPortType          | varchar(32) | YES  |     | NULL                |
        |
| AcctStartTime        | datetime    | NO   | MUL | 0000-00-00 00:00:00 |
        |
| AcctStopTime         | datetime    | NO   | MUL | 0000-00-00 00:00:00 |
        |
| AcctSessionTime      | int(12)     | YES  |     | NULL                |
        |
```

Figure 42: Table radacct de l'accounting de FreeRadius

Dans cette table d'accounting, l'enregistrement de session des clients aura cet aspect :

RadAcctId	AcctSessionId	AcctUniqueId	UserName	Realm	NASIPAddress	NA
1	1226589559.2	39ee376980c04651	SIP/user1-0a1be918		127.0.0.1	0
2	1226590236.4	c5dbb229e38673d4	SIP/user2-0a1be918		127.0.0.1	0
3	1226593427.6	3ff47b57828d52d5	SIP/user1-0a1b9210		127.0.0.1	0
4	1226606755.8	2df90db207ce0337	SIP/user2-0a1bfba8		127.0.0.1	0
5	1226608000.10	61a110feab95853a	SIP/user1-0a1bfba8		127.0.0.1	0
6	1226612799.14	c4a48eb466ce5231	SIP/user2-0a1cf1f0		127.0.0.1	0
7	1226616479.16	4dfca02491f873ce	SIP/user1-0a1cf1f0		127.0.0.1	0
8	1226618237.18	733df45ede874516	SIP/user2-0a1cf1f0		127.0.0.1	0
9	1226620981.0	94f4013e549a2d92	SIP/user1-082f9620		127.0.0.1	0

7.1.2 CDR dans la BD MYSQL d'Asterisk

Pour ce faire, nous avons besoin d'installer le package ***asterisk-addons***. Or par chance, ceci a été installé auparavant dont le but est d'envoyer l'accounting vers FreeRadius. Il ne faut pas oublier bien sûr d'avoir un serveur MYSQL installé sur la même machine. Nous allons citer les étapes de la réalisation de cette configuration : [19]

1. Se connecter au serveur MYSQL

2. Création d'une base de données ***asterisk***.

   ✓ ***create database asterisk ;***

3. Créer un utilisateur pour donner tous les droits de modifier et gérer cette base de données.

   ✓ ***GRANT ALL ON asterisk.* to selim@localhost IDENDIFIED BY 'Password' ;***

4. Ensuite, il faut créer la table CDR dans cette base de données asterisk. on peut le faire en ligne de commande sql par create table. Mais il y a un moyen plus simple est de l'importer à l'aide d'un fichier sql. Dans le dossier où il est installé ce package ***asterisk-addons*** sur notre machine, il y a un fichier texte contenant la structure sql de cette table. Dans notre cas, ce fichier se trouve sur ce chemin ***root/Download /asterisk-addons-1.4.2/doc/cdr_mysql.txt***. Par la suite, nous changeons l'extensions de ce fichier en type sql et on tape cette commande :

   ✓ ***mysql –u root –p asterisk < cdr_mysql.sql;***

5. La structure de cette table CDR est comme suit:

```
CREATE TABLE cdr (
  calldate datetime NOT NULL default '0000-00-00 00:00:00',
  clid varchar(80) NOT NULL default '',
  src varchar(80) NOT NULL default '',
  dst varchar(80) NOT NULL default '',
  dcontext varchar(80) NOT NULL default '',
  channel varchar(80) NOT NULL default '',
  dstchannel varchar(80) NOT NULL default '',
  lastapp varchar(80) NOT NULL default '',
  lastdata varchar(80) NOT NULL default '',
  duration int(11) NOT NULL default '0',
  billsec int(11) NOT NULL default '0',
  disposition varchar(45) NOT NULL default '',
  amaflags int(11) NOT NULL default '0',
  accountcode varchar(20) NOT NULL default '',
  uniqueid varchar(32) NOT NULL default '',
  userfield varchar(255) NOT NULL default ''
)TYPE=MyISAM;

ALTER TABLE cdr ADD INDEX (calldate);
ALTER TABLE cdr ADD INDEX (dst);
ALTER TABLE cdr ADD INDEX (accountcode);
```

Figure 43: Structure SQL de la table CDR sous Asterisk

6. La structure SQL des CDRs est maintenant crée. Il nous reste qu'a modifier les fichiers de configurations d'Asterisk pour implémenter le stockage des données de

facturation dans cette base de données ***asterisk***. Ces fichiers de configuration se trouvent sur ce chemin ***etc/asterisk***.

7. En premier lieu, nous allons modifier le fichier ***etc/asterisk module.conf*** pour ajouter le module cdr. Nous devons ajouter cette ligne dans ce fichier comme ceci :

   - ✓ ***load => cdr_addon_mysql.so***

8. En deuxième lieu, nous avons changé le fichier ***etc/asterisk /cdr_mysql.conf***. Cette modification permet de définir les paramètres de connexion au serveur MySQL par Asterisk. Par exemple nous avons spécifié le nom de la base de données contenant la table CDR, le nom d'utilisateur et le mot de passe pour la connexion à MYSQL.

```
[global]
hostname=localhost
dbname=asterisk
table=cdr
password=root
user=asterisk
port=3306
sock=/var/lib/mysql/mysql.sock
;userfield=1
```

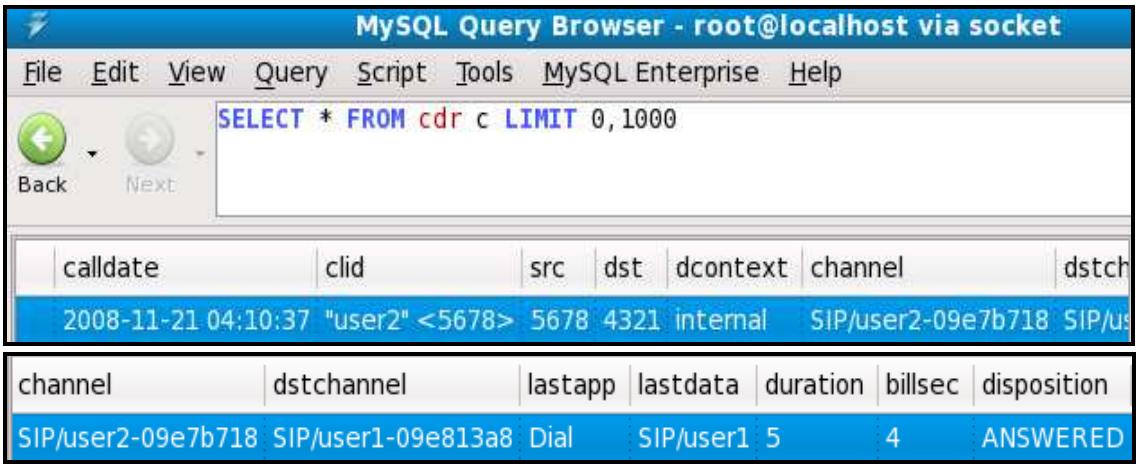
9. Pour le socket MYSQL, il faut chercher le chemin exact dans le fichier ***etc/my.cnf*** sur la machine.

10. En troisième lieu, nous redémarrons Asterisk pour qu'il prenne en charge l'ajout du module ***cdr_addon_mysql.so*** et les changements des fichiers de configuration.

11. Pour finir, nous pouvons faire un test d'appel entre deux utilisateurs SIP à l'aide de X-lite et s'il y a eu enregistrement des paramètres CDR dans notre table ***cdr*** de la BD ***asterisk***. Dans le fichier debug d'Asterisk se trouvant sur le chemin ***var/log/asterisk*/debug**, nous pouvons voir la requête d'insertion sql des CDRs.
    Voir figure ci-dessous :

```
[Nov 21 04:01:55] DEBUG[28558] cdr_addon_mysql.c: cdr_mysql: inserting a CDR record.
[Nov 21 04:01:55] DEBUG[28558] cdr_addon_mysql.c: cdr_mysql: SQL command as follows: INSERT
INTO cdr
(calldate,clid,src,dst,dcontext,channel,dstchannel,lastapp,lastdata,duration,billsec,disposition,
VALUES ('2008-11-21 04:01:34','\"user1\" <4321>','4321','5678','internal', 'SIP/
user1-08d3c928','SIP/user2-08d42b78','Dial','SIP/user2',21,19,'ANSWERED',3,'')
[Nov 21 04:06:59] DEBUG[28532] http.c: Nothing changed in http
[Nov 21 04:06:59] DEBUG[28532] pbx_dundi.c: Seeding global EID '00:0c:29:9e:ab:8c' from 'eth1'
using 'siocgifhwaddr'
[Nov 21 04:06:59] DEBUG[28532] cdr_csv.c: logging time in GMT
[Nov 21 04:06:59] DEBUG[28532] cdr_csv.c: logging CDR field UNIQUEID
[Nov 21 04:06:59] DEBUG[28532] cdr_csv.c: logging CDR user-defined field
```

Nous pouvons aussi vérifier cette insertion via l'interface sql par la commande ***select*** après la terminaison de l'appel effectué. Voir figure ci-dessous :



7.2 Enregistrement des utilisateurs SIP dans une BD d'Asterisk

Précédemment, nous avons enregistré les utilisateurs SIP dans notre fichier de configuration sip.conf. Il y a un autre moyen plus élégant, est de les stocker dans une BD MYSQL. Dans cette partie, nous avons expliqué comment réaliser ce stockage des utilisateurs SIP sur MySQL. [20]

7.2.1 Enregistrement des utilisateurs SIP dans une BD locale

Asterisk utilise une base de données spéciale de type ***AstDB*** propre à lui. Cet AstDB fournit un moyen simple de stocker des données à utiliser dans notre plan de numérotation. Cette base de données stocke ses données dans des groupements appelés familles avec les valeurs identifiées par des clés. Asterisk offre aussi un mécanisme de gestion de familles des utilisateurs par la fonction ***Set ()*** dans l'instruction ***exten =>*** du fichier de configuration ***extensions.conf***. Par défaut, cette base de données est activée et Asterisk stocke une trace des utilisateurs automatiquement qui s'enregistre dans un fichier ***Astdb*** sur ce chemin ***/var/lib/asterisk/astdb***.

Voilà une capture des enregistrements SIP dans la BD locale ***AstDB*** d'Astrerisk :

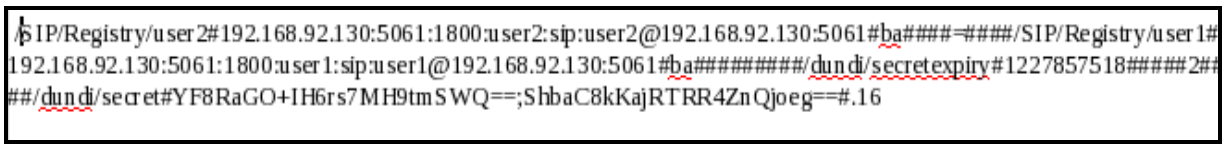


Figure 44: Enregistrement des utilisateurs SIP dans la BD locale Astdb

Le but de notre projet est d'avoir une trace locale des utilisateurs SIP et non pas d'être un expert de la BD d'AstDB. Cette fonctionnalité offre beaucoup de possibilité pour gérer les utilisateurs. Une des possibilités est l'application ***LookupBlacklist()*** qui vérifie si le numéro de l'appelant existe dans la liste noire. Cette liste est simplement une famille appelée ***blacklist*** dans ***AstDB***. Si un numéro est dans cette liste noire, par exemple Asterisk envoie un message qu'il est dans cette

liste et diffuse un audio d'attente. L'insertion d'un numéro dans la liste noire se fait par la commande ***database put blackliste numéro téléphone***.

7.2.2 Enregistrement des utilisateurs SIP dans une BD MYSQL

Asterisk ne va plus chercher les utilisateurs dans le fichier ***sip.conf***, mais il va envoyer des requêtes SQL pour trouver ces utilisateurs SIP. Ceci permet de centraliser les données et de ne plus taper chaque fois la commande ***sip reload*** dans la console d'Asterisk pour prendre en considération l'ajout d'un nouvel utilisateur SIP. Avec le stockage MYSQL des utilisateurs, on ne passe plus par les fichiers de configurations d'Asterisk, on traite les données des utilisateurs par une interface MYSQL, et plus besoin de taper une commande dans la console CLI d'Asterisk.

Ce type d'enregistrement des fichiers de configuration d'Asterisk est fait grâce à sa base de données ***Realtime ARA (Asterisk Relatime Architecture)***. Cette architecture ***Realtime*** permet de stocker des enregistrements dans une base de données de type SQL. Elle peut stocker aussi toutes les informations des fichiers de configuration comme ***extensions.conf***, ***voicemail.conf***, ***sip.conf***… Il peut être utilisé avec d'autres bases de données que MYSQL par exemple ***Postgress***…

Les avantages de développement de cette architecture ARA avec notre serveur Asterisk sont :

- Une meilleure lisibilité des données et d'une meilleure mise à jour.
- Eviter de reloader les changements de configurations statiques en ligne de commande.
- Externalisation de la base de données sur un autre serveur différent de notre Asterisk.

L'utilisation de cette technologie demande qu'un serveur MYSQL soit installé ainsi que le package ***asterisk-addons*** soit installé.

Les procédures d'installation sont :

1. Créer une base de données MYSQL asterisk.
   - ✓ ***CREATE DATABASE asterisk*** ;
2. Importer le schéma de la table ***sipfriends*** dans la BD ***asterisk***. Voir annexe E pour plus de détail du fichier SQL de la table des enregistrements SIP.

```
[root@localhost ~]# mysql -u root -p asterisk < sip_mysql.sql
Enter password:
```

Voir figure 45 de la structure de cette table sipfriends :

```
#
# Table structure for table `sipfriends`
#
CREATE TABLE sipfriends(
  name varchar(40) NOT NULL default '',
  type varchar(10) NOT NULL default '',
  username varchar(40),
  fromuser varchar(40),
  fromdomain varchar(40),
  secret varchar(40),
  md5secret varchar(40),
  auth varchar(10),
  mailbox varchar(20),
  subscribemwi varchar(10), -- yes/no
  vmexten varchar(20),
  callerid varchar(40),
  cid_number varchar(40),
  callingpres varchar(20),
  usereqphone varchar(10),
  language varchar(10),
  incominglimit varchar(10),
  context varchar(40) NOT NULL default '',
```

Figure 45: table sipfriends de la BD asterisk

3. Copier le fichier ***res_mysql.so*** dans le répertoire des modules d'Asterisk ***/usr/lib/asterisk/modules***.

4. Editer le fichier ***/etc/asterisk/modules.conf*** et ajouter cette ligne :

   ✓ ***load => res_config_mysql.so***

5. Editer le fichier ***etc/asterisk/res_mysql.conf*** pour paramétrer les variables nécessaires pour la connexion à la BD ***asterisk***. Voir capture ci-dessous :

```
[general]
dbhost = 127.0.0.1
dbname = asterisk
dbuser = root
dbpass = root
dbport = 3306
dbsock = var/lib/mysql/mysql.sock
```

6. Editer le fichier ***/etc/asterisk/extconfig*** d'Asterisk. Dans ce fichier, on spécifie la famille des utilisateurs SIP, quel driver à utiliser (MySQL, odbc, postgress…), le nom de la base de données et la table.

   ✓ ***Nom de la famille => driver, nom de la base de données, nom de la table***

Voici une capture de notre fichier de configuration ***extconfig.conf***

```
sipusers => mysql,asterisk,sipfriends
sippeers => mysql,asterisk,sipfriends
```

7. On teste la connexion de ***Realtime*** avec notre serveur MySQL en tapant cette commande ***realtime mysql status*** dans la console CLI d'Asterisk. Voir capture en bas :

```
localhost*CLI> realtime mysql status
Connected to asterisk@127.0.0.1, port 3306 with username root for 1 minutes, 24
seconds.
[Dec  8 06:15:16] DEBUG[3927]: res_config_mysql.c:650 mysql_reconnect: MySQL Rea
lTime: Everything is fine.
localhost*CLI>
```

Figure 46: statust de la connexion de Realtime avec le serveur MYSQL

8. Maintenant que la connexion au serveur MySQL fonctionne correctement d'après la figure 46 ci-dessus, nous pouvons ajouter autant d'utilisateur SIP selon nos besoins à l'aide de la commande SQL insert into dans la table sipfrineds.

   Voir figure 47 ci-dessous :

```
mysql> insert into sipfriends (name,type,username,secret,regexten,callerid,conte
xt) values ('77777','friend','77777','test','77777','<77777>','sip_auth');
Query OK, 1 row affected (0.00 sec)

mysql> insert into sipfriends (name,type,username,secret,callerid,context) value
s ('88888','friend','88888','test2','<88888>','sip_auth');
Query OK, 1 row affected (0.02 sec)
```

Figure 47: insertion des utilisateurs SIP dans la table sipfriends sous MYSQL

7.3 Réplication de la BD Asterisk des utilisateurs SIP sur MYSQL

La réplication d'une base de donnée MySQL est une technique permettant de copier à l'identique les données présentes sur un serveur (**le serveur maître)** vers un autre serveur **(le serveur esclave)**.
La réplication consiste à configurer un maître et un esclave en SQL pour communiquer entre eux afin d'envoyer des requêtes. Le serveur maître garde les traces de toutes les modifications réalisées su une BD dans un fichier log binaire (par exemple insertion, suppression..). Le serveur esclave essaye de se connecter au serveur maître pour que ce dernier lui envoie les dernières modifications. La communication entre eux est unidirectionnelle, donc c'est toujours le maître qui envoie des données à l'esclave et le sens contraire est impossible.

Notre objectif est de répliquer la BD de données MYSQL d'Asterisk sur un autre serveur MYSQL distant. Notre but est d'externaliser les données d'Asterisk dans un serveur pour avoir une seconde sauvegarde de nos utilisateurs SIP. Dans le cas d'un problème de notre serveur Asterisk, un autre serveur peut prendre le relais par exemple. Dans cette partie, nous détaillerons les configurations nécessaires pour réaliser cette réplication de notre BD ***asterisk*** de nos utilisateurs SIP d'Asterisk sur une autre machine MYSQL.
On commence par configurer le maître. Ce dernier est notre machine Asterisk qui contient la BD **asterisk** à répliquer vers l'esclave. Voici les étapes de la réplication [21] :

1. Créer un utilisateur sur le maître pour donner les droits de la réplication ainsi que d'autres actions. Voir figure ci-dessous :

```
mysql> GRANT REPLICATION SLAVE, SELECT, SUPER, RELOAD ON *.* To selim@'%' IDENTI
FIED BY 'hello';
Query OK, 0 rows affected (0.09 sec)

mysql> FLUSH TABLES WITH READ LOCK;
Query OK, 0 rows affected (0.01 sec)

mysql> exit
```

Figure 48: Création d'un utilisateur de la réplication sur le maître

2. On décharge toutes les tables et les blocs avec la commande **FLUSH TABLES** pour les tables de type **MyISAM** :

   - ✓ **FLUSH TABLES WITH READ LOCK**

3. On crée une copie de la base de données **asterisk** des enregistrements SIP pour la copier par la suite sur notre serveur esclave. Ceci peut être fait à l'aide d'un fichier compressé par la commande tar et on peut spécifier qu'on compresse seulement la BD à copier. Voir la capture ci-dessous pour la compression d'une copie de la BD ***asterisk*** à importer :

```
[root@localhost mysql]# tar -cvf /tmp/mysql-repl.tar ./asterisk
./asterisk/
./asterisk/sipfriends.MYI
./asterisk/sipfriends.MYD
./asterisk/cdr.MYI
./asterisk/db.opt
./asterisk/cdr.MYD
./asterisk/sipfriends.frm
./asterisk/cdr.frm
```

4. Avant d'enlever le verrou de la lecture, on tape cette commande **SHOW MASTER STATUS** pour avoir les valeurs du fichier log utilisé par le maître et son offset. Ces informations vont être utiles lors de la configuration de l'esclave, donc il faut les enregistrer provisoirement. Dans notre cas le résultat n'affiche rien car il n'utilise pas de fichier log. Ce cas spécial se configure sur l'esclave en mettant la valeur ***log-bin*** qui est vide et la valeur de son offset à ***4***. Voir la capture ci-dessous :

```
mysql> FLUSH TABLES WITH READ LOCK;
Query OK, 0 rows affected (0.00 sec)

mysql> show master status;
Empty set (0.09 sec)

mysql> show master status;
Empty set (0.00 sec)

mysql> UNLOCK TABLES;
Query OK, 0 rows affected (0.16 sec)

mysql> exit
```

5. On réactive l'activité du maître en levant le verrou de lecture par cette commande :

   ✓ **UNLOCK TABLES**

6. Editer le fichier de configuration de MYSQL/***ect/my.cnf***. On ajoute deux paramètres dont un est le fichier log et l'autre est l'indice du serveur maître. Cet indice doit être toujours égal à 1 pour le maître.

```
[mysqld]

Log-bin
Server-id=1
```

Maintenant, nous pouvons passer à la **configuration du serveur esclave**. Ici, nous avons besoin :

1. Copier le fichier compressé et le décompresser dans le dossier de données de MYSQL sur notre machine esclave.

2. On arrête le serveur MYSQL esclave et nous éditons le fichier **/ect/my.cnf** pour insérer une valeur id du serveur.

```
[mysqld]

Server-id = 2
```

3. On tape la commande **CHANGE MASTER TO** de MYSQL pour paramétrer les valeurs de la connexion de l'esclave au maître. Ces paramètres sont : l'adresse IP du maître, le nom d'utilisateur et le mot de passe de la réplication dans le maître, le fichier log-bin du maître et la valeur de l'offset. Dans notre cas, les valeurs paramétrées sont celles-ci :

```
mysql> CHANGE MASTER TO
    -> MASTER_HOST='10.192.51.55',
    -> MASTER_USER='selim',
    -> MASTER_PASSWORD='hello',
    -> MASTER_LOG_FILE='',
    -> MASTER_LOG_POS=4;
Query OK, 0 rows affected (0.06 sec)

mysql>
```

4. On lance les threads de l'esclave par cette commande SQL :

   - ✓ **START SLAVE**

5. Pour tester si tout fonctionne bien, on lance ces 2 commandes SQL :

   - ✓ **SHOW SLAVE STATUS ;**
   - ✓ **SHOW PROCESSLIST ;**

8 TEST DE L'APPLICATION AVEC LES AMELIORATIONS IMPORTES

Nous allons tester le fonctionnement de l'authentification et l'accounting dans FreeRadius et Asterisk en tenant compte es améliorations réalisées et de l'implémentation du clientradius-ng en parallèle avec radiusclient-1.4.

L'architecture de notre projet est composé de :

- **User agent X-lite** (appelé et appelant) sur la machine d'adresse IP **10.192.51.55**
- **Serveur Asterisk** d'adresse IP **101.92.51.55** et un **serveur** MYSQL **maître** intégré.
- **NAS** ou client radius (**radiusclient-ng et radiusclient-1.4**) d'adresse IP **10.192.51.55**
- **Serveur FreeRadius** d'adresse IP **10.192.51.62** et un **serveur MYSQL** intégré.
- **Serveur** MYSQL **esclave** d'adresse IP **10.192.51.92**

La figure 49 ci-dessous donne une idée sur l'infrastructure réalisée et les paquets échangés entre les différents composants dans notre projet :

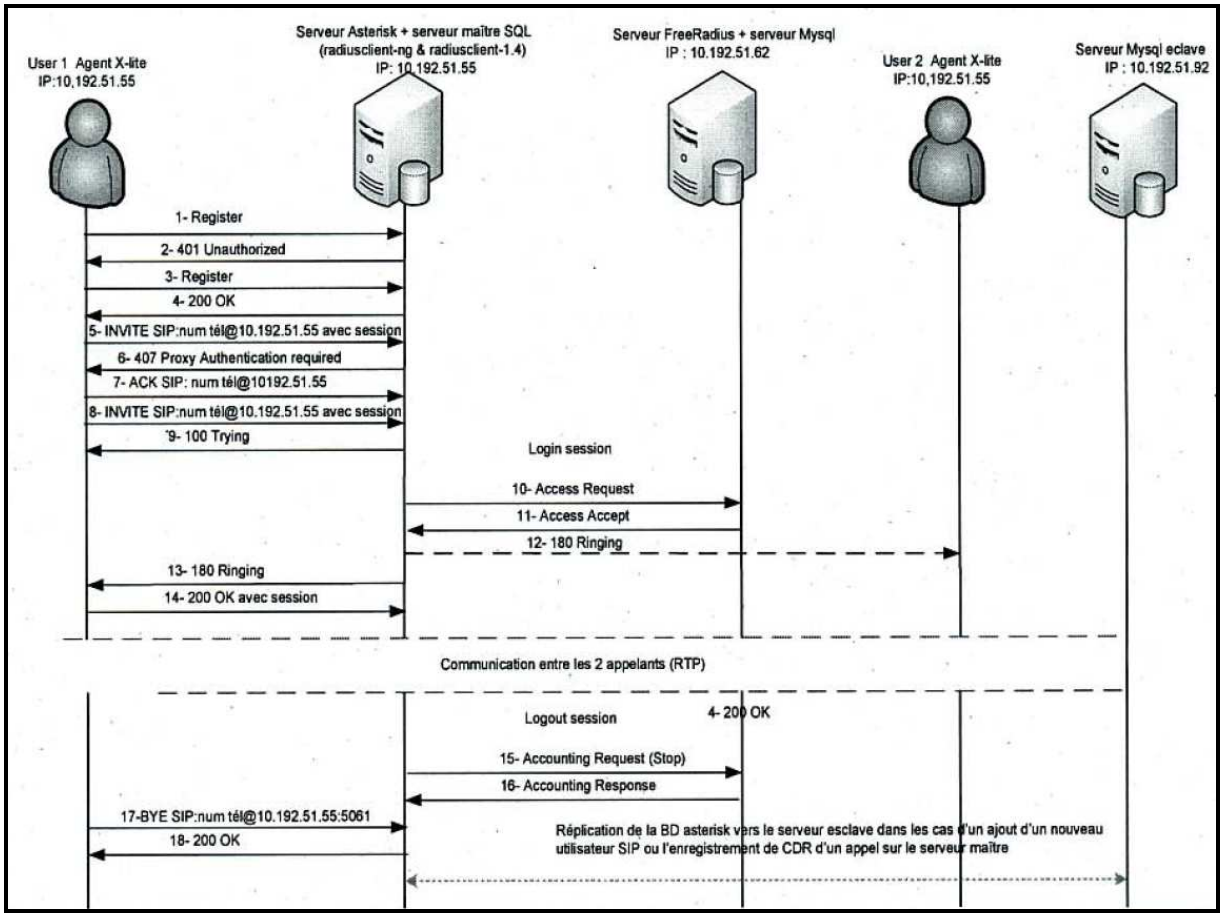


Figure 49: Architecture et diagramme en fléche de l'infrastructure installée dans notre projet

8.1 Test final du mécanisme d'authentification dans notre application

D'après le diagramme en flèche ci-dessus, l'utilisateur SIP de X-lite s'enregistre via Asterisk à l'aide de son numéro de téléphone et un mot de passe. S'il s'est authentifié correctement, un message **Logged** in sur la fenêtre principale de X-lite, et si ce n'est pas le cas, un message **Logged failed out** s'affichera.

On a deux utilisateurs SIP enregistrés dans notre base de données asterisk dans la table ***sipfriends*** d'Asterisk. Ces deux utilisateurs vont faire preuve d'un test d'authentification via radius. Ces utilisateurs sont :

77777	friend	77777	test	0.0.0.0	NULL	0	dynamic	1229311699	777
88888	friend	88888	test2	0.0.0.0	NULL	0	dynamic	1229311699	NULL

En suivant le diagramme en flèche ci-dessus pour l'enregistrement par exemple de **l'utilisateur SIP** portant le numéro de téléphone **77777**, on obtiendra **les paquets SIP de l'étape 1 à 4.**

Voici un aperçu des paquets obtenus de l'utilisateur SIP ***77777*** sur le sniffer lors de son enregistrement :

752	86.842408	10.192.51.55	10.192.51.55	SIP	Request: REGISTER sip:10.192.
753	86.849271	10.192.51.55	10.192.51.55	SIP	Status: 100 Trying (1 binc
754	86.850165	10.192.51.55	10.192.51.55	SIP	Status: 401 Unauthorized
755	86.871810	10.192.51.55	10.192.51.55	SIP	Request: REGISTER sip:10.192.
756	86.873145	10.192.51.55	10.192.51.55	SIP	Status: 100 Trying (1 binc
764	86.899296	10.192.51.55	10.192.51.55	SIP	Status: 200 OK (1 binding:
765	86.901924	10.192.51.55	10.192.51.55	SIP	Request: REGISTER sip:10.192.
766	86.903082	10.192.51.55	10.192.51.55	SIP	Status: 100 Trying (1 binc

Figure 50: Paquets SIP de l'enregistrement de l'utilisateur 77777 sous Asterisk

Un autre aperçu du contenu des ces paquets SIP concernant de l'enregistrement de l'utilisateur ***77777*** sur la console CLI d'Asterisk :

REGISTER
sip:10.192.51.55
SIP/2.0
Via:
SIP/2.0/UDP
10.192.51.55:5061;rport;branch=z9hG4bK430B7A7EBD1D92A653AAB94B9C703050
From:77777<sip:77777@10.192.51.55>;tag=1623834547 To:77777
<sip:77777@10.192.51.55> Contact: "77777" <sip:77777@10.192.51.55:5061> Call-ID: 54E0379C6CDD24A4CC8A3774595D79FE@10.192.51.55 CSeq: 48959 REGISTER Expires: 1800 Max-Forwards: 70 User-Agent: X-Lite release 1105d Content-Length: 0

L'user agent reçoit un paquet standard de la part d'Asterisk avec le message ***401***. C'est l'étape ***2*** :

SIP/2.0
401 Unauthorized
v:SIP/2.0/UDP
10.192.51.55:5061;branch=z9hG4bK430B7A7EBD1D92A653AAB94B9C703050;
received=10.192.51.55;rport=5061
f: 77777 <sip:77777@10.192.51.55>;tag=1623834547
t: 77777 <sip:77777@10.192.51.55>;tag=as77ea04ae
i: 54E0379C6CDD24A4CC8A3774595D79FE@10.192.51.55
CSeq: 48959 REGISTER
User-Agent: Asterisk PBX
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY
k:replaces WWW-Authenticate: Digest algorithm=MD5, realm="asterisk",
nonce="2710c3ea" l: 0

Ce paquet correspond à l'étape **3** de notre diagramme, l'utilisateur renvoie un message Register SIP à l'aide des paramètres Digest par exemple realm, uri, response… Voir à la fin de la figure ci-dessous :

REGISTER
sip:10.192.51.55 SIP/2.0
Via: SIP/2.0/UDP 10.192.51.55:5061;rport;branch=z9hG4bK5FDEEA8281BDDEA49
F8A75C0D67E4CE9
From: 77777 <sip:77777@10.192.51.55>;tag=1623834547
To: 77777 <sip:77777@10.192.51.55>
Contact: "77777" <sip:77777@10.192.51.55:5061>
Call-ID: 54E0379C6CDD24A4CC8A3774595D79FE@10.192.51.55
CSeq: 48960 REGISTER Expires: 1800
Authorization:Digest
username="77777",realm="asterisk",nonce="2710c3ea",response="e352027a76fdeb31e76fb1
99802a80dc",uri="sip:10.192.51.55",algorithm=MD5 Max-Forwards: 70User-Agent: X-Lite
release 1105d Content-Length: 0

Comme notre utilisateur est enregistré dans la base de données MYSQL **asterisk** à l'aide de **Realtime**. Asterisk va interroger la table **sipfriends** pour chercher l'utilisateur ***77777*** afin de l'authentifier. Donc il va exécuter une commande SQL de type update pour mettre à jour les données de cet utilisateur. Voir figure ci-dessous de la console CLI debug d'Astersik :

- Saved useragent "X-Lite release 1105d" for peer 77777

[Dec 14 21:15:37]
DEBUG[2599]: res_config_mysql.c:650 mysql_reconnect: MySQL RealTime: Everything is fine.

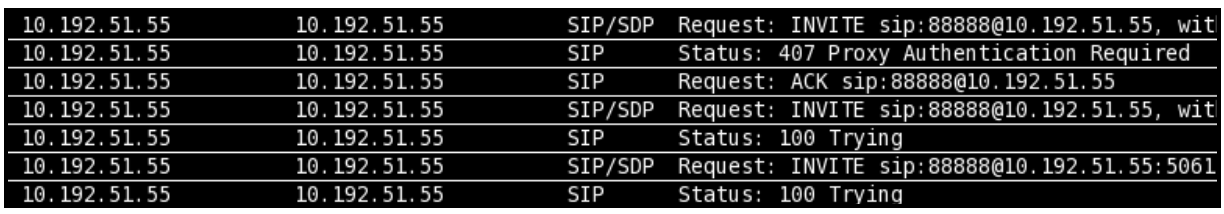
[Dec 14 21:15:37] DEBUG[2599]: res_config_mysql.c:354 update_mysql: MySQL RealTime: Update SQL: UPDATE sipfriends SET ipaddr = '10.192.51.55', port = '5061', regseconds = '1229309137', username = '77777', fullcontact = 'sip:77777@10.192.51.55:5061' WHERE name = '77777'

[Dec 14 21:15:37]
DEBUG[2599]: res_config_mysql.c:368 update_mysql: MySQL RealTime: Updated 1 rows on table: sipfriends

Si tout va bien, nous recevrons un paquet **200 Ok** de l'étape **4** de notre diagramme ci-dessus.
Voilà le résultat dans notre console CLI :

SIP/2.0 200 OK
v:SIP/2.0/UDP
10.192.51.55:5061;branch=z9hG4bK5FDEEA8281BDDEA49F8A75C0D67E4CE9;
received=10.192.51.55;rport=5061
f: 77777 <sip:77777@10.192.51.55>;tag=1623834547
t: 77777 <sip:77777@10.192.51.55>;tag=as77ea04ae
i: 54E0379C6CDD24A4CC8A3774595D79FE@10.192.51.55
CSeq: 48960 REGISTER
User-Agent: Asterisk PBX
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY
k: replaces
expires: 1800
m: <sip:77777@10.192.51.55:5061>;expires=1800
Date: Mon, 15 Dec 2008 02:15:37 GMT

Après que l'utilisateur 77777 s'est enregistré avec succès dans Asterisk, il doit s'authentifier via notre serveur radius pour effectuer un appel.
L'utilisateur 77777 va tenter de composer un numéro **88888** d'un **autre utilisateur SIP** pour effectuer un appel avec la commande **INVITE sip : 88888@101.192.51.55 : 5061**. Asterisk lui renvoi un message **407** qu'il doit passer par un proxy d'authentification car Asterisk n'a pas le droit de traiter cette autorisation. Il répond par un accusé de réception de ce message **ACK**. Alors l'utilisateur 77777 renvoi une autre demande **INVITE** avec une variable de session comme une connexion sur un site internet. Ces échanges correspondent à **l'étape 5 jusqu'à 9** de notre diagramme ci-dessus. Voici une capture ci-dessous sur notre sniffer de l'étape 5 à 9 :



10.192.51.55	10.192.51.55	SIP/SDP	Request: INVITE sip:88888@10.192.51.55, wit
10.192.51.55	10.192.51.55	SIP	Status: 407 Proxy Authentication Required
10.192.51.55	10.192.51.55	SIP	Request: ACK sip:88888@10.192.51.55
10.192.51.55	10.192.51.55	SIP/SDP	Request: INVITE sip:88888@10.192.51.55, wit
10.192.51.55	10.192.51.55	SIP	Status: 100 Trying
10.192.51.55	10.192.51.55	SIP/SDP	Request: INVITE sip:88888@10.192.51.55:5061
10.192.51.55	10.192.51.55	SIP	Status: 100 Trying

Figure 51: Capture SIP de l'étape 5 à 9 de notre diagramme en flèche

A ce moment, Asterisk va exécuter le script perl **agi-rad-auth.agi** qui est appelé dans notre fichier ***extensions.conf*** d'Asterisk. Voici figure ci-dessous de notre fichier extensions.conf d'Asterisk, pour plus de détail voir annexe A :

```
exten => 77777,1,Set(SIP_Authorization=${SIP_HEADER(Proxy-Authorization)})
exten => 77777,n,AGI(agi-rad-auth.agi|Mode=SIP)
exten => 77777,n,Goto(default,${EXTENSION},1)
exten => 77777,n,Hangup()

exten => 88888,1,Set(SIP_Authorization=${SIP_HEADER(Proxy-Authorization)})
exten => 88888,n,AGI(agi-rad-auth.agi|Mode=SIP)
exten => 88888,n,Goto(default,${EXTENSION},1)
exten => 88888,n,Hangup()
```

Figure 52: Configuration du fichier extensions.conf pour la solution radiusclient-1.4

Asterisk exécute premièrement la fonction **Set ()** pour récupérer les valeurs Digest de l'utilisateur 77777 afin de l'appliquer par la suite au script perl d'authentification. Voici le compte rendu de cette application dans la console CLI d'Asterisk en bas :

```
Executing[88888@sip_auth:1]
 Set("SIP/77777-09fb21d8","SIP_Authorization=Digest
username="77777",realm="asterisk",nonce="61ccb859",
response="ac6840a0aada2e0a84580aa71b9ece63",uri="sip:88888@10.192.51.55",
algorithm=MD5") in new stack
```

En deuxième lieu, on exécute le script perl **agi-rad-auth.agi** avec les paramètres récupérés auparavant par la fonction ***Set().*** Voici ci-dessous, un aperçu de son exécution sur notre console CLI :

```
-- Executing [88888@sip_auth:2] AGI("SIP/77777-09fb21d8", "agi-rad-auth.agi|Mode=SIP")
in new stack

   -- Launched AGI Script /var/lib/asterisk/agi-bin/agi-rad-auth.agi
   --  agi-rad-auth.agi|Mode=SIP: AGI Environment Dump:
   --  agi-rad-auth.agi|Mode=SIP:  -- accountcode =
   --  agi-rad-auth.agi|Mode=SIP:  -- callerid = 77777
   --  agi-rad-auth.agi|Mode=SIP:  -- calleridname = 77777
   --  agi-rad-auth.agi|Mode=SIP:  -- callingani2 = 0
   --  agi-rad-auth.agi|Mode=SIP:  -- callingpres = 0
   --  agi-rad-auth.agi|Mode=SIP:  -- callingtns = 0
   --  agi-rad-auth.agi|Mode=SIP:  -- callington = 0
   --  agi-rad-auth.agi|Mode=SIP:  -- channel = SIP/77777-09fb21d8
   --  agi-rad-auth.agi|Mode=SIP:  -- context = sip_auth
```

```
   -- agi-rad-auth.agi|Mode=SIP:  -- dnid = 88888
   -- agi-rad-auth.agi|Mode=SIP:  -- enhanced = 0.0
   -- agi-rad-auth.agi|Mode=SIP:  -- extension = 88888
   -- agi-rad-auth.agi|Mode=SIP:  -- language = en
   -- agi-rad-auth.agi|Mode=SIP:  -- priority = 2
   -- agi-rad-auth.agi|Mode=SIP:  -- rdnis = unknown
   -- agi-rad-auth.agi|Mode=SIP:  -- request = agi-rad-auth.agi
   -- agi-rad-auth.agi|Mode=SIP:  -- type = SIP
   -- agi-rad-auth.agi|Mode=SIP:  -- uniqueid = 1229308554.20
   -- agi-rad-auth.agi|Mode=SIP: RADIUS server response type = 2
   -- AGI Script Executing Application: (UserEvent)
Options: (_SIP_Auth|User-Name: 77777|DNID: |Channel: SIP/77777-09fb21d8)

   -- AGI Script agi-rad-auth.agi completed, returning 0
```

D'après notre capture de l'exécution de notre script ci-dessus, nous constatons que le script s'est exécuté correctement en ne générant aucune erreur. Ceci est montré par le message à la fin de la capture **« returning 0 »**.

L'exécution du script signifie que notre client radius a communiqué avec FreeRadius en lui envoyant un paquet radius de type **Access Request** pour authentifier l'utilisateur SIP ***77777*** et donner l'autorisation de faire cet appel à ***88888***.
Donc le client radius récupère le nom de l'utilisateur et le mot de passe de l'utilisateur SIP ***77777***, il crypte ces informations à l'aide du secret partagé et il envoie ce paquet ***Access-Request*** pour notre serveur FreeRadius ***(authentification)***. Ensuite si le client radius reçoit un message de type **Access Accept** alors l'appel peut être effectué.
Si le mot de passe est incorrect ainsi que le numéro de téléphone ou que l'utilisateur n'est pas enregistré dans FreeRadius, ce dernier transmettra un message de type **Access Reject** à notre client radius ou NAS. Les étapes **10** et **11** de notre diagramme correspondent à ces échanges des deux paquets radius d'authentification (***Access Request et Accept***). Voici une capture de ces paquets sur le sniffer :

Figure 53: Paquet radius d'authentification sous FreeRadius

Le client envoi à FreeRadius un paquet radius de type **Access Request** contenant toutes les informations de l'utilisateur SIP **77777** pour l'authentifier. Cette identification est faite selon le **protocole Digest**, donc notre client va envoyer les attributs digest encapsulé dans le paquet radius Request. Il les insère dans les **attributs AVP du protocole radius**. Voir dans figure 35 les paramètres envoyés par le client radius dans le sniffer :

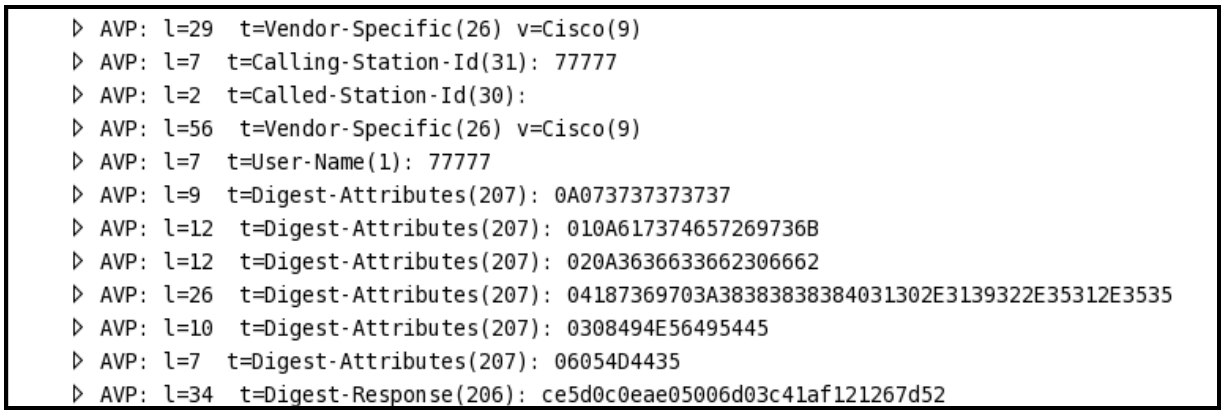


Figure 54: Attributs AVP de l'utilisateur SIP 77777 encapsulés dans Access Request

Voici une autre capture de ce même paquet **radius Access Request** dans la **console debug de FreeRadius** pour voir tous les attributs digest envoyés par le client radius :

rad_recv: Access-Request packet from host 10.192.51.55:33763, id=36, length=262
NAS-IP-Address = 10.192.51.55
Cisco-NAS-Port = "SIP/77777-0917daf8"
Cisco-AVPair = "call-id=1229316984.2"
Calling-Station-Id = "77777"
Called-Station-Id = ""
h323-conf-id = "h323-conf-id=0AFA0ACE F5E3B7C2 D676D429 D48F8C14"
User-Name = "77777"
Digest-Attributes = 0x0a073737373737
Digest-Attributes = 0x010a617374657269736b
Digest-Attributes = 0x020a3539393063623532
Digest-Attributes = 0x04187369703a38383838384031302e3139322e35312e3535
Digest-Attributes = 0x0308494e56495445
Digest-Attributes = 0x06054d4435
Digest-Response = "1744b1ca1914440898aafd6f65f356a2"
Processing the authorize section of radiusd.conf

Comme nous utilisons une authentification basé sur le protocole digest et que nous savons que nos utilisateurs sont stockés dans une BD sous FreeRadius. Ce dernier offre la possibilité de stocker des utilisateurs selon un groupe utilisant un certain type d'authentification. Le principe est simple car il faut juste définir un groupe dans la table **radgroupcheck** de FreeRadius et lui attribuer un type d'authentification. Bien sûr dans notre projet, le type d'authentification est digest. Ensuite dans la table **usergroup**, nous introduisons l'appartenance d'un nouveau utilisateur à ce groupe crée dans **radgroupcheck**.
Voici la procédure sur la manière de créer un groupe d'authentification digest avec les tables MYSQL de FreeRadius.

```
mysql> insert into radgroupcheck (GroupName,Attribute,op,value) values ('digestG
roup', 'Auth-Type','==','Digest');
Query OK, 1 row affected (0.00 sec)

mysql> insert into usergroup(UserName,GroupName) values('77777', 'digestGroup');

Query OK, 1 row affected (0.00 sec)

mysql> insert into usergroup(UserName,GroupName) values('88888', 'digestGroup');

Query OK, 1 row affected (0.00 sec)
```

L'introduction d'un nouvel utilisateur se fait comme d'habitude dans la table **radcheck** de FreeRadius et l'ajout du client (NAS) dans la table ***nas***. Voire capture ci-dessous :

```
mysql> insert into radcheck (UserName,Attribute,op,value) values ('77777', 'User
-Password','==','test');
Query OK, 1 row affected (0.00 sec)

mysql> insert into radcheck (UserName,Attribute,op,value) values ('88888', 'User
-Password','==','test2');
Query OK, 1 row affected (0.00 sec)

mysql> insert into nas (nasname,shortname,secret) values ('10.192.51.55', 'NasAs
terisk','testing123');
Query OK, 1 row affected (0.00 sec)
```

Revenant au principe de l'authentification, par conséquent la prochaine étape est celle de l'envoi du paquet ***Access Request*** à FreeRadius par le client pour l'identification. Les utilisateurs sont enregistrés dans la BD de données de FreeRadius alors ce dernier va exécuter une série de **requêtes SQL**. Voici un aperçu des ces requêtes dans la console debug de FreeRadius :

rlm_sql (sql): sql_set_user escaped user --> '77777'
radius_xlat: **'SELECT id, UserName, Attribute, Value, op FROM radcheck WHERE Username = '77777' ORDER BY id'**
rlm_sql (sql): Reserving sql socket id: 4
radius_xlat: **'SELECT radgroupcheck.id,radgroupcheck.GroupName,radgroupcheck.Attribute, radgroupcheck.Value,radgroupcheck.op FROM radgroupcheck,usergroup WHERE usergroup.Username = '77777' AND usergroup.GroupName = radgroupcheck.GroupName ORDER BY radgroupcheck.id'**
radius_xlat: **'SELECT id, UserName, Attribute, Value, op FROM radreply WHERE Username = '77777' ORDER BY id'**
radius_xlat: **'SELECT radgroupreply.id,radgroupreply.GroupName,radgroupreply.Attribute,radgroupreply.Value, radgroupreply.op FROM radgroupreply,usergroup WHERE usergroup.Username = '77777' AND usergroup.GroupName = radgroupreply.GroupName ORDER BY radgroupreply.id'**

FreeRadius va commencer l'authentification de l'utilisateur **77777** après avoir eu la récupération de toutes les données via SQL avec succès. Pour ce faire, il va convertir les paramètres digest envoyés par le client radius dans le paquet Access Request à l'aide du cryptage MD5 et les comparer avec les infos tirées par FreeRadius de SQL. Si toutes les données sont identiques c'est à dire le nom d'utilisateur et le mot de passe, FreeRadius envoi un message **Access Accept** au client Rradius. Pour avoir une idée sur le principe du décryptage des données digest par FreeRadius voir la capture ci-dessous :

```
Processing the authenticate section of radiusd.conf
modcall: entering group authenticate for request 16
  rlm_digest: Converting Digest-Attributes to something sane...
        Digest-User-Name = "77777"
        Digest-Realm = "asterisk"
        Digest-Nonce = "5990cb52"
        Digest-URI = "sip:88888@10.192.51.55"
        Digest-Method = "INVITE"
        Digest-Algorithm = "MD5"
A1 = 77777:asterisk:test
A2 = INVITE:sip:88888@10.192.51.55
KD = c714f2ba7e6f1eca7147cd98fd05e1ff:5990cb52:adf8eccf711ec8c5df21d9faa6988cea
  modcall[post-auth]: module "sql" returns ok for request 16
modcall: leaving group post-auth (returns ok) for request 16
Sending Access-Accept of id 36 to 10.192.51.55 port 33763
```

Quand le paquet ***Access Accept*** arrive vers le client radius, l'appel peut être effectué donc le canal de communication va être activé pour l'utilisateur SIP 77777. Les étapes **12** à **14** vont être réalisées de notre digramme en flèche ci-dessus. La communication commence par le transfert des paquets RTP d'audio.
Voici la capture ci-dessous dans la console CLI de l'exécution des instructions restantes de notre fichier extensions.con d'Asterik sur l'acheminement de l'appel entre le 77777 et 88888 :

```
   -- Executing [88888@sip_auth:3] Goto("SIP/77777-09fb21d8", "default||1") in new stack
   -- Goto (default,88888,1)
   -- Executing [88888@default:1] Dial("SIP/77777-09fb21d8", "SIP/88888") in new stack
Audio is at 10.192.51.55 port 17000
Adding codec 0x400 (ilbc) to SDP
Adding codec 0x2 (gsm) to SDP
 Reliably Transmitting (no NAT) to 10.192.51.55:5061:
INVITE sip:88888@10.192.51.55:5061
SIP/2.0 v:  SIP/2.0/UDP  10.192.51.55:5060;branch=z9hG4bK3413f831;rport f:  "77777"
<sip:77777@10.192.51.55>;tag=as3aa0860e t: <sip:88888@10.192.51.55:5061>
```

Suite à ce test réalisé, nous avons pu voir les détails de l'authentification entre Asterisk et FreeRadius à l'aide du ***client radiusclient-1.4*** écrit en perl. Cette démonstration nous montre que ***le client radius fonctionne correctement***.

8.2 Test final du mécanisme d'accounting dans notre application

L'accounting dans notre projet consiste d'avoir une trace de chaque appel effectué après une authentification via FreeRadius. Cette empreinte de chaque appel réalisé doit être gérée par notre serveur FreeRadius. Vu les tests réalisés auparavant, l'accounting sera géré par un autre client radius que celui utilisé dans l'authentification. Ce client est notre client radiusclient-ng qui permet d'envoyer les informations de l'accounting sous forme de champs CDR. Ces champs sont crées dans Asterisk donc le rôle de notre client est d'envoyer ces paramètres à notre serveur FreeRadius à la fin de chaque appel.

Avec le test précédent de l'authentification, nous sommes arrivés à l'étape ***14*** de notre diagramme en flèche ci-dessus. Ici, nous traiterons le reste des étapes **15 à 18**.

Nous avons parlé de la variable de session qui est le point déterminant pour l'accounting. Cette session utilise le même principe que la session du protocole http d'internet. Un utilisateur SIP essaie chaque fois de se connecter pour effectuer un appel, et Asterisk créera un id de session unique pour chaque appel. Sur la base de cet intervalle du temps de la session, FreeRadius et Asterisk calculera la durée de chaque appel.

Le principe général est que le client radius envoi un paquet radius **Accounting Request de type Start** par l'intermédiaire d'Asterisk. Ce paquet doit être envoyé juste après la réception du paquet radius Access Accept avant de commencer l'appel. Ensuite, FreeRadius répond par un paquet radius **Accounting response** pour confirmer la réception de la **session start**. Or lors du test de ce client, nous avons remarqué qu'il n'envoi pas ce genre de paquet à FreeRadius. Donc peut-être il utilise un autre moyen pour envoyer la session start.
Dès que l'utilisateur se déconnecte en raccrochant le téléphone, à cet instant précis, la session de cet utilisateur se termine. Et notre client radius envoie un paquet radius de type **Accounting Request stop**. Par la suite notre serveur FreeRadius répond par un accusé de réception de la fin de la session. Alors il envoi un paquet radius de type **Accounting Response**.

Voici ci-dessous une capture de ces paquets de la fin de session sur notre serveur FreeRadius à l'aide du sniffer :

Figure 55: Paquet radius de l'accounting sous FreeRadius

Cette autre capture du paquet ***Accounting Request***, nous montre qu'il s'agit bien d'un paquet de type **Accounting Request** car il a la valeur du code **4**. Car selon cette valeur du code dans le paquet Accounting qu'on peut vérifier s'il s'agit d'un ***Request*** ou ***Reponse***. Et si la valeur de ce code est de **5**, donc il s'agit d'un **Response**.

Et dans les attributs d'***AVP*** intégrés dans le paquet ***Accounting Request***, on vérifie s'il s'agit d'un Accounting de type Stop, Start ou Interim et etc….
D'après la capture ci-dessous de notre paquet ***Accounting***, nous pouvons voir qu'il s'agit d'un Accounting **Request** à cause de la valeur **4** et il est de type **Stop** à cause de la valeur **2** dans le champ **Value** de l'attribut **Acct-Status-Type** des **AVP**s (**Start** la valeur **1**,**interim-Update** la valeur **3**).

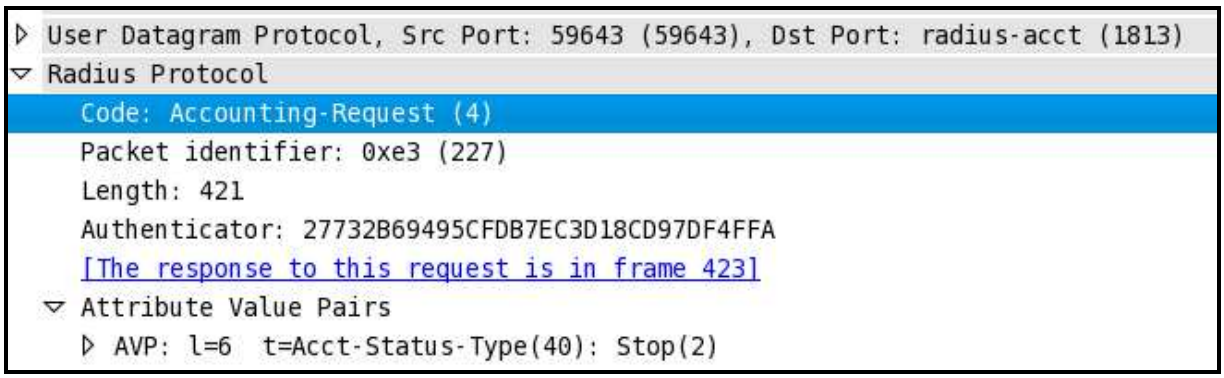


Figure 56: Paquet Accounting Request à l'état Stop

Dans ces attributs ***AVP*** du paquet ***Accounting Request***, on trouve des informations sur la variable session. Donc elle est identifiée à l'aide de l'attribut **Acct-Session-Id** du paquet radius.

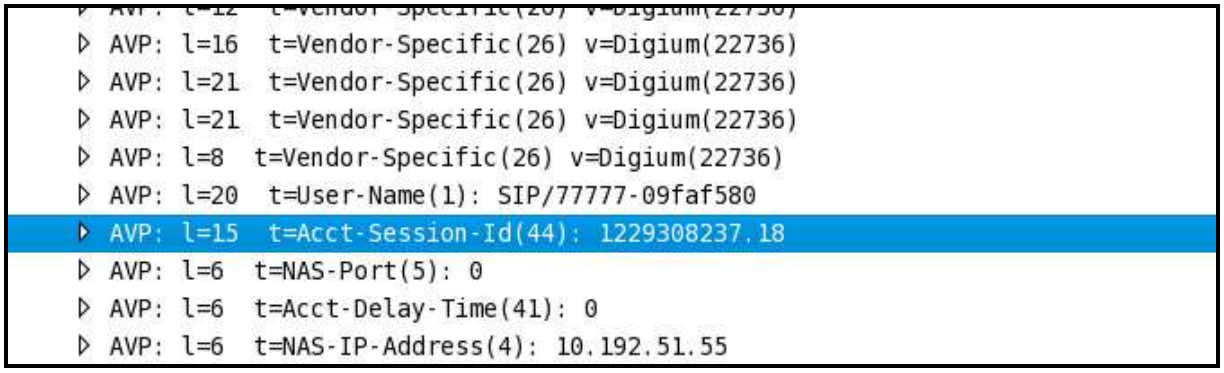


Figure 57: la variable session dans les attributs AVP du paquet Accounting Request

Dès que la session se termine, Asterisk enregistre ces champs ***CDR*** concernant l'accounting sous forme de fichier texte. Et dans un point de nos améliorations importées, nous avons mis en place un mécanisme d'enregistrement des CDR dans une table CDR MYSQL. En même temps, notre ***clientradius-ng*** envoi à FreeRadius ces données CDR de l'accounting. Cet envoi se fait lors de la réception de notre serveur FreeRadius le paquet **Accounting Request Stop**. Voici une démo de la réception de ce paquet sur la console debug de FreeRadius comme indiqué en bas :

```
rad_recv: Accounting-Request packet from host 10.192.51.55:36337, id=235, length=419

Acct-Status-Type = Stop
Asterisk-Acc-Code = ""
Asterisk-Src = "77777"
```

```
Asterisk-Dst = "88888"
Asterisk-Dst-Ctx = "default"
Asterisk-Clid = ""77777" <77777>"
Asterisk-Chan = "SIP/77777-0917daf8"
Asterisk-Dst-Chan = "SIP/88888-091906b8"
Asterisk-Last-App = "Dial"
Asterisk-Last-Data = "SIP/88888"
Asterisk-Start-Time = "2008-12-15 04:56:24 +0000"
Asterisk-Answer-Time = "2008-12-15 04:56:28 +0000"
Asterisk-End-Time = "2008-12-15 04:56:30 +0000"
Asterisk-Duration = 6
Asterisk-Bill-Sec = 2
Asterisk-Disposition = "ANSWERED"
Asterisk-AMA-Flags = "DOCUMENTATION"
Asterisk-Unique-ID = "1229316984.2"
Asterisk-User-Field = ""
User-Name = "SIP/77777-0917daf8"
Acct-Session-Id = "1229316984.2"
NAS-Port = 0
Acct-Delay-Time = 0
NAS-IP-Address = 10.192.51.55 modcall: entering group accounting for request 17
```

La capture ci-dessus, nous explique bien le principe de la session dans l'accounting. La variable CDR **Asterisk-Start-Time = "2008-12-15 04 :56 :24 +0000"** est le temps du start de la session. L'autre variable CDR **Asterisk-Answer-Time = "2008-12-15 04 :56 :28 +0000"** est le temps auquel l'appelé a décroché le téléphone. Ceci veut dire que la session est créée avant que l'appel soit réalisé, donc lors de la composition du numéro.
Pour confirmer cette logique, nous avons trouvé deux autres variables importantes dans la capture ci-dessus. Ces variables sont **Asterisk-Duration = 6** et **Asterisk-Bill-Sec = 2**. La première donne le temps de la durée de la session et le deuxième donne le temps réel de la communication. La facturation de l'appel sera faite à la base du champ **Asterisk-Bill-Sec** du CDR.

Maintenant que FreeRadius a reçu ce message d'accounting, il tentera d'exécuter des commandes SQL pour enregistrer ces informations dans la table ***radacct*** de MYSQL. Voir ci-dessus un exemple des requêtes envoyés par FreeRadius au serveur MySQL dans sa console debug :

```
rlm_sql (sql): sql_set_user escaped user --> 'SIP/77777-0917daf8'

radius_xlat:  'UPDATE radacct SET AcctStopTime = '2008-12-15 23:57:28',
AcctSessionTime   = '',   AcctInputOctets   = '0' << 32 |                    '0',
AcctOutputOctets  = '0' << 32 |   '0',  AcctTerminateCause = '',          AcctStopDelay     = '0',
ConnectInfo_stop  = ''        WHERE AcctSessionId  = '1229316984.2'  AND UserName  =
```

'SIP/77777-0917daf8' AND NASIPAddress = '10.192.51.55''

rlm_sql (sql): Reserving sql socket id: 2

radius_xlat: 'stop packet with zero session length. [user 'SIP/77777-0917daf8', nas '10.192.51.55']'

rlm_sql (sql) in sql_accounting: stop packet with zero session length. [user 'SIP/77777-0917daf8', nas '10.192.51.55']

rlm_sql (sql): Released sql socket id: 2

radius_xlat: 'INSERT INTO radacct (AcctSessionId, AcctUniqueId, UserName, Realm, NASIPAddress, NASPortId, NASPortType, AcctStartTime, AcctStopTime, AcctSessionTime, AcctAuthentic, ConnectInfo_start, ConnectInfo_stop, AcctInputOctets, AcctOutputOctets, CalledStationId, CallingStationId, AcctTerminateCause, ServiceType, FramedProtocol, FramedIPAddress, AcctStartDelay, AcctStopDelay) VALUES ('1229316984.2', '9251c9705fae753f', 'SIP/77777-0917daf8', '', '10.192.51.55', '0', '', '2008-12-15 04:56:28 =2B0000','2008-12-15 04:56:30 =2B0000', '2', '', '2008-12-15 04:56:24 =2B0000', '2008-12-15 04:56:30 =2B0000', '0' << 32 | '0', '0' << 32 | '0', '88888', '77777', '', '', '', '','0', '0')'

rlm_sql (sql): Released sql socket id: 2
modcall[accounting]: module "sql" returns noop for request 17
modcall: leaving group accounting (returns ok) for request 17

Sending Accounting-Response of id 235 to 10.192.51.55 port 36337

Pour vérifier encore une autre fois cette insertion d'accounting dans la base de données radius de FreeRadius, nous avons fait un test de la commande SQL select sur la table **radacct**. Voir les figure ci-dessous de la table ***radacct*** :

AcctSessionId	AcctUniqueId	UserName	Realm	NASIPAddress	NASP
1229255363.12	fbdca913991d35b7	SIP/77777-09faf580		10.192.51.55	0
1229255417.14	394d8086c5fae8f9	SIP/88888-09fc3850		10.192.51.55	0
1229255567.16	136f585a459e9b23	SIP/88888-09faf580		10.192.51.55	0
1229308237.18	e5ee10f481482849	SIP/77777-09faf580		10.192.51.55	0
1229308554.20	99a6b964687c57a2	SIP/77777-09fb21d8		10.192.51.55	0
1229309435.0	6bd21505630ca51f	SIP/88888-097a6558		10.192.51.55	0
1229309869.0	f628ef67520f1a61	SIP/77777-08d5b190		10.192.51.55	0
1229310342.0	e4fdfb677068a1db	SIP/88888-089ee5d0		10.192.51.55	0

AcctStartTime	AcctStopTime	AcctSessionTime	AcctAuthentic	ConnectIr
2008-12-14 11:49:24	2008-12-14 11:49:45	21		2008-12-1
2008-12-14 11:50:18	2008-12-14 11:50:52	34		2008-12-1
2008-12-14 11:52:50	2008-12-14 11:52:58	8		2008-12-1
2008-12-15 02:30:53	2008-12-15 02:31:06	13		2008-12-1
2008-12-15 02:36:01	2008-12-15 02:36:03	2		2008-12-1
2008-12-15 02:50:38	2008-12-15 02:50:40	2		2008-12-1
2008-12-15 02:57:52	2008-12-15 02:57:54	2		2008-12-1
2008-12-15 03:05:44	2008-12-15 03:06:02	18		2008-12-1

2008-12-14 11:49:23 =2B0000	2008-12-14 11:49:45 =2B0000	0	0
2008-12-14 11:50:17 =2B0000	2008-12-14 11:50:52 =2B0000	0	0
2008-12-14 11:52:47 =2B0000	2008-12-14 11:52:58 =2B0000	0	0
2008-12-15 02:30:37 =2B0000	2008-12-15 02:31:06 =2B0000	0	0
2008-12-15 02:35:54 =2B0000	2008-12-15 02:36:03 =2B0000	0	0
2008-12-15 02:50:35 =2B0000	2008-12-15 02:50:40 =2B0000	0	0
2008-12-15 02:57:49 =2B0000	2008-12-15 02:57:54 =2B0000	0	0
2008-12-15 03:05:42 =2B0000	2008-12-15 03:06:02 =2B0000	0	0
2008-12-15 03:11:52 =2B0000	2008-12-15 03:12:05 =2B0000	0	0

Ensuite, nous avons vérifié l'enregistrement de toutes les valeurs de l'accounting dans la table CDR de la base de données asterisk. Le test s'est avéré positif car il y a des requêtes sql qui ont été envoyés par notre serveur Asterisk vers MYSQL à la fin de l'appel. Voir aperçu ci-dessous :

[Dec 14 21:36:03] DEBUG[13505]: cdr_addon_mysql.c:210 mysql_log: cdr_mysql: inserting a CDR record.

[Dec 14 21:36:03] DEBUG[13505]: cdr_addon_mysql.c:226 mysql_log: cdr_mysql: SQL command as follows:INSERT INTO cdr (calldate,clid,src,dst,dcontext,channel,dstchannel,lastapp,lastdata,duration,billsec,disposition, amaflags, accountcode, userfield) VALUES ('2008-12-14 21:35:54','\"77777\" <77777>','77777','88888','default', 'SIP/77777-09fb21d8','SIP/88888-09fc9860','Dial','SIP/88888',9,2,'ANSWERED',3,'','')

Et dans la table **sipfriends** de la base de données asterisk, nous enregistrons la trace des utilisateurs SIP qui ont effectuées l'appel comme ceci :

77777	friend	77777	test	10.192.51.55	NULL	5061	dynamic	122931332
88888	friend	88888	test2	10.192.51.55	NULL	5061	dynamic	122931332

Figure 58: Update de l'adresse IP des utilisateurs SIP de la table sipfriends

Il y a eu un changement dans la table au dessus, c'est qu'Asterisk a introduit l'adresse IP réelle et le port de l'utilisateur SIP. Alors qu'au début, cette valeur est initialisée à 0.0.0.0. Donc il a exécuté entre autre la commande update à l'aide de Realtime.

Aussi nous avons une autre trace locale de tous les utilisateurs SIP enregistrés dans la base de données **Astdb** d'Asterisk sous forme de texte. Voici une capture de la BD Astdb d'Asterisk :

```
b1
9/SIP/Registry/8888810.192.51.55:5061:1800:88888:sip:88888@10.192.51.55:5061
=;9/SIP/Registry/8888810.192.51.55:5061:1800:88888:sip:88888@10.192.51.55:50
61=;/dundi/secretexpiry12293374222/dundi/secret036KUrlv6R0vka4DCJPwAQ==;5
OWYR+yDzxXL1mNS2DYPlw==4vg
```

Figure 59: Trace locale des utilisateurs SIP dans la BD AstDB

9 CONCLUSION

Le but de ce projet est de pouvoir mettre en place un mécanisme d'authentification des utilisateurs et de facturation des appels à l'aide d'un serveur Asterisk via Radius. Ce qui implique l'intégration de ce serveur Asterisk dans l'infrastructure VOIP de la société ***Switzernet*** après des tests effectués sur son fonctionnement.

Au terme de ce projet de diplôme, nous estimons que le travail accompli respecte les points visés dans notre cahier de charge. Nous avons pu construire une infrastructure VOIP qui peut identifier les utilisateurs à l'aide d'un nom d'utilisateur et un mot de passe, et enregistrer les données de facturation des appels dans Radius. Le but principal du projet est parfaitement accompli à l'aide des multiples tests de plusieurs solutions réalisées durant le projet. Donc cette plateforme proposée dans notre projet peut être utilisée chez Switzernet pour gérer la facturation de leurs appels.

Dans notre rapport, nous avons mentionnés en détail les différentes difficultés rencontrées durant notre travail. Nous avons rencontré des problèmes dont des bugs et des solutions non fonctionnelles à cause du manque de documentation et des modules qui ne sont plus disponibles sur le net pour faire fonctionner l'authentification. Tous ces facteurs et surtout l'absence de la documentation concernant l'utilisation d'Asterisk avec Radius pour l'authentification et l'accounting, ont joué un grand rôle sur le rétrécissement de notre temps disponible dans la réalisation de notre projet de fin d'étude.
Malgré ces facteurs, nous avons pu finaliser une solution d'authentification et de facturation à l'aide de deux clients radius différents combinés ensemble. D'après nos tests réalisés, il s'est avéré qu'une marche mieux que l'autre pour authentifier les utilisateurs via radius et réciproquement pour gérer la facturation dans le serveur Radius.

L'amélioration possible est d'essayer de finaliser la réplication de la BD MYSQL sur un autre serveur pour externaliser les données le plus possible. Cette option est d'ordre facultatif dans notre projet. L'autre amélioration possible est d'étudier le principe de routage, Voice mail et IVR dans Asterisk afin d'implémenter ces options dans notre application. Ceci dont le but de découvrir les multitudes fonctions et intéressantes qu'Asterisk offre aux utilisateurs dans le domaine de VOIP.

Pour conclure, nous pouvons dire que ce projet est très enrichissant du point de vue découvertes de nouvelles technologies VOIP. Il nous a permis aussi de nous confronter à plusieurs types de problèmes et de les résoudre avec patience et courage. Nous avons constaté que cette technologie VOIP est entrain de devenir de plus en plus innovante et mature. Elle a une belle avenir devant elle et elle est une des technologies les plus prometteuses d'ici les années à venir. La seule contrainte est le manque de documentation et des mises à jour de certaines solutions VOIP surtout dans les environnements de notre projet, ceci s'explique par l'évolution rapide de cette technologie et la commercialisation de certaines solutions.

10 BIBLIOGRAPHIE

- [1] : http://www.sestream.com/docCom/Asterisk.pdf
- [2]: http://www-igm.univmlv.fr/~dr/XPOSE2007/jgauth02_RADIUS/presentation.pdf
- [3] : http://www.commentcamarche.net/authentification/radius.php3
- [4]: Livre Authentification réseau avec Radius : 802.1x, EAP, FreeRadius de Serge Border
- [5] : http://www.normes-internet.com/normes.php?rfc=rfc2865&lang=fr
- [6] : http://books.google.ch/books?id=LEfV303mJX0C&pg=PA164&lpg=PA164&dq=exemple+du+fichier+Master.csv&source=web&ots=eHTnzzTldD&sig=Z1UE4W-dstyv2vdhrkNU-N6bL_A&hl=fr&sa=X&oi=book_result&resnum=1&ct=result
- [7]: http://bugs.digium.com/file_download.php?file_id=9744&type=bug
- [8] : http://www.voip-info.org/wiki/view/Asterisk+billing
- [9]:http://wapiti.telecomlille1.eu/commun/ens/peda/options/ST/RIO/pub/exposes/exposesrio2006/Cappelaere-Lukic/presentation.html
- [10]: http://www.asteriskguru.com/tutorials/xlite_softphone.html
- [11]:http://wwwrocq.inria.fr/who/Philippe.Sultan/Asterisk/asterisk_sip_external_authentication.html
- [12]: Livre en anglais VOIP et TOIP Asterisk de Sébastien Déon
- [13] : http://www.frontios.com/freeradius.html
- [14] :http://infodotnet.blogspot.com/2008/03/install-and-configure-freeradius-with.html
- [15] : http://www.voip-info.org/wiki/view/Asterisk+Documentation+1.4+radius.txt
- [16] : http://ing.ctit.utwente.nl/WU5/D5.1/Technology/radius/
- [17] : http://www.lanbilling.ru/asterisk_solution.html
- [18] : http://www.voip-info.org/tiki-index.php?page=PortaOne+Radius+auth

- [19] : http://www.voip-info.org/wiki-Asterisk+cdr+mysql
- [20] : http://www.voip-info.org/wiki/view/Asterisk+RealTime+Sip
- [21] : http://www.lephpfacile.com/manuel-mysql/replication.php

11 LISTES DES FIGURES

12 ANNEXES