Projet Ilty : I'm listening to you (via VoIP) !

Nicolas Bareil

EADS/CCR
nbareil@mouarf.org

Introduction

Les nouvelles offres de téléphones utilisant la technologie IP sont de plus en nombreuses et paraissent toujours plus alléchantes. En effet, on nous propose une économie d'installation, de maintenance et des fonctionnalités merveilleuses tout en assurant une sécurité maximale. Qu'en est-il réellement ? Dans cet article, nous ne nous attacherons pas à tester les gadgets vendus mais nous allons étudier les problèmes que pose ce nouvel environnement de téléphonie.

En plus de pointer du doigt ces faiblesses connues[1], nous avons implémenté une centrale d'interception téléphonique nommée ilty que l'on présentera dans la deuxième partie de ce document.

Cet outil a été créé et développé pour *EADS CCR/SSI* afin de faciliter les tests d'intrusion (démonstration aisée des faiblesses du réseau, ainsi qu'apport de preuve de réussite) et de sensibiliser ses *business units* aux problèmes de sécurité. En effet, peu d'administrateurs et de décideurs imaginent que leurs conversations peuvent être écoutées.

Le terme VoIP signifie *Voice over IP*, ou voix sur IP. C'est à dire que la voix va être transportée de la même facon que le sont vos paquets de données vers un serveur Web lorsque vous naviguez sur Internet.

Tout comme les anciens réseaux téléphoniques des entreprises qui nécessitaient un central téléphonique (un PABX), un réseau VoIP nécessite un serveur jouant le rôle d'entremetteur entre les téléphones (appelés clients).

Au moins deux protocoles sont nécessaires aux communications : l'un qui va gérer la signalisation (les informations échangées entre le client et le serveur) et l'autre qui gère le transport de la voix.

Plusieurs protocoles de signalisation, propriétaires ou libres, s'affrontent, les plus connus sont SIP, Skinny et H.323. Leur rôle est de mettre en relation les téléphones entre eux, gérer les redirections de lignes, afficher des informations sur les écrans des téléphones, etc.). À l'heure actuelle, la majorité des réseaux utilisent le protocole de transport RTP afin d'acheminer la voix ou la vidéo. Mais voyons plus en détail les protocoles de signalisation SIP et Skinny.

SIP (Session Initialization Protocol) est un protocole pour les réseaux VoIP, il est formalisé dans diverses RFCs, dont la RFC 3261. C'est un protocole extrêmement complexe qui offre de nombreuses possibilités.

Les messages SIP échangés ressemblent fortement aux requêtes HTTP. Tout est texte. Par exemple, voici une demande d'appel :

[1] Voir la présentation de Nicolas FISCHBACH au SSTIC 2004

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776as...
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142
```

Cette façon de présenter les données diffère totalement de `Skinny`, le protocole élaboré par *Cisco*, qui préfère utiliser le format binaire. De la même façon que le protocole `DNS`, les champs sont positionnés à des endroits fixes.

Le protocole est ainsi beaucoup moins verbeux donc plus rapide à traiter. Par contre, l'analyse des flux par un béotien est impossible ! Il y a d'ailleurs très peu de documentation existante sur le protocole.

Le projet `ilty` supporte pour le moment les protocoles `SIP` et `Skinny` ; Il est prévu d'implémenter, dans le futur, le support de `H.323` et `Skype` (si ce dernier est documenté un jour).

Dans ce rapport, nous allons nous intéresser plus particulièrement au protocole `Skinny`. Tous les téléphones *Cisco* utilisent ce moyen de communication et il est donc largement répandu.

1 Étude des protocoles

1.1 Protocole de signalisation Skinny

Architecture globale Grâce aux réseaux de téléphonie sur `IP`, il est beaucoup plus facile de gérer la mobilité, on peut désormais se déplacer et conserver notre ligne téléphonique facilement. Mais afin de gérer cette mobilité, il est nécessaire d'en informer le central, le *Call Manager*, qui se charge de relier les téléphones entre eux (via le protocole de signalisation) comme l'illustre la figure 1.

Ainsi, lorsqu'une personne souhaite téléphoner, il est nécessaire de demander au *Call Manager* de créer une ligne entre les deux correspondants et une fois la mise en relation terminée, les deux clients vont se connecter directement entre eux via le protocole de transport sans avoir à passer par le central.

Toutes ces données transitent par l'intermédiaire des réseaux `IP`. Le protocole de signalisation `Skinny` utilise `TCP` qui est un moyen de communication fiable alors que `SIP` n'impose pas de protocole de transport, cela pouvant être aussi bien de l'`UDP` ou du `TCP` en fonction de la taille du message. Par contre, le transport de la voix est réalisé en `UDP` afin de privilégier la rapidité de transmission. De plus, on ne se soucie pas de la perte de paquets.

En plus de la mise en relation des téléphones, c'est au *Call Manager* de gérer les fonctionnalités du téléphone comme sa mise à l'heure automatique, l'affichage de caractères sur l'écran ou l'activation de la sonnerie d'appel.

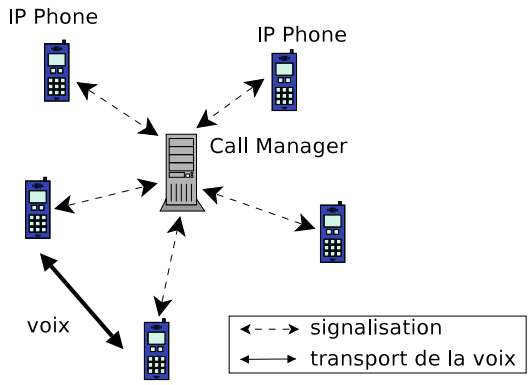

Fig. 1. Architecture d'un réseau VoIP

Nous allons voir comment sont composés les messages `Skinny` entre les téléphones et le *Call Manager* dans la partie suivante.

Déroulement du protocole Tous les messages `Skinny` comportent au minimum trois champs de quatre octets :

- un entier représentant la taille du message total
- un deuxième champ réservé qui doit toujours être à zéro
- un identifiant (ou *MessageId*) pour déterminer la nature du message

En fonction de ce *MessageId*, on peut savoir si c'est un message relatif à une demande d'appel, la réception d'un message, l'affichage de l'heure ou juste les touches qui sont composées sur le téléphone. En effet, un événement est envoyé immédiatement au *Call Manager* lorsque vous tapez sur une touche de votre téléphone, ainsi, une fois que vous avez composé suffisament de chiffres, le *Call Manager* commence tout de suite la mise en relation sans aucune validation de votre part.

La figure 2 montre les messages envoyés lors de la composition d'un numéro : lorsque le téléphone est libre et que l'on décroche le combiné, un message est envoyé au *Call Manager* qui nous répond par une notification de tonalité où l'on peut commencer à composer un numéro.

Pour chaque chiffre composé, un événement `Skinny` est envoyé au *Call Manager*, pour le premier chiffre tapé, le *Call Manager* indique au téléphone d'arrêter la tonalité puisque nous sommes en train de composer.

Lorsque le numéro est complet, le *Call Manager* met en relation les correspondants. Voyons plus en détail la construction d'un message `Skinny`.

Étude de traces La figure 3 montre un message envoyé du téléphone au serveur pour annoncer la pression sur une touche du téléphone :

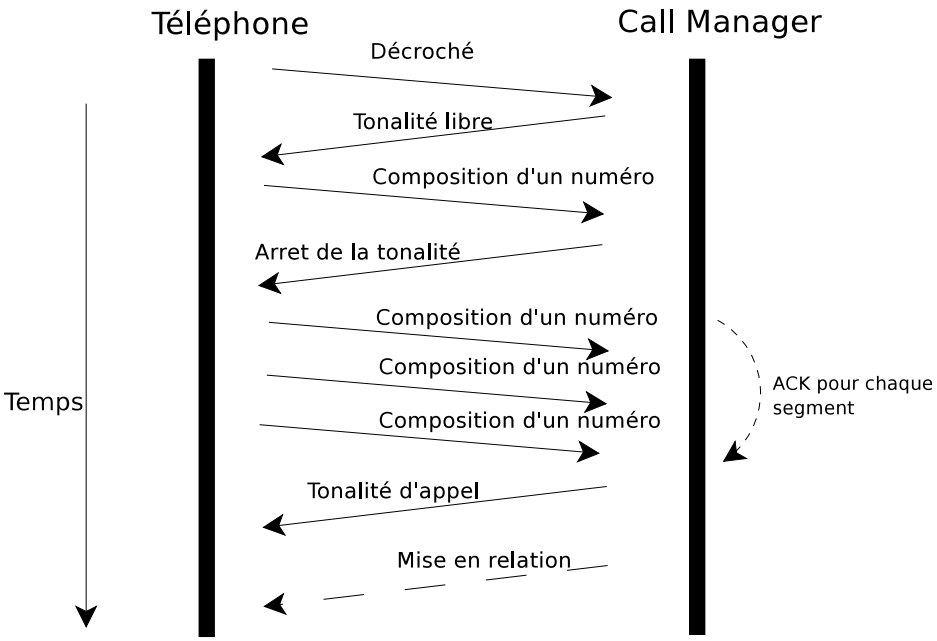

Fig. 2. Chronogramme d'un appel

```
45 60 00 40 2e c5 00 00 40 06 00 00 c0 a8 00 01
c0 a8 fe fe ca a7 07 d0 8c 52 31 d8 a0 8d a5 3b
50 18 05 78 d1 aa 00 00 10 00 00 00 00 00 00 00
03 00 00 00 08 00 00 00
```

MessageID — Valeur '8' — Longueur — Champ réservé

Fig. 3. Événement KeypadButtonPressed

Le client (`192.168.0.1`, ou `0xc0a80001`) indique via un message *KeypadButtonPressed* (`0x00000003`) au serveur (`192.168.254.254` ou `0xc0a8fefe`) que la touche 8 (`0x00000008`) a été pressée.

En retour, le *Call Manager* répond par un acquitement (`TCP PUSH + ACK`) annoncant également la coupure de la tonalité et attend le reste des touches.

La figure 4 montre ce message envoyé par le serveur avec un *MessageId* égal à *StopToneMessage* ou `0x83`.

Lorque toutes les étapes du protocole de signalisation sont achevées, les deux téléphones connaissent alors les adresses et les ports `UDP` et ils vont donc se

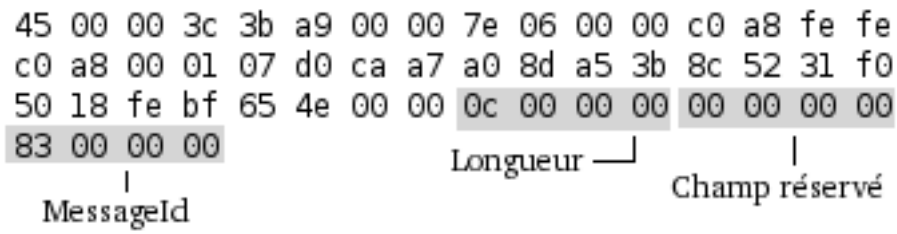

Fig. 4. Événement StopToneMesssage

connecter l'un à l'autre en utilisant un protocole de transport, tel que `RTP` détaillé ci-après.

1.2 Protocole d'échange de la voix (RTP)

Aperçu Le *Real Time Protocol* est un protocole généraliste : il est capable de transporter n'importe quel type de flux audio ou vidéo. L'intérêt de `RTP` est de gérer le séquencement (chaque message est identifié par un numéro de séquence et un *timestamp*, permettant de détecter la perte ou le retard de paquet) et de contenir un champ décrivant le type de compression utilisé, son échantillonnement, etc.

De cet en-tête suit directement le *payload* : la voix (ou la vidéo) numérisée.

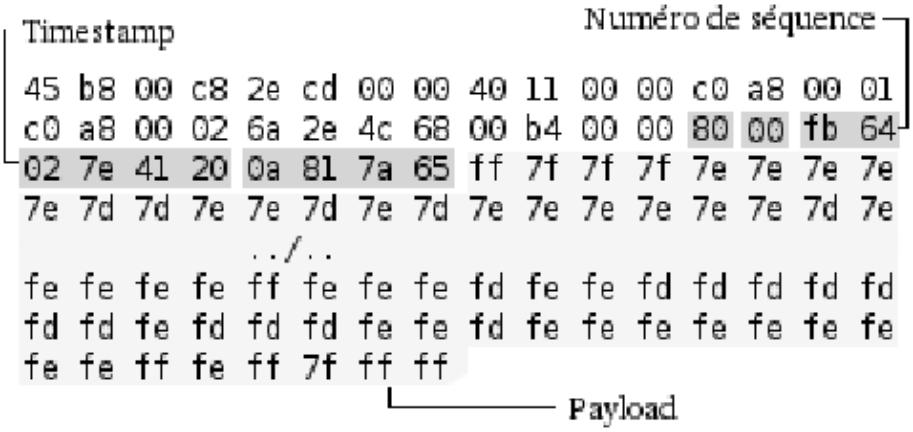

Fig. 5. Message RTP

Étude de traces La figure 5 montre un extrait de message `RTP` où apparaissent les différents champs :

- `0x80` : un champ paramètre qui contient plusieurs drapeaux (version, bourrage, extension, marqueur, etc.)
- `0x00` : le type de payload utilisé (`G.711`, `G.729`...)
- `0xfb64` : le numéro de séquence
- `0x027e4120` : un *timestamp*
- `0x0a817a65` : la source de synchronisation

Puis le *payload*.

Un message `RTP` contient quelques dixièmes de seconde de communication donc la perte d'un paquet n'est pas problématique, de plus, des algorithmes de compression ont été écrit afin de ne pas « sonner faux » en mixant les derniers paquets reçus.

Nous allons maintenant présenter le projet `ilty` qui implémente ces différents protocoles afin d'intercepter les communications téléphoniques.

2 Présentation d'ilty

2.1 Présentation générale

Aperçu `ilty` est un projet écrit en *Python* destiné à être modulaire et nécessitant peu d'intervention humaine. C'est une centrale d'interception téléphonique en environnement VoIP, il est capable d'enregistrer et/ou d'écouter les discussions en direct et changer de conversation facilement grâce à une interface textuelle listant toutes les communications en cours.

Aucune configuration n'est nécessaire, on installe le logiciel sur une machine placée sur le réseau physique (ou logique) des téléphones et `ilty` est opérationnel sans la moindre action humaine.

Objectif Nous avons vu dans la partie précédente les différents protocoles utilisés, en particulier `Skinny` et `RTP`, si nous arrivons à recevoir ces paquets, nous sommes alors en mesure d'écouter et de voir tout ce qu'il se passe sur le réseau étant donné qu'aucun chiffrement n'est réalisé par défaut.

Les informations récupérables sont les mêmes que celles transmises au téléphone : on voit ce qui est écrit sur l'écran, toutes les touches composées par l'utilisateur et surtout, la communication vocale.

Positionnement À l'heure actuelle, les logiciels travaillant sur du trafic VoIP (*vomit* et *voipong* par exemple) sont assez limités en fonctionnalités. *vomit* se contente de restituer une conversation enregistrée par une capture `tcpdump`.

voipong va plus loin et est capable d'enregistrer de multiples appels en parallèle en écoutant directement sur le réseau. Mais aucun de ces deux outils ne s'occupent de surveiller le protocole de signalisation, ce qu'`ilty` se charge de faire.

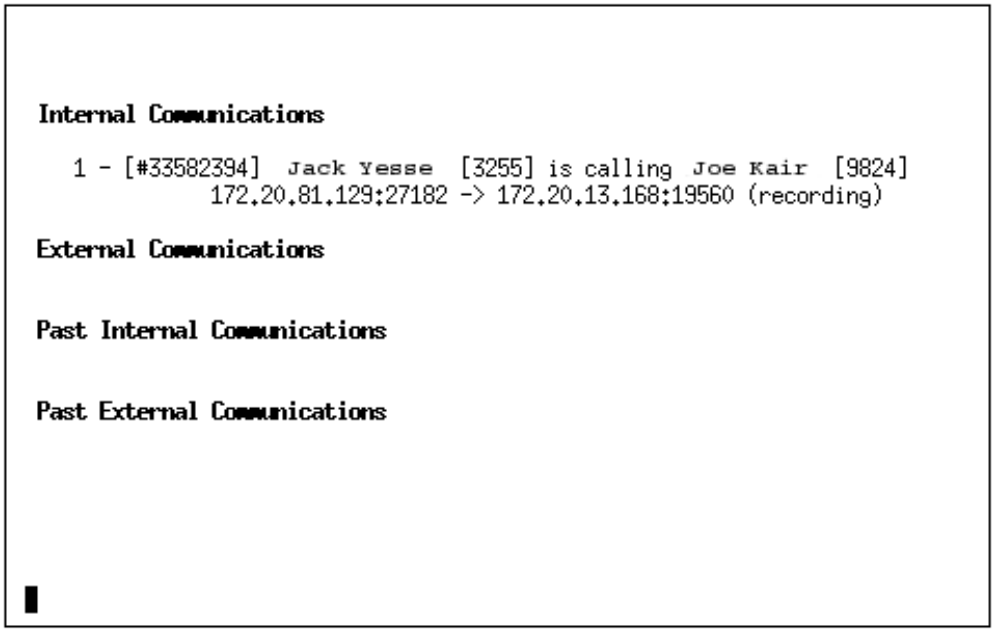

Fig. 6. Écran principal

Mise en situation du logiciel Au lancement d'`ilty`, on arrive sur l'interface présentée à la figure 6 présentant quatres types de conversation :

- communication interne actuelle
- communication externe actuelle
- communication interne passée
- communication externe passée

À partir de cette liste, on peut choisir une communication (en la sélectionnant par son numéro) et lui appliquer une action : écoute sur les hauts parleurs de la discussion en direct ou en différé si la discussion avait été enregistrée.

On enregistre toujours un historique des communications indiquant les dates de début et de fin ainsi que les différentes informations au sujet des correspondants.

Pour sélectionner une conversation, il faut d'entrer le numéro de la ligne (le chiffre tout à gauche, par exemple `1` sur l'écran ci-dessus) puis un menu nous propose les actions applicables : `record`, `listen`, ou `view`. Cette dernière fonction nous permet d'accéder à l'affichage des écrans des téléphones tel que représentés à la figure 7 (qui nous montre toutes les étapes d'établissement d'un appel).

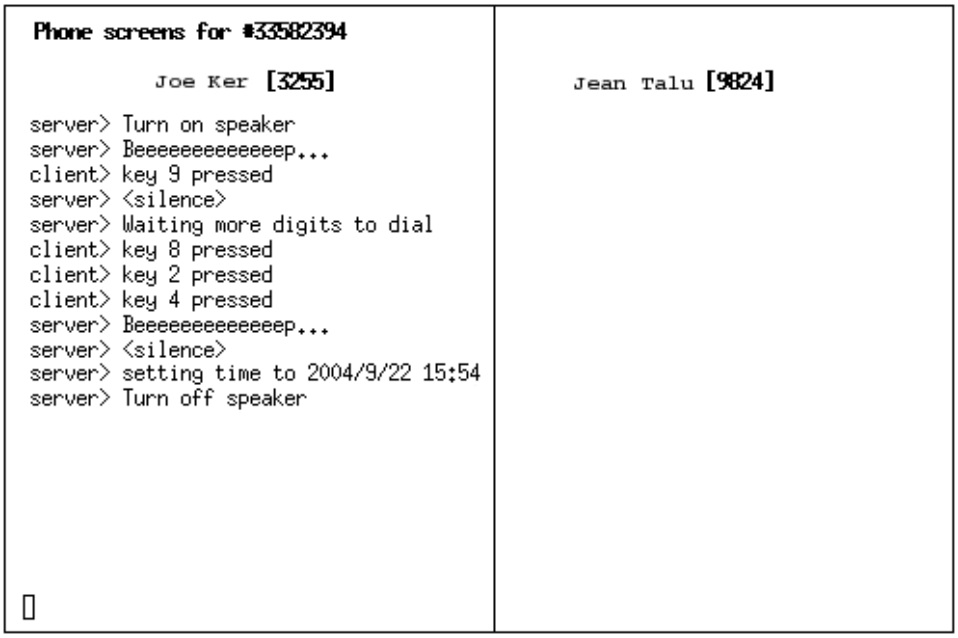

Fig. 7. Représentation des écrans des téléphones

2.2 Discussion sur son implémentation

Structure du programme `ilty` est composé de six classes importantes, une classe principale réalisant les opérations liées au réseau (initialisation de la `libpcap`, choix de l'interface d'écoute, traitement des protocoles `Ethernet`, `IP`, `UDP` et `TCP`) et une classe par protocole de signalisation (`SIP` et `Skinny`) et de transport de la voix (`RTP`).

Pour écouter les trames circulant sur notre interface, on s'y abonne via la `libpcap` en lui fournissant une fonction de retour (ou *callback*). Celle-ci sera appelée à chaque remontée de trame.

Notre *callback* appelle ensuite une fonction qui comprend le protocole de la couche supérieure (la couche `Ethernet` appelle la couche `IP` par exemple) et ainsi de suite. La figure 8 représente cette succession d'appels.

Toutes les informations au sujet des protocoles sont stockées dans une table se maintenant en permanence à jour grâce au protocole de signalisation. Afin de simplifier son actualisation, seule la classe `TrafficManagement` a la capacité de modifier la table. Une instance de cette classe est initialisée au démarrage et est fournie à toutes les autres instances.

Détection des appels Contrairement aux autres outils qui utilisent une heuristique de détection d'appel, `ilty` suit le protocole de signalisation et gère une véritable pile VoIP afin de reconnaitre les appels et les surveiller.

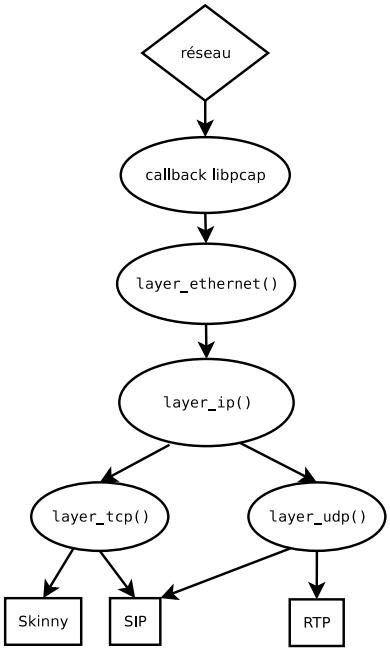

Fig. 8. Structure logique du programme

Dans le protocole Skinny, on peut considérer un appel comme validé à la réception des messages *OpenReceiveChannelAcknowledgment* et *StartMediaTransmission*.

L'inconvénient du `RTP` est que les téléphones communiquent entre eux (en `UDP`) sur des numéros de port variables échangés via le protocole de signalisation. Si on se force à ne pas interpréter la signalisation, on peut deviner un flux `RTP` en se basant sur la taille des datagrammes échangés, la durée entre deux paquets, ou une interprétation forcée des données (on regarde si les champs sont « cohérents » avec le numéro de séquence ou le *timestamp* par exemple).

ARP Cache poisoning Les réseaux des entreprises reposent en général sur le protocole `Ethernet` où tous les ordinateurs sont connectés entre eux via un équipement réseau. Lorsque ce matériel reçoit une trame depuis l'un de ses ports, il doit la transmettre à son destinataire final, mais ce mode de transmission varie en fonction de sa nature : un *hub* ou un *switch*. En effet, un *hub* va diffuser la trame reçue à tous les ports connectés, c'est à dire que tous les ordinateurs du réseau vont recevoir du trafic qui ne leur est pas destiné.

Le *switch* tente de faire mieux en maintenant une table locale indiquant sur quel port est joignable telle personne (son adresse `MAC`). Cette table est alors consultée lorsqu'une trame est reçue afin de n'envoyer la trame que sur un seul port. En théorie, cela signifie qu'un ordinateur connecté au *switch* ne recevra

que les données le concernant. Certains administrateurs ont alors cru trouver la solution parfaite pour sécuriser leur réseau (empêcher l'écoute passive).

Mais une méthode très populaire[2] existe pour ruiner cet espoir : la corruption de cache `ARP`. L'attaque consiste à polluer la table locale de la victime afin de rediriger le trafic vers nous.

L'inconvénient de cette attaque est qu'elle reste limitée au réseau local, car l'`ARP` ne franchit pas les barrières des routeurs. Il existe alors d'autres techniques de détournement de trafic détaillées dans la présentation « Protocoles Réseau : Grandeur et décadence »[3].

Traitement audio par sox Lors de l'établissement d'une communication, les téléphones se mettent d'accord sur le format d'encodage/décodage des données, ils s'accordent souvent à utiliser du `G.711` afin de transmettre la voix numérisée.

Il est donc nécessaire de décoder ce flux pour le faire passer dans les hauts parleurs. Plutôt que de réinventer la roue (et surtout après des essais infructueux), nous avons décidé d'utiliser le programme `sox`[4] qui est capable de tout décoder très simplement.

Une fois la décompression effectuée, il faut rendre le son sur les enceintes, mais bien que le traitement des paquets soit séquentiel, on reçoit plusieurs paquets de « voix » pour un même instant, en effet, les correspondants s'envoient simultanément le son de leur micro (sauf si c'est un silence et que la détection des silences est activée). Il est donc nécessaire d'utiliser plusieurs canaux (un pour chaque interlocuteur) de voix avant de les mixer et sortir ce mélange sur les hauts parleurs.

Ce démultiplexage est réalisé par *Esound* (plus connu sous le nom d'`esd`) qui va fusionner tous les canaux. Pour cela, on envoie la voix numérisée à `sox` qui va la décoder et transmettre (via un tube) ce flux à `esdcat`, qui lui même communiquera avec `esd`.

Ainsi, pour chaque partie d'une conversation écoutée, on créé deux (en fait trois) processus comme ceci :

```
(sox  -Ub -r 8000 -t .raw - -t .ub -|esdcat -b -m -r 8000)
   > /dev/null 2>&1
```

3 Sécurisation

3.1 D'un point de vue réseau

Utiliser des Vlans ? Un `vlan` est un domaine de *broadcast ethernet* logique, c'est à dire un réseau logique complétement indépendant des autres. Cela agit comme un réseau physique à part entière. C'est pour cela qu'il est préconisé

[2] `http://www.arp-sk.org/`
[3] Conférence de Cédric Blancher, Nicolas Fischbach et Pierre Bétouin au *SSTIC 2005*
[4] `sox` est disponible sur `http://sox.sourceforge.net/`

de placer tous les téléphones sur un ou plusieurs `vlans` dédiés afin de ne pas interférer sur celui des postes de travail.

En théorie, il n'est pas possible de « déborder » de son réseau virtuel, mais une mauvaise configuration d'un *switch* peut permettre de sauter d'un `vlan` à un autre. En pratique, il y a peu de risque de tomber à nouveau sur ce type de problème si l'on suit à la lettre les recommendations du constructeur.

De plus, les `vlans` n'empêchent pas le branchement sauvage (avec ou sans l'usurpation d'adresse `MAC`), c'est à dire se brancher sur la prise `Ethernet` du téléphone et accéder au `vlan` convoité.

Empêcher l'ARP Cache Poisoning ? Quelques parades existent pour contrer la corruption de cache `ARP` :

- cache `ARP` statique, on entre en dur les adresses `MAC` des machines afin d'éviter la résolution `ARP`
- l'utilisation d'un *switch* fonctionnant sur les niveaux deux et trois afin de réaliser des associations « adresse `IP`, adresse `MAC` et numéro de port »
- l'utilisation de `pvlan`[5] qui permet de restreindre la communication entre les machines d'un même `vlan`. Par exemple, on peut demander à ce que des ordinateurs ne puissent communiquer qu'avec la passerelle mais pas entre eux. Cette isolation n'est pas forcément judicieuse dans le cas où les téléphones sont tous sur le même `vlan` puisque le transport de la voix est en *peer to peer* et nécessite donc à chaque poste de pouvoir se connecter à son voisin

La seule solution contre l'écoute passive via ce biais est d'utiliser intensivement le chiffrement et la signature cryptographique des données.

3.2 Chiffrement des communications

L'équipementier *Cisco* recommande d'utiliser intensivement le chiffrement des données grâce à *IPSec* ou `SRTP` (Secure `RTP`[6], l'équivalent du `RTP` mais avec le chiffrement du *payload*) mais cela nécessite un effort non-négligeable de configuration et de maintenance.

De plus, l'utilisation de cette technologie nécessite son support par le téléphone (que le constructeur ne manquera pas de facturer) or peu de constructeurs proposent cette fonctionnalité dans leurs gammes de produits.

Pire encore, le chiffrement des communications via *IPSec* augmente sensiblement la latence.

3.3 Comment détecter ce genre de logiciel ?

Étant donné qu'on se contente d'écouter les données sans rien modifier, il n'est pas possible de repérer `ilty`, seul la corruption de cache `ARP` est suspiscieux

[5] `http://www.cisco.com/en/US/tech/tk389/tk814/tk841/tsd\_technology\_support\_sub-protocol\_home.html`

[6] Standard proposé en Mars 2004 dans dans la `RFC 3711`

et donc détectable. Cela signifie que si on dispose le logiciel sur un *span port*, il ne sera pas détectable...

... Ou presque car il existe des méthodes expérimentales de repérer les machines en mode *promiscuous* (écoute passive du trafic) suivant des bugs dans certaines piles TCP/IP.

Conclusion

Le tableau reste assez noir concernant la sécurité de la téléphonie sur `IP`, en effet, nous venons de voir les principales mesures disponibles afin de rendre la téléphonie plus sûre et elles sont bien maigres sans un effort humain important : sans renfort de la cryptographie, il n'y a pas de protection efficace.

De plus, le fait d'utiliser `Ethernet` simplifie les choses pour les pirates puisque cette technologie est connue de tous et ne requiert pas de matériel autre qu'une prise et carte réseau.

Remerciements Merci à Philippe Biondi et Cédric Blancher pour leurs précieux conseils et leurs relectures.