Université d'Artois

IUT de Béthune

Département Réseaux et Télécommunications

SAÉ5.ROM03

Déployer et gérer les services ROM

par

Rémi BONNEL, Victor DAUVIN et Eliot HULEUX

Table des matières

Résumé du projet

Le projet vise à déployer et gérer des services de téléphonie et visioconférence basés sur Asterisk, un serveur de communication. Il est organisé en dix étapes clés, commençant par l'installation d'Asterisk sur une machine virtuelle avec la distribution Alpine. Ensuite, deux téléphones IP sont configurés pour tester les communications vocales. L'administration d'Asterisk est rendue accessible via une interface web développée sur un serveur Apache ou NGINX. La communication voix est réalisée grâce à la librairie sip.js et des tests sont effectués avec Wireshark. Les étapes suivantes intègrent la vidéo, une messagerie instantanée, et un portail web de visioconférence. Un audit de sécurité est ensuite réalisé, suivi de la mise en place de solutions pour sécuriser les échanges avec TLS.

The project aims to deploy and manage telephony and video conferencing services based on Asterisk, a communication server. It is organized in ten key steps, starting with the installation of Asterisk on a virtual machine with the Alpine distribution. Then, two IP phones are set up to test voice communications. Asterisk administration is accessible via a web interface developed on an Apache or NGINX server. Voice communication is done with the sip.js library and tests are performed with Wireshark. The following steps include video, instant messaging, and a web-based videoconferencing portal. A security audit is then carried out, followed by the implementation of solutions to secure exchanges with TLS.

1 Déployer un serveur Asterisk sur une machine virtuelle avec la distribution Alpine

Pour installer Asterisk, nous devons télécharger l'archive depuis le dépôt officiel :

Alpine Linux 1

```
wget https ://downloads.asterisk.org/pub/telephony/asterisk/asterisk-20.11.0.tar.gz
tar -xvzf asterisk-20.11.0.tar.gz
```

Puis éxécuter le script d'installation dans le dossier décompessé, et éxécuter le menu d'installation, permettant de définir l'ensemble des modules et codecs à installer avec Asterisk :

Alpine Linux 2

```
./configure
make menuselect
```

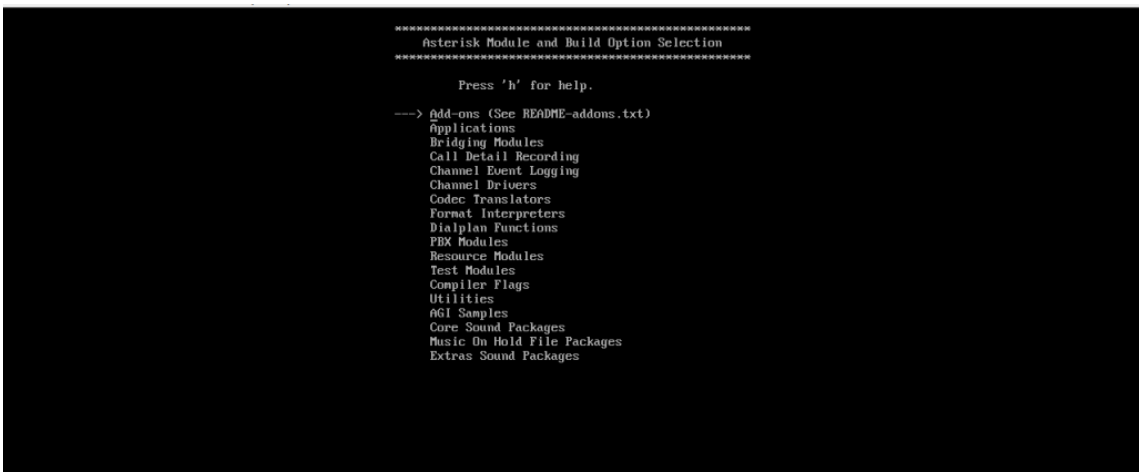


Figure 1 – *make menuselect*

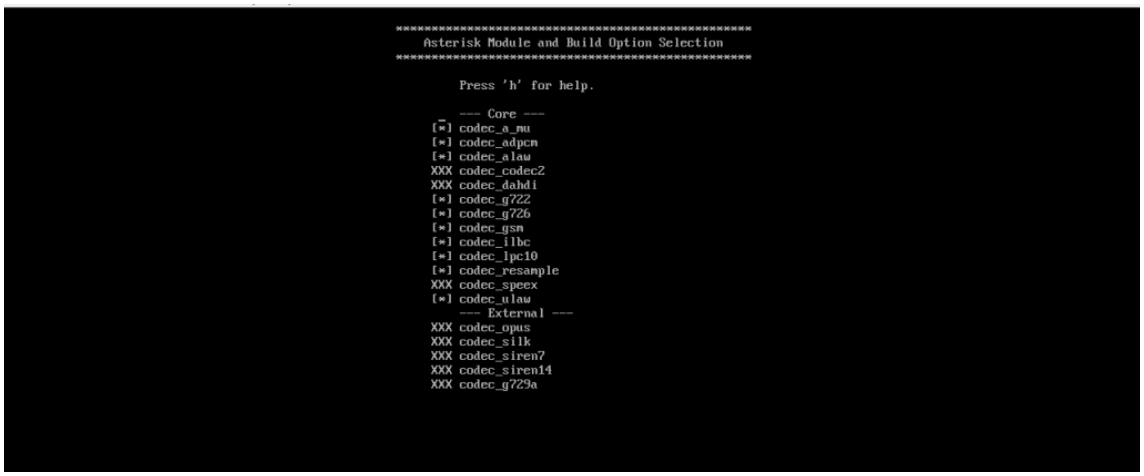


Figure 2 – *make menuselect pour les codecs*

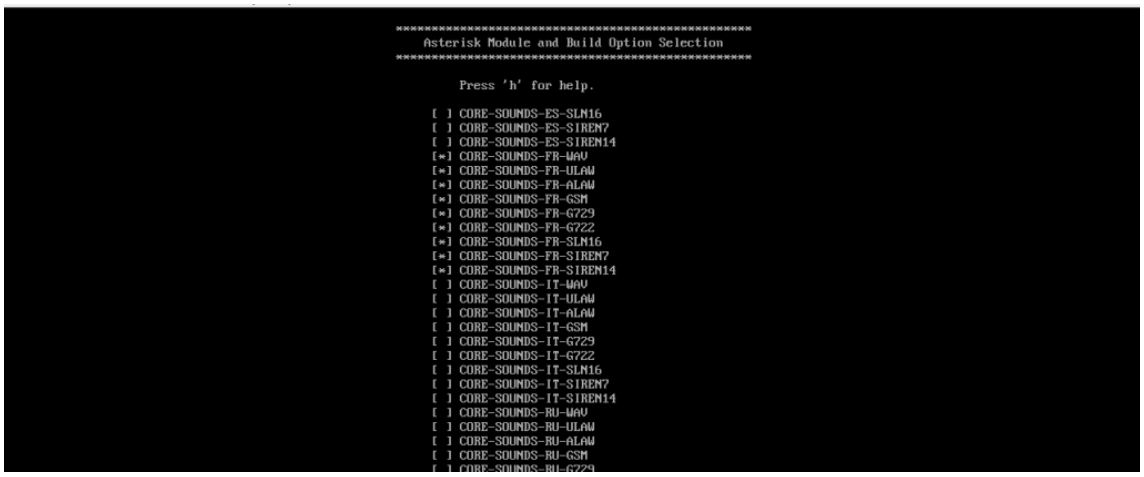


Figure 3 – *make menuselect pour la langue*

Il ne reste plus qu'à lancer l'installation :

Alpine Linux 3

```
./configure
make install
```

Pour démarrer le service Asterisk et le démarrer au démarrage de la machine, nous utiliserons OpenRC.

Alpine Linux 4

Pour démarrer le service

rc-service asterisk start

Pour ajouter le service au démarrage

rc-update add asterisk

Nous pouvons vérifier également son status :

Alpine Linux 5

rc-service asterisk status

```
rt-mv:~# rc-service asterisk start
 * Caching service dependencies ...                                    [ ok ]
 * Starting asterisk PBX (as asterisk) ...                             [ ok ]
rt-mv:~# rc-service asterisk status
 * status: started
```

Figure 4 – *rc-service asterisk status*

2 Connecter deux téléphones IP (FANVIL et CISCO) sur le serveur tester la communication en mode voix

2.1 Configuration de `sip.conf` et `extensions.conf` pour les comptes et numérotations

Maintenant, nous pouvons configurer les deux principaux fichiers d'Asterisk, `/etc/asterisk/sip.conf` et `/etc/asterisk/extensions.conf` :

Figure 5 – *sip.conf*

Figure 6 – *extensions.conf*

Et redémarrer le service Asterisk pour appliquer les modifications :

Alpine Linux 6

Pour redémarrer le service

```
rc-service asterisk start
```

2.2 Téléphone Fanvil & Cisco

Pour connecter le téléphone Fanvil, il suffit de se rendre sur son interface web (le login et mot de passe sont par défaut *admin/admin*). Sur l'interface web, dans la section *Ligne*, il faut remplir les champs suivants :

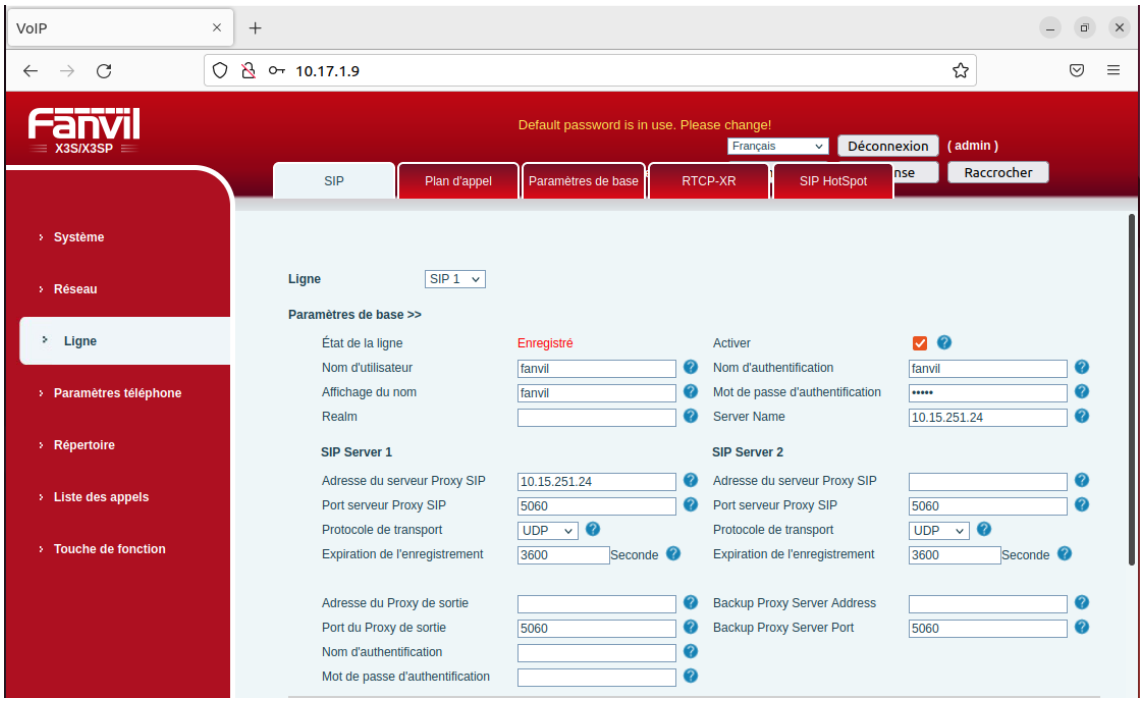


Figure 7 – *Configuration du téléphone Fanvil via son interface Web*

Les champs *Nom d'utilisateur, Nom d'authentification* doivent être rempli par le nom d'utilisateur associé, de même pour le mot de passe.

Les champs *Server Name, Adresse du serveur proxy SIP* correspondent à l'adresse du serveur Asterisk.

Le statut *Enregistré* prouve du bon fonctionnement.

Pour connecter le téléphone Cisco, il faut mettre en place un serveur TFTP :

Alpine Linux 7

```
apk add tftpd-hpa
```

Et éditez le fichier de configuration */etc/conf.d/in.tftpd* :

Alpine Linux 8

```
TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/srv/tftp"
TFTP_ADDRESS="0.0.0.0 :69"
TFTP_OPTIONS="–secure"
```

Et démarrer le service :

Alpine Linux 9

```
rc-service in.tftpd start
```

Sur le téléphone Cisco, il faut le réinitialiser et renseigner l'adresse IP du serveur TFTP :

Alpine Linux 10

```
1234567890#
```

Figure 8 – *Renseigner l'adresse du serveur TFTP*

Maintenant, il faut éditer le fichier de configuration à envoyer au téléphone. Le fichier peut se nommer de deux manières : *SEP<@MAC>.cnf.xml* ou *XMLDefault.cnf.xml*, et les grandes lignes à modifier :

Alpine Linux 11

<proxy>10.15.251.26</proxy>
<port>5060</port>
<authName>Cisco</authName>
<authPassword>admin</authPassword>

Sur Wireshark, nous visualisons bien l'enregistrement :

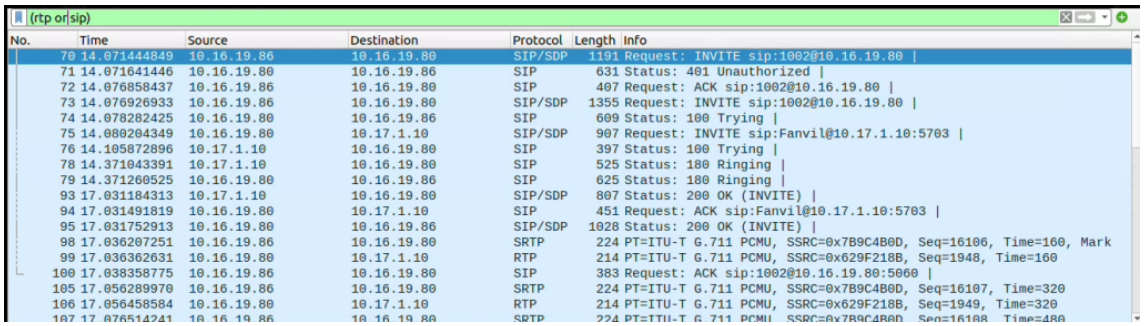


Figure 9 – *Requête REGISTER du téléphone Cisco*

2.3 Mise en place des boîtes vocales

Pour mettre en place des boîtes vocales, cela se passe dans les fichier */etc/asterisk/-sip.conf*, */etc/asterisk/voicemail.conf* & */etc/asterisk/extensions.conf* :

Pour */etc/asterisk/sip.conf* :

Alpine Linux 12

```
[cisco]
type=friend
host=dynamic
secret=admin
context=internal
mailbox=1001@default
[fanvil]
type=friend
host=dynamic
secret=admin
context=internal
mailbox=1002@default
```

Pour */etc/asterisk/extensions.conf* :

Alpine Linux 13

```
[internal]
exten => 1001,1,Dial(SIP/cisco,5)
exten => 1001,n,Voicemail(1001@default,u)
exten => 1001,n,Hangup()
exten => 1002,1,Dial(SIP/fanvil,5)
exten => 1002,n,Voicemail(1002@default,u)
exten => 1002,n,Hangup()
exten => 888,1,VoiceMailMain($CALLERID(nom)@default)
```

Pour */etc/asterisk/voicemail.conf* :

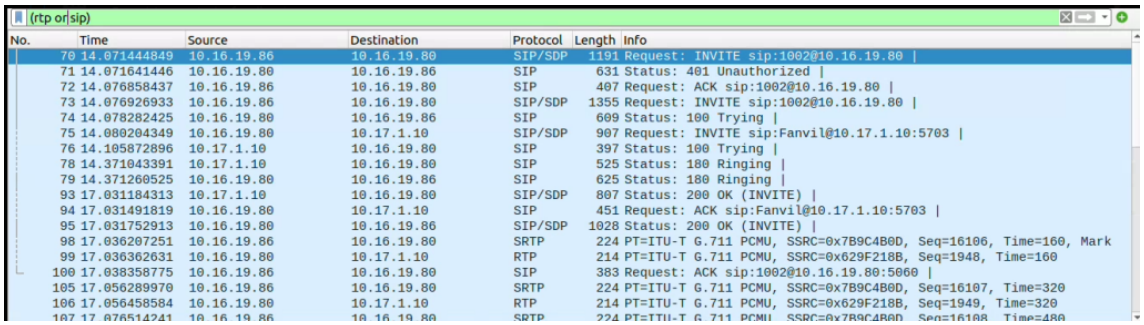
Alpine Linux 14

```
[default]
1001 => 0000,cisco
1002 => 0000,fanvil
```

Pour chaque extension, si l'utilisateur ne répond pas après 5 secondes, l'appel est redirigé vers la boîte vocale.

2.4 Échange entre les deux téléphones

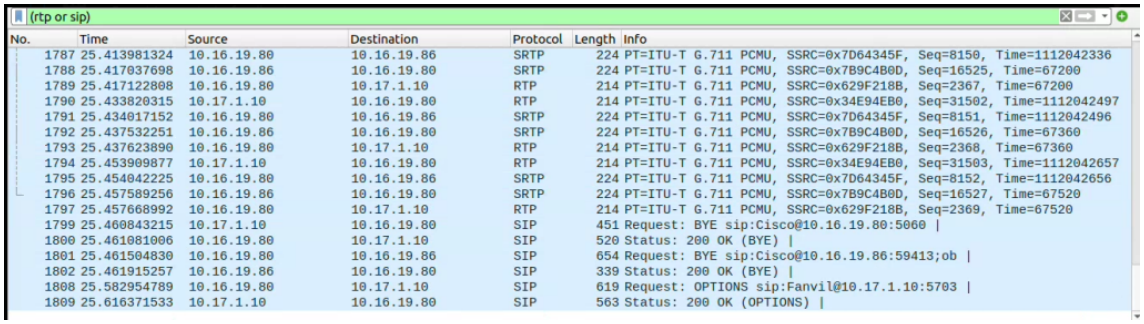
Lors d'un échange entre deux téléphones, sur Wireshark, nous avons bien les trames SIP *Invite*, *Ringing*, et *Bye* qui permettent d'établir un appel, ainsi que les trames RTP et SRTP pour le transport de la voix.



(rtp or sip)

No.	Time	Source	Destination	Protocol	Length	Info
70	14.071444849	10.16.19.86	10.16.19.80	SIP/SDP	1191	Request: INVITE sip:1002@10.16.19.80 |
71	14.071641446	10.16.19.80	10.16.19.86	SIP	631	Status: 401 Unauthorized |
72	14.076858437	10.16.19.86	10.16.19.80	SIP	407	Request: ACK sip:1002@10.16.19.80 |
73	14.076926933	10.16.19.86	10.16.19.80	SIP/SDP	1355	Request: INVITE sip:1002@10.16.19.80 |
74	14.078282425	10.16.19.80	10.16.19.86	SIP	609	Status: 100 Trying |
75	14.080204349	10.16.19.80	10.17.1.10	SIP/SDP	907	Request: INVITE sip:Fanvil@10.17.1.10:5703 |
76	14.105872896	10.17.1.10	10.16.19.80	SIP	397	Status: 100 Trying |
78	14.371043391	10.17.1.10	10.16.19.80	SIP	525	Status: 180 Ringing |
79	14.371260525	10.16.19.80	10.16.19.86	SIP	625	Status: 180 Ringing |
93	17.031184313	10.17.1.10	10.16.19.80	SIP/SDP	807	Status: 200 OK (INVITE) |
94	17.031491819	10.16.19.80	10.17.1.10	SIP	451	Request: ACK sip:Fanvil@10.17.1.10:5703 |
95	17.031752913	10.16.19.80	10.16.19.86	SIP/SDP	1028	Status: 200 OK (INVITE) |
98	17.036207251	10.16.19.86	10.16.19.80	SRTP	224	PT=ITU-T G.711 PCMU, SSRC=0x7B9C4B0D, Seq=16106, Time=160, Mark
99	17.036362631	10.16.19.80	10.17.1.10	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x629F218B, Seq=1948, Time=160
100	17.038358775	10.16.19.86	10.16.19.80	SIP	383	Request: ACK sip:1002@10.16.19.80:5060 |
105	17.056289970	10.16.19.86	10.16.19.80	SRTP	224	PT=ITU-T G.711 PCMU, SSRC=0x7B9C4B0D, Seq=16107, Time=320
106	17.056458584	10.16.19.80	10.17.1.10	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x629F218B, Seq=1949, Time=320
107	17.076514241	10.16.19.86	10.16.19.80	SRTP	224	PT=ITU-T G.711 PCMU, SSRC=0x7B9C4B0D, Seq=16108, Time=480

Figure 10 – *Trame Wireshark d'une échange entre deux téléphones prt. 1*



(rtp or sip)

No.	Time	Source	Destination	Protocol	Length	Info
1787	25.413981324	10.16.19.80	10.16.19.86	SRTP	224	PT=ITU-T G.711 PCMU, SSRC=0x7D64345F, Seq=8150, Time=1112042336
1788	25.417037698	10.16.19.86	10.16.19.80	SRTP	224	PT=ITU-T G.711 PCMU, SSRC=0x7B9C4B0D, Seq=16525, Time=67200
1789	25.417122808	10.16.19.80	10.17.1.10	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x629F218B, Seq=2367, Time=67200
1790	25.433820315	10.17.1.10	10.16.19.80	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x34E94EB0, Seq=31502, Time=1112042497
1791	25.434017152	10.16.19.80	10.16.19.86	SRTP	224	PT=ITU-T G.711 PCMU, SSRC=0x7D64345F, Seq=8151, Time=1112042496
1792	25.437532251	10.16.19.86	10.16.19.80	SRTP	224	PT=ITU-T G.711 PCMU, SSRC=0x7B9C4B0D, Seq=16526, Time=67360
1793	25.437623890	10.16.19.80	10.17.1.10	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x629F218B, Seq=2368, Time=67360
1794	25.453909877	10.17.1.10	10.16.19.80	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x34E94EB0, Seq=31503, Time=1112042657
1795	25.454042225	10.16.19.80	10.16.19.86	SRTP	224	PT=ITU-T G.711 PCMU, SSRC=0x7D64345F, Seq=8152, Time=1112042656
1796	25.457589256	10.16.19.86	10.16.19.80	SRTP	224	PT=ITU-T G.711 PCMU, SSRC=0x7B9C4B0D, Seq=16527, Time=67520
1797	25.457668992	10.16.19.80	10.17.1.10	RTP	214	PT=ITU-T G.711 PCMU, SSRC=0x629F218B, Seq=2369, Time=67520
1799	25.460843215	10.17.1.10	10.16.19.80	SIP	451	Request: BYE sip:Cisco@10.16.19.80:5060 |
1800	25.461081006	10.16.19.80	10.17.1.10	SIP	520	Status: 200 OK (BYE) |
1801	25.461504830	10.16.19.80	10.16.19.86	SIP	654	Request: BYE sip:Cisco@10.16.19.86:59413;ob |
1802	25.461915257	10.16.19.86	10.16.19.80	SIP	339	Status: 200 OK (BYE) |
1808	25.582954789	10.16.19.80	10.17.1.10	SIP	619	Request: OPTIONS sip:Fanvil@10.17.1.10:5703 |
1809	25.616371533	10.17.1.10	10.16.19.80	SIP	563	Status: 200 OK (OPTIONS) |

Figure 11 – *Trame Wireshark d'une échange entre deux téléphones prt. 2*

3 Accéder à l'administration d'Asterisk via son serveur HTTP

Pour administrer Asterisk grâce à son serveur HTTP, il faut pour cela installer le framework Asterisk GUI, et, il y a plusieurs paramètres à prendre en compte :

— La version d'Asterisk, au minimum 1.4
— L'adresse IP souhaité
— Le port souhaité
— L'utilisation que l'on souhaitera en faire : HTML, CSS, Javascript, etc...

Alpine Linux 15

Pour connaître la version d'Asterisk
asterisk -r

La version Asterisk installé est la 18.2.2
Plusieurs composants représentent Asterisk GUI :

— Asterisk Manager Interface
— AJAM et Javascript
— Gestionnaire via HTTP et le serveur web Asterisk

Pour commencer, téléchargeons le code source :

Alpine Linux 16

Ensemble d'outils nécessaires pour compiler des logiciels à partir de leur code source
apk add build-base
Pour gérer les dépôts Git
apk add git
Cloner le dépôt
git clone ://github.com/wardmundy/asterisk-gui-2.0.git asterisk-gui

Et installons le code source :

Alpine Linux 17

Se déplacer dans le dépôt

```
cd asterisk-gui
```

Lancer le script de configuration

```
./configure
```

Copier les fichiers compilés dans les répertoires système appropriés

```
make install
```

Vérifier que la configuration a été correctement définie

```
make checkconfig
```

Passons à la configuration des fichiers */etc/asterisk/http.conf* et */etc/asterisk/manager.conf*.

La configuration du serveur Web Asterisk pour traiter les requêtes comporte plusieurs étapes simples.

Dans le fichier */etc/asterisk/http.conf* :

Alpine Linux 18

```
[general]
enabled = yes
enablestatic = yes
webenabled = yes
bindaddr = 0.0.0.0
bindport = 8088
prefix = asterisk
redirect = / /asterisk/static/config/index.html
httptimeout=600

[postmappings]
backups=/var/lib/asterisk/gui_backups
moh=/var/lib/asterisk/moh
```

Maintenant, le fichier */etc/asterisk/manager.conf* sert à la gestion des droits et à la création des profils utilisateurs, il permet notamment de définir les actions possibles par utilisateur, pour cela :

Alpine Linux 19

```
[general]
enabled = yes
port = 5038
bindaddr = 0.0.0.0
bindport = 8088
webenabled = yes

[admin]
secret = progtr00
read = system,call,log,verbose,agent,user,config,dtmf,reporting,cdr,dialplan
write = system,call,agent,user,config,command,reporting,originate,message
```

Pour que le GUI accède au dossier, il faut attribuer les droits suivants :

Alpine Linux 20

```
chown asterisk :asterisk /etc/asterisk/extensions.conf
chmod 664 /etc/asterisk/extensions.conf
chown -R asterisk :asterisk /var/lib/asterisk/static-http/config/
chmod -R 775 /var/lib/asterisk/static-http/config/
```

Nous pouvons redémarrer le service :

Alpine Linux 21

```
rc-service asterisk restart
```

Vérifions maintenant dans le nagivateur (*http ://localhost :8088*) :

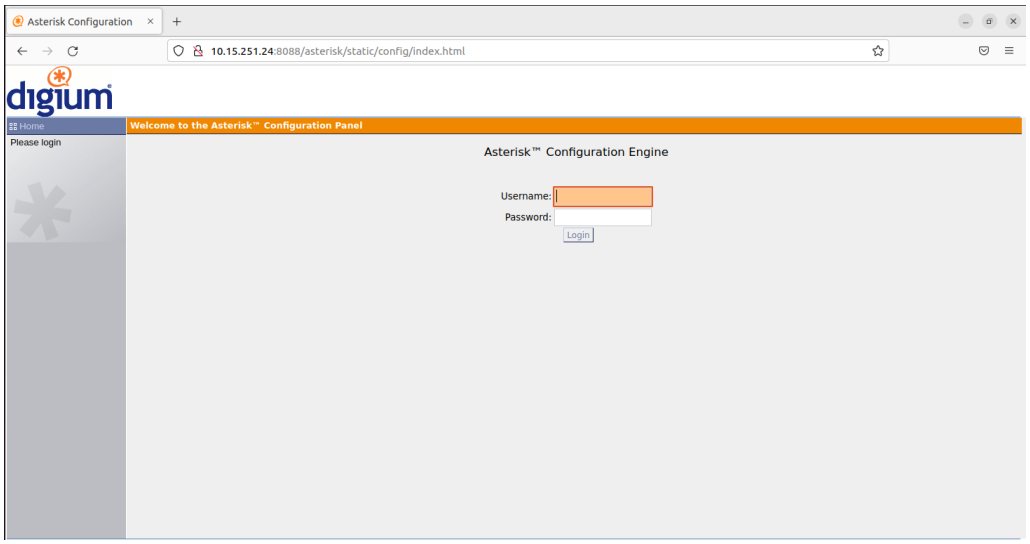


Figure 12 – *Interface de connexion GUI Asterisk*

Une fois connecté, l'interface d'administration permet de gérer le dialplan, les logs, d'ajouter des extensions, de créer des salles de conférences, des boîtes vocales, et toute choses réalisables avec Asterisk.

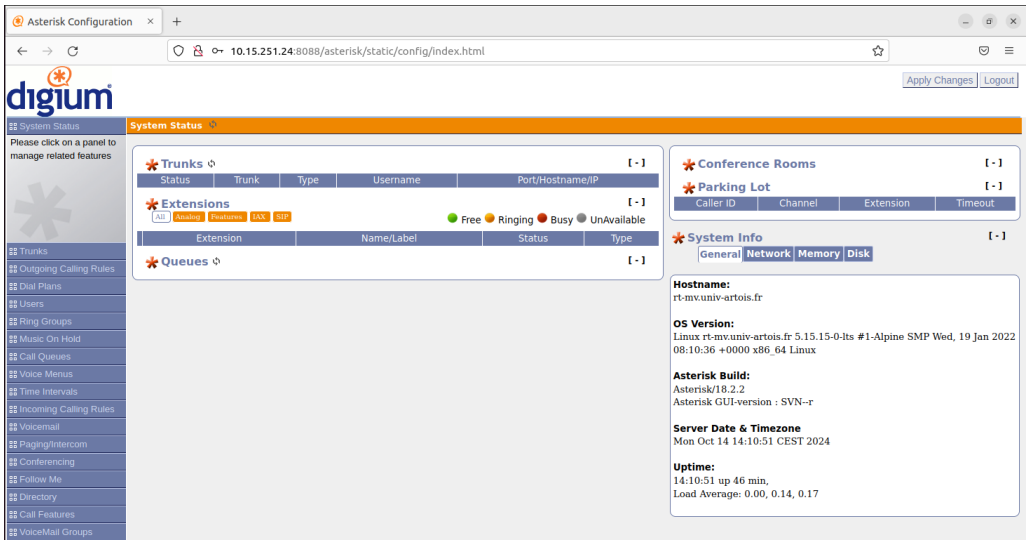


Figure 13 – *Interface d'administration GUI Asterisk*

4 Déployer un serveur WEB sur une machine virtuelle avec la distribution Alpine

4.1 Installation du serveur web

Par expérience et facilité, nous utiliserons un serveur Apache2.

Alpine Linux 22

Installer le serveur web

apk add apache2

Démarrer Apache2

rc-service start apache2

Vérifier son statut

rc-service status apache2

Ajouter le service au démarrage

rc-update add apache2

4.2 Test d'accès au serveur avec création d'une page d'accueil

Les fichiers HTML se situe dans le répertoire */var/www/localhost/htdocs/index.html* :

Figure 14 – *Page simple d'Apache2*

5 Intégrer une page WEB sur le serveur précédent pour réaliser un échange voix à l'aide de la librairie sip.js

5.1 Résumé du protocole WebRTC (Web Real-Time Communication)

WebRTC est une technologie qui permet d'ajouter des fonctionnalités de communication en temps réel à des applications web. Voici quelques points clés à retenir :

- Communication Peer-to-Peer : WebRTC prend en charge l'envoi de données vidéo, vocales et génériques entre pairs (peer-to-peer). Cela signifie que le flux média ne transite pas par un serveur centralisé, mais circule directement entre les navigateurs des utilisateurs.
- Open Source : WebRTC est un projet open source, ce qui permet à une large communauté de développeurs de contribuer à son amélioration et à son évolution.
- Compatibilité : Cette technologie est compatible avec les principaux navigateurs et plateformes, notamment Apple, Google, Microsoft et Mozilla.

En résumé, WebRTC offre une solution efficace et flexible pour intégrer des communications en temps réel dans les applications web, tout en garantissant une interopérabilité entre différents systèmes.

Packages nécessaires pour l'installation :

Alpine Linux 23

```
apk add ncurses
apk add sqlite
apk add libsrtp
apk add libuuid
```

5.2 Mise en place d'une communication entre deux PC grâce à sip.js

Tout d'abord, il faut vérifier sur la console Asterisk que certains modules sont activés :

Alpine Linux 24

module show like res_http_websocket

module show like chan_pjsip

module show like res_crypto

```
rt-mv:/etc/asterisk# asterisk -rvvv
Asterisk 18.2.2, Copyright (C) 1999 - 2018, Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for details.
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 18.2.2 currently running on rt-mv (pid = 2595)
Core debug is still 3.
rt-mv*CLI> module show like res_crypto
Module                          Description                          Use Count  Status      Suppo
rt Level
res_crypto.so                   Cryptographic Digital Signatures     4          Running
    core
1 modules loaded
rt-mv*CLI> module show like chan_pjsip
Module                          Description                          Use Count  Status      Suppo
rt Level
chan_pjsip.so                   PJSIP Channel Driver                 0          Running
    core
1 modules loaded
rt-mv*CLI> module show like res_http_websocket
Module                          Description                          Use Count  Status      Suppo
rt Level
res_http_websocket.so           HTTP WebSocket Support               4          Running
extended
1 modules loaded
```

Figure 15 – *Vérification des modules*

pjsip.conf fonctionne d'autre manière que *sip.conf*, sur ces points ci-dessous :

1. Technologie sous jacente
   - sip.conf : utilise chan_sip, un ancien module SIP dans Asterisk
   - pjsip.conf : utilise chan_pjsip, une implémentation plus moderne basée sur la bibliothèque PJSIP
2. État et avenir
   - chan_sip (sip.conf) : considéré comme obsolète et déprécié dans les versions récentes d'Asterisk

— chan_pjsip (pjsip.conf) : remplaçant officiel de chan_sip, plus performant et robuste, recommandé pour toutes les nouvelles configurations, prend en charge des fonctionnalités avancées non disponibles dans chan_sip

3. Gestion des connexions
   — sip.conf (chan_sip) : chaque connexion SIP utilise un seul thread, ce qui limite la performance, plus complexe à gérer lorsque le nombre de sessions augmente
   — pjsip.conf (chan_pjsip) : utilise un modèle multi-thread beaucoup plus efficace, peut gérer un grand nombre de connexions simultanées

Voici une configuration basique de *pjsip.conf* :

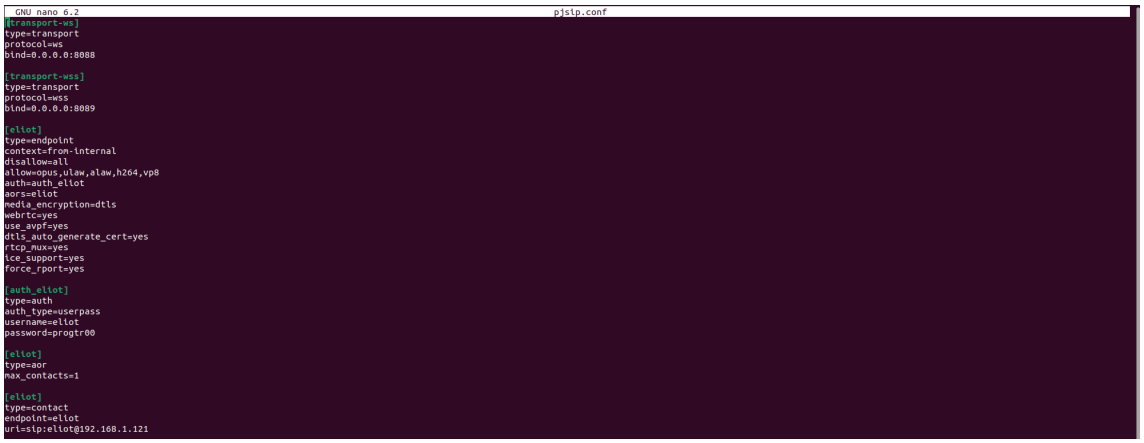


Figure 16 – *pjsip.conf pt. 1*

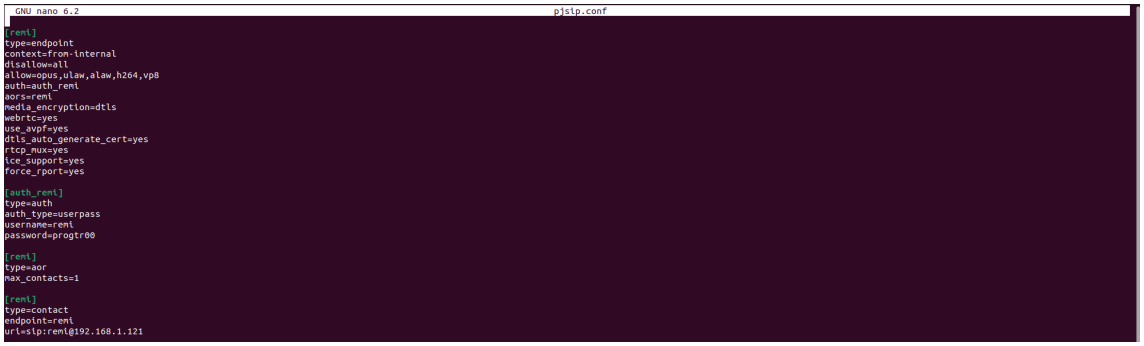


Figure 17 – *pjsip.conf pt. 2*

```
[auth_victor]
type=auth
auth_type=userpass
username=victor
password=progtr00

[victor]
type=aor
max_contacts=1

[victor]
type=contact
endpoint=victor
uri=sip:victor@192.168.1.121
```

Figure 18 – *pjsip.conf pt. 3*

Et simplement les déclarer dans *extensions.conf* :

```
[iron-internal]
exten => eliot,1,Dial(PJSIP/eliot,30)
exten => eliot,n,Hangup()

exten => remi,1,Dial(PJSIP/remi,30)
exten => remi,n,Hangup()

exten => victor,1,Dial(PJSIP/victor,30)
exten => victor,n,Hangup()
```

Figure 19 – *extensions.conf*

Maintenant sur le serveur web, authentifions les client avec une page de connexion et javascript ainsi que la librairie sip.js afin de stocker en mémoire le websocket pour uniformer le portail web.

La librairie a été téléchargé en local pour accélérer les requêtes et éviter toutes potentielles erreurs de chargement.

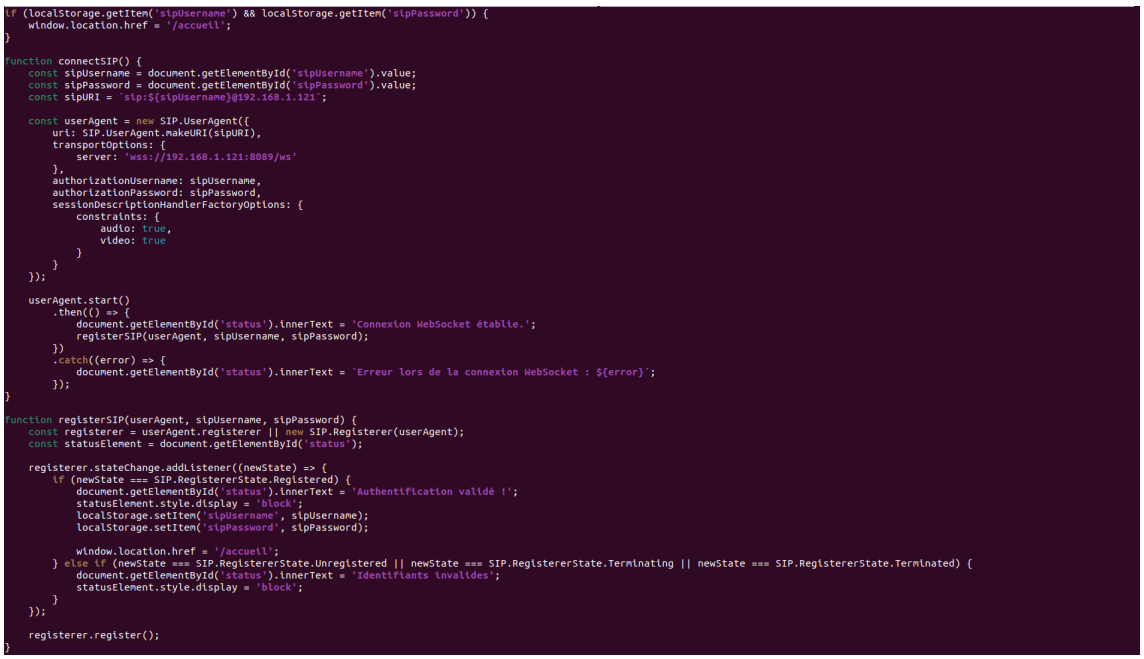
Côté HTML, mettons des IDs dans les input et un appel de fonction dans le button :

```
<form id="sipLoginForm">
    <h3>S'identifier</h3>
    <label for="sipUsername">Utilisateur</label>
    <input type="text" id="sipUsername" name="sipUsername" required>
    <label for="sipPassword">Mot de passe</label>
    <input type="password" id="sipPassword" name="sipPassword" required>
    <button class="button" type="button" onclick="connectSIP()">Connexion</button>
    <p class="bubble" id="status"></p>
</form>
```

Figure 20 – *HTML de la page login*

Le code Javascript implémente une connexion SIP. Si des identifiants SIP (sipUsername et sipPassword) existent déjà dans le localStorage, l'utilisateur est automatiquement redirigé vers la page /accueil. La fonction connectSIP récupère les identifiants saisis par l'utilisateur, construit l'URI SIP et initialise un SIP.UserAgent avec les informations

de connexion (URI SIP, serveur WebSocket sécurisé en wss, et autorisations). Une fois la connexion WebSocket établie, elle appelle registerSIP, qui enregistre l'utilisateur sur le serveur SIP via un Registerer. Le statut de l'enregistrement est surveillé : si l'enregistrement réussit, un message de succès est affiché, les identifiants sont stockés dans localStorage, et l'utilisateur est redirigé vers /accueil. En cas d'erreur d'authentification ou d'autres états invalides, un message d'erreur est affiché à l'utilisateur.



```
if (localStorage.getItem('sipUsername') && localStorage.getItem('sipPassword')) {
    window.location.href = '/accueil';
}

function connectSIP() {
    const sipUsername = document.getElementById('sipUsername').value;
    const sipPassword = document.getElementById('sipPassword').value;
    const sipURI = `sip:${sipUsername}@192.168.1.121`;

    const userAgent = new SIP.UserAgent({
        uri: SIP.UserAgent.makeURI(sipURI),
        transportOptions: {
            server: 'wss://192.168.1.121:8089/ws'
        },
        authorizationUsername: sipUsername,
        authorizationPassword: sipPassword,
        sessionDescriptionHandlerFactoryOptions: {
            constraints: {
                audio: true,
                video: true
            }
        }
    });

    userAgent.start()
        .then(() => {
            document.getElementById('status').innerText = 'Connexion WebSocket établie.';
            registerSIP(userAgent, sipUsername, sipPassword);
        })
        .catch((error) => {
            document.getElementById('status').innerText = `Erreur lors de la connexion WebSocket : ${error}`;
        });
}

function registerSIP(userAgent, sipUsername, sipPassword) {
    const registerer = userAgent.registerer || new SIP.Registerer(userAgent);
    const statusElement = document.getElementById('status');

    registerer.stateChange.addListener((newState) => {
        if (newState === SIP.RegistererState.Registered) {
            document.getElementById('status').innerText = 'Authentification validé !';
            statusElement.style.display = 'block';
            localStorage.setItem('sipUsername', sipUsername);
            localStorage.setItem('sipPassword', sipPassword);

            window.location.href = '/accueil';
        } else if (newState === SIP.RegistererState.Unregistered || newState === SIP.RegistererState.Terminating || newState === SIP.RegistererState.Terminated) {
            document.getElementById('status').innerText = 'Identifiants invalides';
            statusElement.style.display = 'block';
        }
    });

    registerer.register();
}
```

Figure 21 – *JS de la page login*

Une fois authentifié, l'utilisateur peut alors passer des appels, nous avons créer deux routes (call et videocall) afin de différencier appels vidéos et appels audios.

HTML de la page appels vidéos et audios (celle-ci elle est la même car la différence entre audio et vidéo se joue sur une variable, nous verrons dans le Javascript) :

```
</nav>
</header>
<main id="main">
    <video id="localVideo" autoplay playsinline></video>
    <video id="remoteVideo" autoplay playsinline></video>
    <div class="accueil">
    <input type="text" id="destinationInput" placeholder="Contact à appeler">
    <button id="callButton">Start Call</button>
    <button id="hangupButton">Hangup</button>
```

Figure 22 – *Code HTML pour passer des appels*

Le Javascript va quant à lui, appeler des fonctions de la librairie sip.js pour passer

des appels :

```
const localVideo = document.getElementById('localVideo');
const remoteVideo = document.getElementById('remoteVideo');
const callButton = document.getElementById('callButton');
const hangupButton = document.getElementById('hangupButton');
const destinationInput = document.getElementById('destinationInput');

let simpleUser;

async function main() {
    const server = "wss://192.168.1.121:8089/ws";
    const sipUsername = localStorage.getItem('sipUsername');
    const sipPassword = localStorage.getItem('sipPassword');

    if (!sipUsername || !sipPassword) {
        alert("Veuillez vous connecter.");
        window.location.href = '/';
        return;
    }

    const aor = `sip:${sipUsername}@192.168.1.121`;
    const options = {
        aor,
        media: {
            constraints: {
                audio: true,
                video: false
            },
            local: {
                video: localVideo
            },
            remote: {
                video: remoteVideo
            }
        },
        userAgentOptions: {
            authorizationUsername: sipUsername,
            authorizationPassword: sipPassword
        }
    };

    simpleUser = new SIP.Web.SimpleUser(server, options);
    callButton.addEventListener('click', async () => {
        try {
            await simpleUser.connect();
            await simpleUser.register();

            const destination = destinationInput.value;
            if (!destination) {
                alert("Ecrivez un bon URI");
                return;
            }

            await simpleUser.call(destination);

            messagesContainer.innerHTML = '';
        } catch (error) {
            console.error('Erreur : ', error);
            alert('Erreur : ' + error.message);
        }
    });
```

Figure 23 – *Code JS pour passer des appels pt. 1*

```
    hangupButton.addEventListener('click', async () => {
        try {
            await simpleUser.hangup();
            messagesContainer.innerHTML = '';
        } catch (error) {
            console.error('Erreur : ', error);
        }
    });
}

main()
    .then(() => console.log('OK'))
    .catch((error) => console.error('Erreur : ', error));
```

Figure 24 – *Code JS pour passer des appels pt. 2*

Ce code initialise les éléments vidéo locaux et distants (désactivés pour les appels audios), ainsi que les boutons pour passer et raccrocher des appels. Il vérifie les identifiants SIP depuis le localStorage. En cliquant sur le bouton d'appel, l'utilisateur initie un appel vers une destination SIP saisie. Le bouton de raccrochage termine l'appel en cours. Enfin, la fonction principale main() est exécutée pour démarrer l'application.

6 Ajout sur l'interface précédente de la vidéo

Pour ajouter la vidéo, le code Javascript est identique au code Javascript des appels audios. Simplement, dans les options SIP, nous venons mettre la valeur de *video* à *true*.

Le CSS se chargera de centrer les balises localVideo et remoteVideo.

```
const options = {
    aor,
    media: {
        constraints: {
            audio: true,
            video: true
        }
```

Figure 25 – *Valeur true à la variable video*

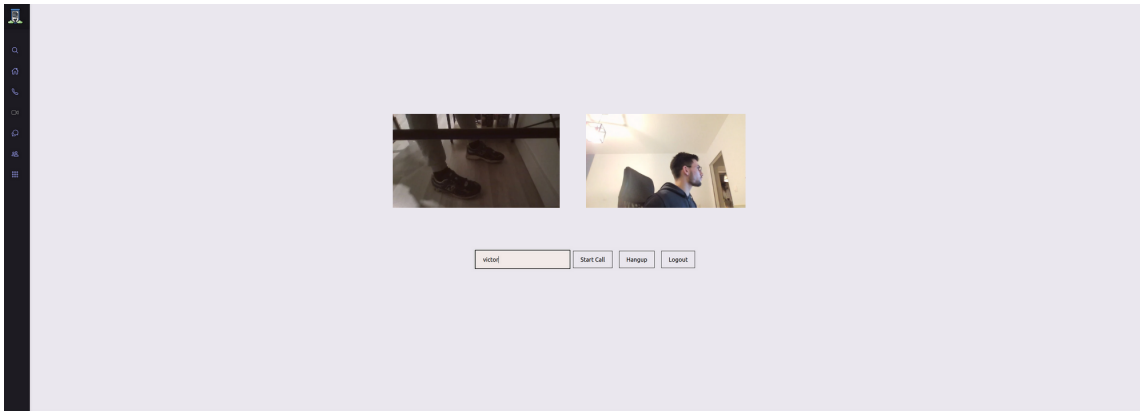


Figure 26 – *Appel vidéo*

7 Réalisation d'un portail web permettant de réaliser des visio-conférences

Pour installer un système de visioconférence, il faut éditer les fichiers confbridge.conf et extensions.conf :

```
[general]
; The general section of this config
; is not currently used, but reserved
; for future use.

; --- ConfBridge User Profile Options ---
[default_user]
type=user
video_codec = vp8,h264


; --- ConfBridge Bridge Profile Options ---
[default_bridge]
type=bridge
max_members=50

[sample_user_menu]
type=menu
*=playback_and_continue(conf-usermenu)
*1=toggle_mute
1=toggle_mute
*4=decrease_listening_volume
4=decrease_listening_volume
*6=increase_listening_volume
6=increase_listening_volume
*7=decrease_talking_volume
7=decrease_talking_volume
*8=leave_conference
8=leave_conference
*9=increase_talking_volume
9=increase_talking_volume

[sample_admin_menu]
type=menu
*=playback_and_continue(conf-adminmenu)
*1=toggle_mute
1=toggle_mute
*2=admin_toggle_conference_lock ; only applied to admin users
2=admin_toggle_conference_lock  ; only applied to admin users
*3=admin_kick_last      ; only applied to admin users
3=admin_kick_last       ; only applied to admin users
*4=decrease_listening_volume
4=decrease_listening_volume
*6=increase_listening_volume
6=increase_listening_volume
*7=decrease_talking_volume
7=decrease_talking_volume
*8=no_op
8=no_op
*9=increase_talking_volume
9=increase_talking_volume
```

Figure 27 – *confbridge.conf*

```
[from-internal]
exten => eliot,1,Dial(PJSIP/eliot,30)
exten => eliot,n,Hangup()

exten => remi,1,Dial(PJSIP/remi,30)
exten => remi,n,Hangup()

exten => victor,1,Dial(PJSIP/victor,30)
exten => victor,n,Hangup()

exten => 100,1,NoOp(Accès à la conférence)
 same => n,Answer()
 same => n,Wait(1)
 same => n,ConfBridge(100,default_bridge,default_user,default_menu)
 same => n,Hangup()
```

Figure 28 – *extensions.conf*

Sur le serveur web, nous avons recréer la même HTML, simplement, nous n'avons pu tester que l'audio, car nous n'avons pas réussi à gérer dynamiquement les balises vidéos pour *n* utilisateurs.

Le Javascript a la même base, simplement au lieu de passer un appel vers le résultat de la balise input, nous définissons en brut l'extension à appeler :

```
callButton.addEventListener('click', async () => {
    try {
        await simpleUser.connect();
        await simpleUser.register();

        const destination = "sip:100@192.168.1.121";
        if (!destination) {
            alert("Écrivez un bon URI");
            return;
        }

        await simpleUser.call(destination);
```

Figure 29 – *Javascript pour rejoindre la visioconférence*

8 Réalisation d'un audit de sécurité sur les échanges portail WEB/Asterisk

Pour que PJSIP fonctionne, il est obligatoire de passer par HTTPS, et donc ainsi de passer par TLS que ça soit sur Asterisk ou Apache2 (voir chapitre 9)

Ainsi, les sniff des requêtes HTTP sont bloqués car elles sont scryptés, et de même pour une attaque du type Man In The Middle.

9 Solutions pour sécuriser les échanges

9.1 Mise en place du protocole TLS

Pour mettre en place TLS au près d'Asterisk et d'Apache2, il faut créér des certificats autosignés pour Apache2 et Asterisk.

Ensuite, ces derniers seront à renseigner au près d'*http.conf* pour Asterisk et du *VirtualHost* pour Apache2.

Sur Apache2, nous forçerons l'HTTPS et bloqueront également les en-têtes.

Sur Asterisk :

Alpine Linux 25

sudo mkdir /etc/asterisk/keys
sudo contrib/scripts/ast_tls_cert -C 192.168.1.121 -O "EVR Asterisk" -b 2048 -d /etc/asterisk/keys

Et configurer le fichier */etc/asterisk/http.conf* (Attention : pour que la suite fonctionne, il faudra accepter le certificat à l'adresse *https ://192.168.1.121 :8089/ws*) :

```
rt-mv:/etc/asterisk# ls -l keys/
total 32
-rw-------    1 asterisk asterisk      1391 Oct 23 08:56 asterisk.crt
-rw-------    1 asterisk asterisk       928 Oct 23 08:56 asterisk.csr
-rw-------    1 asterisk asterisk      1675 Oct 23 08:56 asterisk.key
-rw-------    1 asterisk asterisk      3066 Oct 23 08:56 asterisk.pem
-rw-------    1 root     root           158 Oct 23 08:56 ca.cfg
-rw-------    1 root     root          1773 Oct 23 08:56 ca.crt
-rw-------    1 root     root          3311 Oct 23 08:56 ca.key
-rw-------    1 root     root           120 Oct 23 08:56 tmp.cfg
rt-mv:/etc/asterisk# _
```

Figure 30 – *Certificats et clés*

```
[general]
enabled = yes
enablestatic = yes
bindaddr = 0.0.0.0
bindport = 8088
websocket_enabled = yes
tlsenable = yes
tlsbindaddr = 0.0.0.0:8089
tlscertfile = /etc/asterisk/keys/asterisk.crt
tlsprivatekey = /etc/asterisk/keys/asterisk.key
[post_mappings]
backups = /var/lib/asterisk/gui_backups
moh = /var/lib/asterisk/moh
```

Figure 31 – *http.conf*

Et pour Apache2 :

Alpine Linux 26

sudo mkdir /etc/apache2/ssl

cd /etc/asterisk/ssl

sudo openssl req -x509 -newkey rsa :4096 -keyout key.pem -out cert.pem -days 365

Et configurer le VirtualHost :

```
<IfModule mod_ssl.c>
                <VirtualHost _default_:443>
                ServerAdmin webmaster@localhost

                DocumentRoot /var/www/html

                ErrorLog ${APACHE_LOG_DIR}/error.log
                CustomLog ${APACHE_LOG_DIR}/access.log combined

                SSLEngine on

                SSLCertificateFile /etc/apache2/ssl/cert.pem
                SSLCertificateKeyFile /etc/apache2/ssl/key.pem

                <Directory /var/www/html>
                        Options Indexes FollowSymLinks MultiViews
                        AllowOverride All
                        Require all granted
                </Directory>

                Header set Strict-Transport-Security "max-age=31536000; includeSubDomains"
                Header set X-Content-Type-Options "nosniff"
                Header set X-Frame-Options "SAMEORIGIN"
                Header set X-XSS-Protection: 1; mode=block

        </VirtualHost>
</IfModule>
<VirtualHost *:80>
        ServerName 192.168.1.121
        Redirect permanent / https://192.168.1.121/
</VirtualHost>
```

Figure 32 – *default-ssl.conf*

9.2 Vérification sur les échanges

Lors d'un appel entre deux PCs (192.168.1.160 et 192.168.1.130), on peut voir que les échanges se font via TLS1.3 ou TLS1.2 et que le transport se fait par TCP :

ip.addr == 192.168.1.121

No.	Time	Source	Destination	Protocol	Length	Info
272	8.393226771	192.168.1.121	192.168.1.131	TCP	60	8089 → 55267 [ACK] Seq=4892 Ack=10550 Win=63744 Len=0
273	8.393234195	192.168.1.121	192.168.1.131	TCP	60	[TCP Dup ACK 272#1] 8089 → 55267 [ACK] Seq=4892 Ack=10550 Win=637…
274	8.393270603	192.168.1.121	192.168.1.131	TCP	66	[TCP Dup ACK 272#2] 8089 → 55267 [ACK] Seq=4892 Ack=10550 Win=637…
275	8.393274370	192.168.1.121	192.168.1.131	TCP	66	[TCP Dup ACK 272#3] 8089 → 55267 [ACK] Seq=4892 Ack=10550 Win=637…
276	8.394276322	192.168.1.121	192.168.1.160	TLSv1.2	592	Application Data
277	8.394339100	192.168.1.160	192.168.1.121	TCP	66	41874 → 8089 [ACK] Seq=9649 Ack=4221 Win=501 Len=0 TSval=17945291…
361	18.952684477	192.168.1.160	192.168.1.121	TLSv1.2	486	Application Data
362	18.953557106	192.168.1.121	192.168.1.160	TLSv1.2	505	Application Data
363	18.953606558	192.168.1.160	192.168.1.121	TCP	66	41874 → 8089 [ACK] Seq=10069 Ack=4660 Win=501 Len=0 TSval=1794539…
364	18.954588112	192.168.1.121	192.168.1.131	TLSv1.3	573	Application Data
365	18.954610033	192.168.1.121	192.168.1.131	TCP	573	[TCP Retransmission] 8089 → 55267 [PSH, ACK] Seq=4892 Ack=10550 W…
366	18.980299556	192.168.1.131	192.168.1.121	TLSv1.3	386	Application Data
367	18.980332958	192.168.1.131	192.168.1.121	TCP	386	[TCP Retransmission] 55267 → 8089 [PSH, ACK] Seq=10550 Ack=5411 W…
368	18.980636799	192.168.1.121	192.168.1.131	TCP	60	8089 → 55267 [ACK] Seq=5411 Ack=10882 Win=64128 Len=0

Figure 33 – *Appel entre PCs avec TLS et TCP*