

Interprétation des captures Wireshark - Partie 2

Introduction

Ce document présente l'interprétation des captures Wireshark pour les protocoles DHCP, DNS, mDNS et messages TCP/TLS, conformément aux exigences de la partie 2 du projet. L'analyse se concentre sur l'interprétation des paquets capturés selon les spécifications des protocoles, en utilisant toutes les captures d'écran fournies.

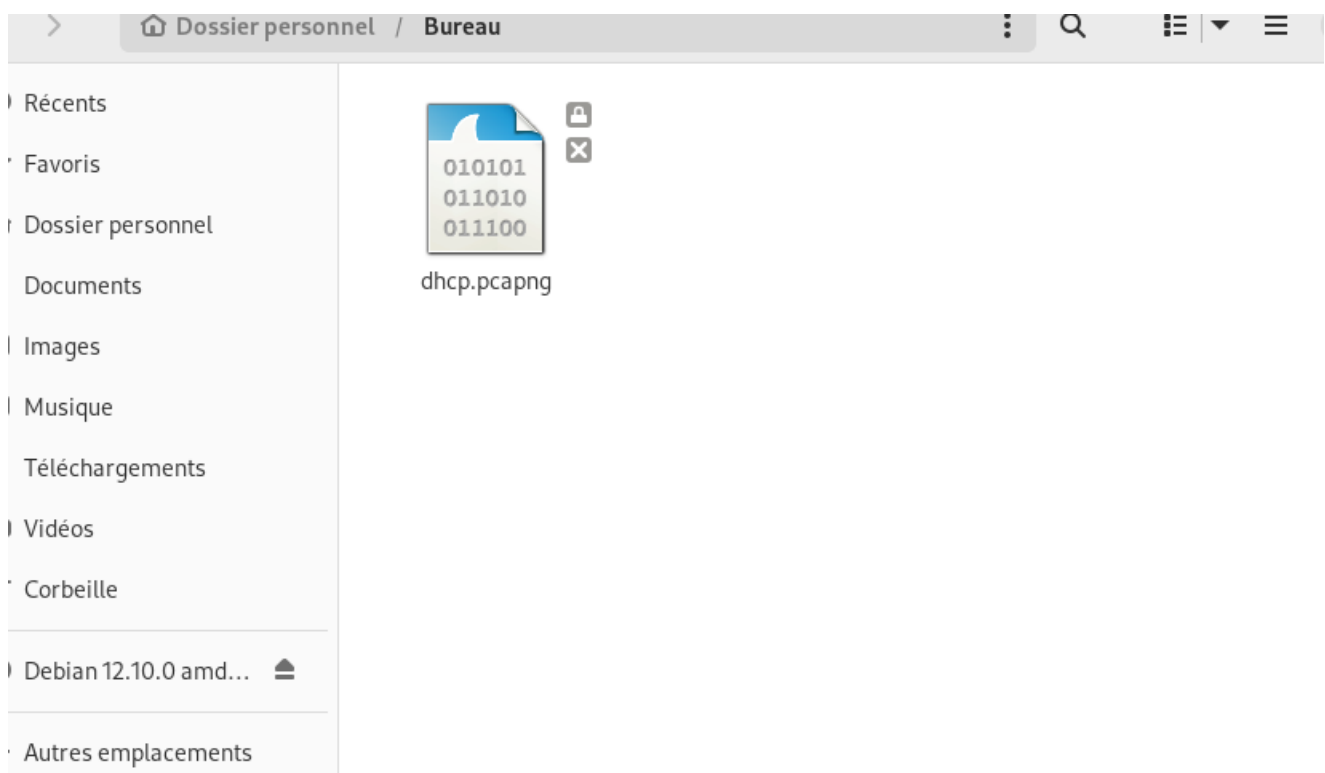
1. Protocole DHCP

1.1 Filtres Wireshark utilisés

dhcp

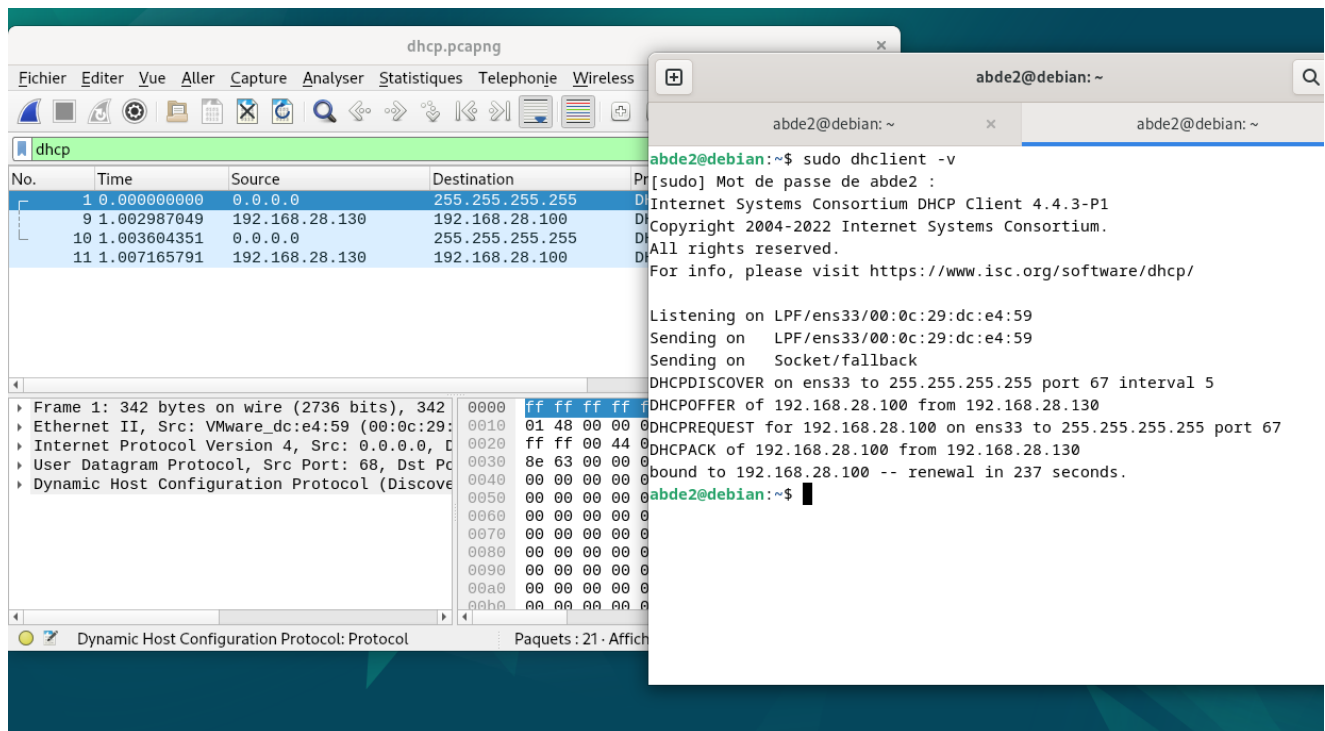
1.2 Configuration du serveur DHCP

La capture suivante montre le fichier de configuration DHCP `/etc/dhcp/dhcpd.conf` :



Cette configuration définit : - Un temps de bail par défaut de 600 secondes (10 minutes) - Un temps de bail maximum de 7200 secondes (2 heures) - Le mode autoritatif (le serveur est l'autorité DHCP principale sur ce réseau) - Un sous-réseau 192.168.28.0/24 - Une plage d'adresses de 192.168.28.100 à 192.168.28.120 - Le routeur par défaut à l'adresse 192.168.28.130 - Les serveurs DNS Google (8.8.8.8) et Cloudflare (1.1.1.1) - Le nom de domaine "local"

La capture suivante montre le fichier de configuration du service DHCP `/etc/default/isc-dhcp-server` :



Ce fichier indique que le service DHCP écoute sur l'interface réseau `enp0s3` .

1.3 Interprétation des paquets capturés

La capture suivante montre l'analyse des paquets DHCP dans Wireshark :

```
abde@debian: ~
GNU nano 7.2 /etc/default/isc-dhcp-server
# Defaults for isc-dhcp-server (sourced by /etc/init.d/isc-dhcp-server)

# Path to dhcpd's config file (default: /etc/dhcp/dhcpd.conf)
#DHCPDv4_CONF=/etc/dhcp/dhcpd.conf
#DHCPDv6_CONF=/etc/dhcp/dhcpd6.conf

# Path to dhcpd's PID file (default: /var/run/dhcpd.pid)
#DHCPDv4_PID=/var/run/dhcpd.pid
#DHCPDv6_PID=/var/run/dhcpd6.pid

# Additional options to start dhcpd with.
# Don't use options -cf or -pf here; use DHCPD_OPTS instead.
#OPTIONS=""

# On what interfaces should the DHCP server (dhcpd) serve clients?
# Separate multiple interfaces with spaces, e.g. "eth0 eth1"
INTERFACESv4="enp0s3"
INTERFACESv6=""

^G Aide      ^O Écrire    ^W Chercher  ^K Couper    ^_
^X Quitter   ^R Lire fich.^_ Remplacer  ^U Coller    ^_

```

Dans cette capture, on observe le processus DORA (Discover, Offer, Request, Acknowledge) complet :

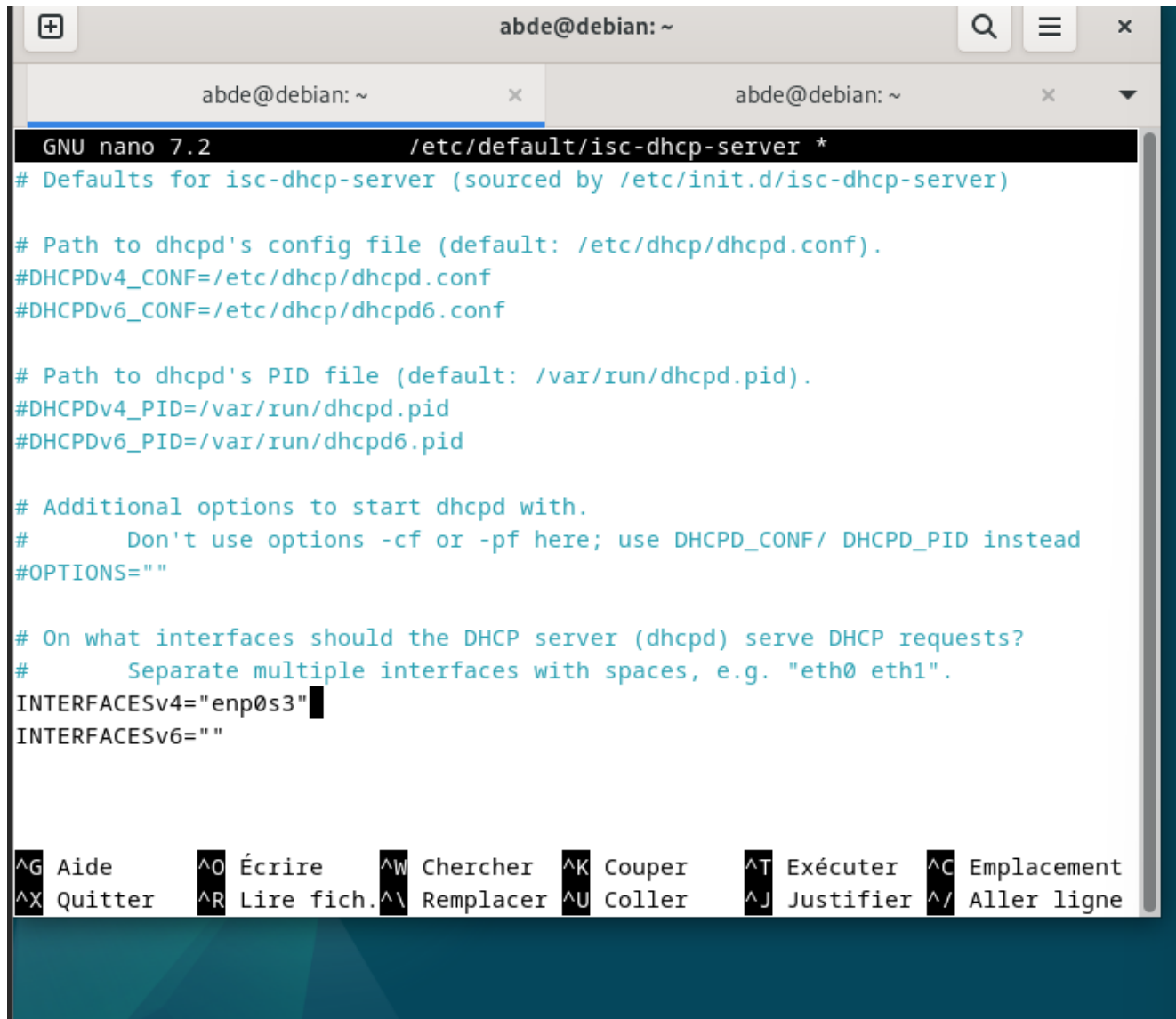
Trame 1 (DHCPDISCOVER) : - Source : 0.0.0.0 (client sans adresse IP) - Destination : 255.255.255.255 (broadcast) - Le client envoie un message de découverte pour trouver des serveurs DHCP - Contient l'adresse MAC du client pour identification

Trame 9 (DHCPOFFER) : - Source : 192.168.28.130 (serveur DHCP) - Destination : 192.168.28.100 (adresse proposée) - Le serveur propose l'adresse IP 192.168.28.100 au client

Trame 10 (DHCPREQUEST) : - Source : 0.0.0.0 (client) - Destination : 255.255.255.255 (broadcast) - Le client demande formellement l'adresse proposée - Le broadcast permet d'informer tous les serveurs DHCP potentiels

Trame 11 (DHCPACK) : - Source : 192.168.28.130 (serveur DHCP) - Destination : 192.168.28.100 (client) - Le serveur confirme l'attribution de l'adresse

La capture suivante montre l'exécution du client DHCP avec la commande `dhclient -v` :



```
GNU nano 7.2 /etc/default/isc-dhcp-server *
# Defaults for isc-dhcp-server (sourced by /etc/init.d/isc-dhcp-server)

# Path to dhcpd's config file (default: /etc/dhcp/dhcpd.conf).
#DHCPDv4_CONF=/etc/dhcp/dhcpd.conf
#DHCPDv6_CONF=/etc/dhcp/dhcpd6.conf

# Path to dhcpd's PID file (default: /var/run/dhcpd.pid).
#DHCPDv4_PID=/var/run/dhcpd.pid
#DHCPDv6_PID=/var/run/dhcpd6.pid

# Additional options to start dhcpd with.
# Don't use options -cf or -pf here; use DHCPD_CONF/ DHCPD_PID instead
#OPTIONS=""

# On what interfaces should the DHCP server (dhcpd) serve DHCP requests?
# Separate multiple interfaces with spaces, e.g. "eth0 eth1".
INTERFACESv4="enp0s3"
INTERFACESv6=""

^G Aide      ^O Écrire    ^W Chercher  ^K Couper    ^T Exécuter  ^C Emplacement
^X Quitter   ^R Lire fich.^\\ Remplacer  ^U Coller    ^J Justifier ^_ Aller ligne
```

Cette sortie confirme le processus DORA côté client :

```
Listening on LPF/ens33/00:0c:29:dc:e4:59
Sending on LPF/ens33/00:0c:29:dc:e4:59
Sending on Socket/fallback
DHCPDISCOVER on ens33 to 255.255.255.255 port 67 interval 5
DHCPOFFER of 192.168.28.100 from 192.168.28.130
DHCPREQUEST for 192.168.28.100 on ens33 to 255.255.255.255 port 67
DHCPACK of 192.168.28.100 from 192.168.28.130
bound to 192.168.28.100 -- renewal in 237 seconds.
```

1.4 Analyse selon les spécifications

Selon la RFC 2131, le protocole DHCP utilise le format de message suivant :

1. **En-tête :**
2. Op code : 1 (BOOTREQUEST) pour les messages client, 2 (BOOTREPLY) pour les messages serveur
3. Type de matériel : 1 pour Ethernet
4. Longueur d'adresse matérielle : 6 octets pour Ethernet
5. Hops : 0 pour les messages directs
6. ID de transaction : identique dans les 4 messages pour associer la séquence
7. Secondes écoulées : temps depuis le début de la requête
8. Flags : 0x8000 pour broadcast, 0x0000 pour unicast
9. Adresses : ciaddr (client), yiaddr (your IP), siaddr (server IP), giaddr (relay)
10. Adresse MAC client : chaddr
11. **Options DHCP :**
12. Option 53 : Type de message (1=DISCOVER, 2=OFFER, 3=REQUEST, 5=ACK)
13. Option 1 : Masque de sous-réseau (255.255.255.0)
14. Option 3 : Routeur (192.168.28.130)
15. Option 6 : Serveurs DNS (8.8.8.8, 1.1.1.1)
16. Option 15 : Nom de domaine ("local")
17. Option 51 : Durée du bail (600 secondes par défaut)
18. Option 54 : Identifiant du serveur DHCP

Dans la capture, on peut voir que le serveur DHCP est configuré en mode autoritatif (`authoritative;`), ce qui signifie qu'il est l'autorité principale pour ce réseau et peut envoyer des messages DHCPNAK si nécessaire.

2. Protocole DNS

2.1 Filtres Wireshark utilisés

```
dns
```

2.2 Configuration du serveur DNS

La capture suivante montre la configuration et le statut du service DNS BIND9 :

```
abde2@debian: ~
abde2@debian: ~
abde2@debian: ~
abde2@debian:~$ sudo systemctl start avahi-daemon
abde2@debian:~$ sudo systemctl status avahi-daemon
• avahi-daemon.service - Avahi mDNS/DNS-SD Stack
    Loaded: loaded (/lib/systemd/system/avahi-daemon.service; enabled; pri
    Active: active (running) since Tue 2025-05-20 08:55:51 CEST; 14min ago
TriggeredBy: • avahi-daemon.socket
    Main PID: 579 (avahi-daemon)
    Status: "avahi-daemon 0.8 starting up."
    Tasks: 2 (limit: 18619)
    Memory: 2.1M
    CPU: 234ms
    CGroup: /system.slice/avahi-daemon.service
            └─579 "avahi-daemon: running [debian.local]"
              └─661 "avahi-daemon: chroot helper"

nai 20 08:55:51 debian avahi-daemon[579]: Registering new address record f
nai 20 08:55:51 debian systemd[1]: Started avahi-daemon.service - Avahi m
nai 20 08:55:52 debian avahi-daemon[579]: Server startup complete. Host na
nai 20 08:55:53 debian avahi-daemon[579]: Joining mDNS multicast group on
nai 20 08:55:53 debian avahi-daemon[579]: New relevant interface ens33.IPv
nai 20 08:55:53 debian avahi-daemon[579]: Registering new address record f
nai 20 08:55:53 debian avahi-daemon[579]: Joining mDNS multicast group on
nai 20 08:55:53 debian avahi-daemon[579]: New relevant interface ens33.IPv
nai 20 08:55:53 debian avahi-daemon[579]: Registering new address record f
```

Cette capture montre l'installation et la configuration du service :

```
Ajout de l'utilisateur système « bind » (UID 115) ...
Ajout du nouvel utilisateur « bind » (UID 115) avec pour groupe
d'appartenance « bind » ...
Pas de création du répertoire personnel « /var/cache/bind ».
wrote key file "/etc/bind/rndc.key"
named-resolvconf.service is a disabled or a static unit, not
starting it.
Created symlink /etc/systemd/system/bind9.service → /lib/
systemd/system/named.service.
Created symlink /etc/systemd/system/multi-user.target.wants/
named.service → /lib/systemd/system/named.service.
```

La capture suivante montre le service DNS BIND9 en fonctionnement :

```
Activités Terminal 20 mai 09:07
abde@debian: ~
Ajout de l'utilisateur système « bind » (UID 115) ...
Ajout du nouvel utilisateur « bind » (UID 115) avec pour groupe d'appartenance « bind » ...
Pas de création du répertoire personnel « /var/cache/bind ».
wrote key file "/etc/bind/rndc.key"
named-resolvconf.service is a disabled or a static unit, not starting it.
Created symlink /etc/systemd/system/bind9.service → /lib/systemd/system/named.service.
Created symlink /etc/systemd/system/multi-user.target.wants/named.service → /lib/systemd/system/named.service.
Traitement des actions différées (« triggers ») pour man-db (2.11.2-2) ...
abde@debian:~$ systemctl status bind9
● named.service - BIND Domain Name Server
   Loaded: loaded (/lib/systemd/system/named.service; enabled; preset: enabled)
   Active: active (running) since Tue 2025-05-20 09:06:57 CEST; 13s ago
     Docs: man:named(8)
  Main PID: 4010 (named)
    Status: "running"
     Tasks: 6 (limit: 18619)
  Memory: 30.2M
     CPU: 180ms
    CGroup: /system.slice/named.service
            └─4010 /usr/sbin/named -f -u bind

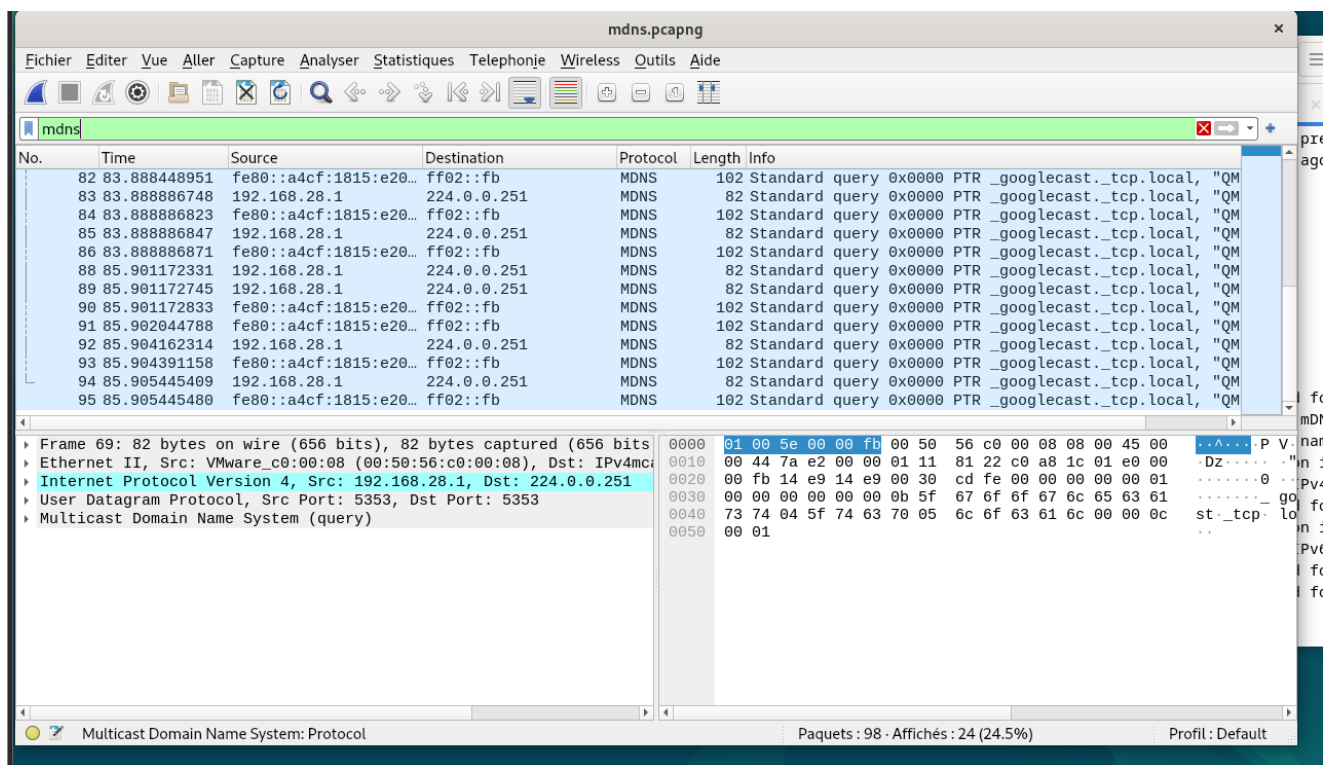
lines 1-11/11 (END)
```

Le statut du service montre qu'il est correctement démarré :

```
● named.service - BIND Domain Name Server
   Loaded: loaded (/lib/systemd/system/named.service;
enabled; preset: enabled)
   Active: active (running) since Tue 2025-05-20 09:06:57
CEST; 13s ago
     Docs: man:named(8)
  Main PID: 4010 (named)
    Status: "running"
     Tasks: 6 (limit: 18619)
  Memory: 30.2M
     CPU: 180ms
    CGroup: /system.slice/named.service
            └─4010 /usr/sbin/named -f -u bind
```

2.3 Interprétation des paquets capturés

La capture suivante montre l'analyse des paquets DNS dans Wireshark :



Dans cette capture, on observe plusieurs types d'échanges DNS :

Trames 106-107 : - Source : 192.168.28.130 - Destination : 192.112.36.4 (serveur DNS) - Requête : Standard query 0x40a0 A google.com - Réponse : Standard query response 0x40a0 Refused A google.com

Trames 108-109 : - Source : 192.168.28.130 - Destination : 199.7.83.42 (autre serveur DNS) - Requête : Standard query 0x7e4b A google.com - Réponse : Standard query response 0x7e4b Refused A google.com

Trames 110-111 : - Source : 192.168.28.130 - Destination : 192.36.148.17 - Requête : Standard query 0xf092 A google.com - Réponse : Standard query response 0xf092 Refused A google.com

Trames 116-117 : - Source : 192.168.28.130 - Destination : 198.97.190.53 - Requête : Standard query 0x5a53 A google.com - Réponse : Standard query response 0x5a53 Refused A google.com

2.4 Analyse selon les spécifications

Selon la RFC 1035, les messages DNS ont la structure suivante :

1. **En-tête (12 octets)** :
2. ID de transaction (16 bits) : identifie la requête/réponse (ex: 0x40a0)
3. Flags (16 bits) :
 - QR (1 bit) : 0 pour requête, 1 pour réponse
 - Opcode (4 bits) : 0 pour requête standard

- AA (1 bit) : Authoritative Answer
- TC (1 bit) : Truncated
- RD (1 bit) : Recursion Desired (1 dans les requêtes observées)
- RA (1 bit) : Recursion Available
- Z (3 bits) : Réservé
- RCODE (4 bits) : 0=pas d'erreur, 5=refusé (dans les réponses observées)

4. Nombre de questions, réponses, autorités, additionnelles

5. **Section Question :**

6. QNAME : Nom de domaine encodé (google.com)

7. QTYPE : Type de requête (1=A pour adresse IPv4)

8. QCLASS : Classe (1=IN pour Internet)

9. **Section Réponse** (si présente) :

10. NAME : Nom de domaine

11. TYPE : Type d'enregistrement

12. CLASS : Classe

13. TTL : Time To Live

14. RDLENGTH : Longueur des données

15. RDATA : Données (adresse IP, etc.)

Dans les captures, toutes les réponses ont le code "Refused" (RCODE=5), ce qui indique que les serveurs DNS contactés refusent de répondre aux requêtes, probablement en raison de restrictions d'accès ou de configuration. Le client essaie donc plusieurs serveurs DNS différents sans succès.

La partie hexadécimale de la trame montre :

```
00 0c 29 f5 e1 2b 00 0c 29 dc e4 59 08 00 45 00
00 4f 9b 1f 00 00 40 11 25 29 c0 a8 1c 82 c0 70
24 04 56 49 00 35 00 3b ba a2 40 a0 01 00 00 01
00 00 00 00 00 00 06 67 6f 6f 67 6c 65 03 63 6f
6d 00 00 01 00 01
```

On y distingue : - L'ID de transaction DNS (40 a0) - Les flags de requête (01 00) - Une question (00 01) - Le nom de domaine encodé "google.com" (06 67 6f 6f 67 6c 65 03 63 6f 6d 00) - Le type A (00 01) et la classe IN (00 01)

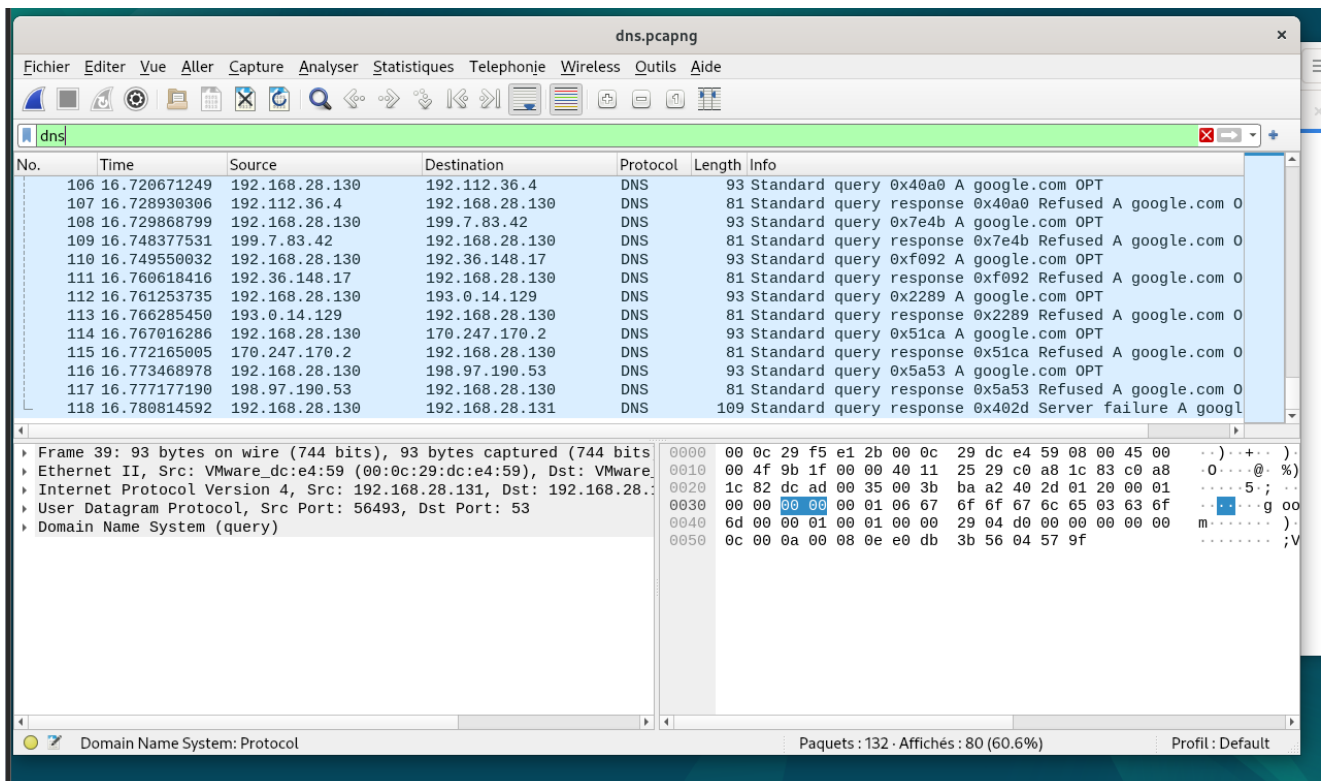
3. Protocole mDNS

3.1 Filtres Wireshark utilisés

mdns

3.2 Configuration du service mDNS

La capture suivante montre la configuration et le démarrage du service mDNS Avahi :



Cette capture montre le démarrage et la configuration du service :

```
$ sudo systemctl start avahi-daemon
$ sudo systemctl status avahi-daemon
● avahi-daemon.service - Avahi mDNS/DNS-SD Stack
   Loaded: loaded (/lib/systemd/system/avahi-daemon.service;
   enabled; preset: enabled)
   Active: active (running) since Tue 2025-05-20 08:55:51
   CEST; 14min ago
   Main PID: 579 (avahi-daemon)
   Status: "avahi-daemon 0.8 starting up."
   Tasks: 2 (limit: 18619)
   Memory: 2.1M
   CPU: 234ms
   CGroup: /system.slice/avahi-daemon.service
```

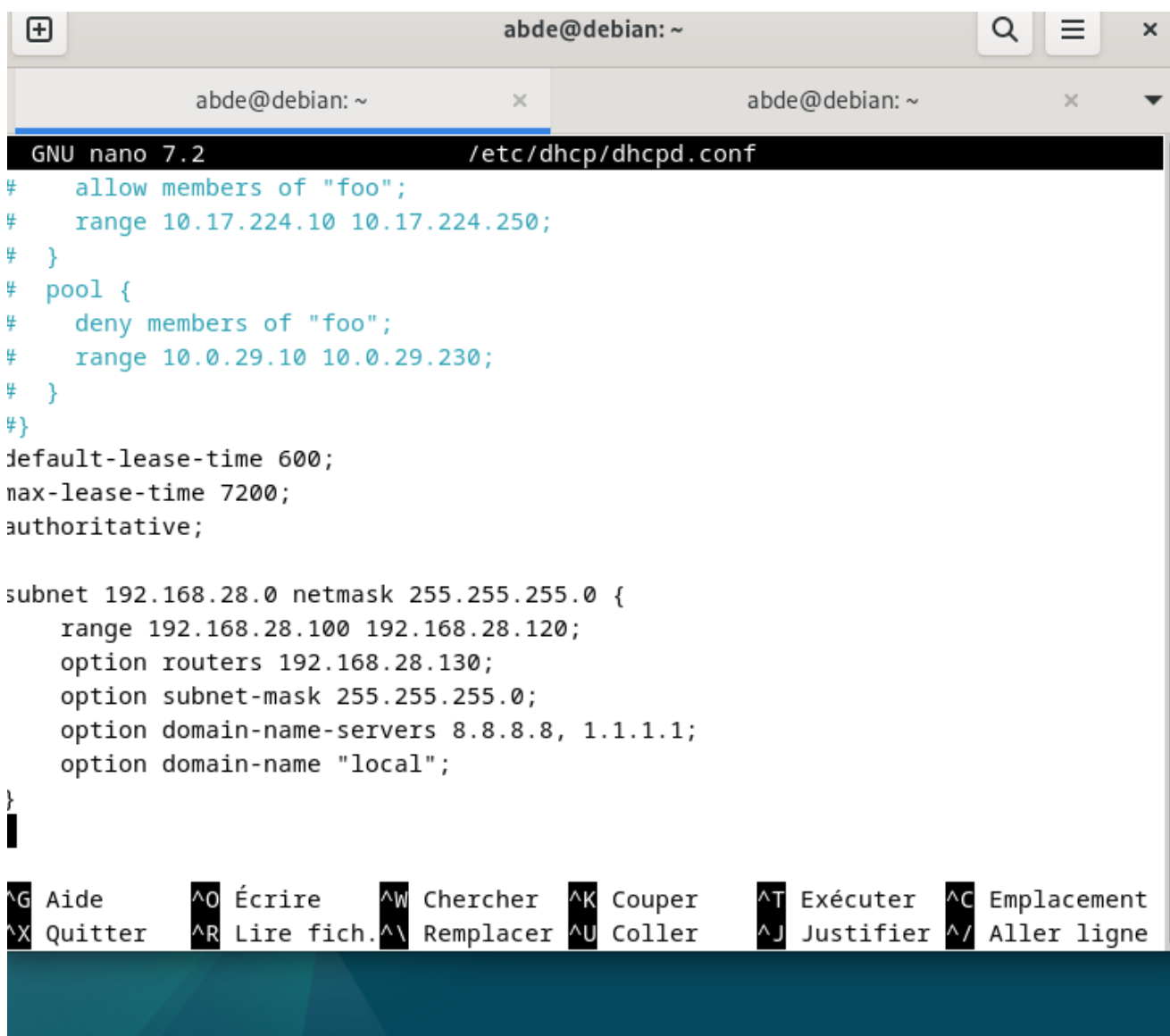
```
└─579 "avahi-daemon: running [debian.local]"
└─661 "avahi-daemon: chroot helper"
```

Les logs du service montrent son initialisation :

```
mai 20 08:55:51 debian avahi-daemon[579]: Registering new
address record for...
mai 20 08:55:51 debian systemd[1]: Started avahi-daemon.service
- Avahi mDNS/DNS-SD Stack.
mai 20 08:55:52 debian avahi-daemon[579]: Server startup
complete. Host name is debian.local.
mai 20 08:55:53 debian avahi-daemon[579]: New relevant
interface ens33.IPv4...
mai 20 08:55:53 debian avahi-daemon[579]: Registering new
address record for...
mai 20 08:55:53 debian avahi-daemon[579]: Joining mDNS
multicast group on...
mai 20 08:55:53 debian avahi-daemon[579]: New relevant
interface ens33.IPv6...
mai 20 08:55:53 debian avahi-daemon[579]: Registering new
address record for...
```

3.3 Interprétation des paquets capturés

La capture suivante montre l'analyse des paquets mDNS dans Wireshark :



```
GNU nano 7.2 /etc/dhcp/dhcpd.conf
#   allow members of "foo";
#   range 10.17.224.10 10.17.224.250;
# }
# pool {
#   deny members of "foo";
#   range 10.0.29.10 10.0.29.230;
# }
#}
default-lease-time 600;
max-lease-time 7200;
authoritative;

subnet 192.168.28.0 netmask 255.255.255.0 {
    range 192.168.28.100 192.168.28.120;
    option routers 192.168.28.130;
    option subnet-mask 255.255.255.0;
    option domain-name-servers 8.8.8.8, 1.1.1.1;
    option domain-name "local";
}

```

^G Aide ^O Écrire ^W Chercher ^K Couper ^T Exécuter ^C Emplacement
^X Quitter ^R Lire fich. ^\ Remplacer ^U Coller ^J Justifier ^_ Aller ligne

Dans cette capture, on observe plusieurs requêtes pour le service
"_googlecast._tcp.local" :

Trames 82, 84, 86, etc. : - Source : fe80::a4cf:1815:e20... (adresse IPv6 locale) -
Destination : ff02::fb (adresse multicast IPv6 pour mDNS) - Requête : Standard query
0x0000 PTR _googlecast._tcp.local

Trames 83, 85, 88, etc. : - Source : 192.168.28.1 (adresse IPv4) - Destination : 224.0.0.251
(adresse multicast IPv4 pour mDNS) - Requête : Standard query 0x0000 PTR
_googlecast._tcp.local

3.4 Analyse selon les spécifications

Selon la RFC 6762 (Multicast DNS), mDNS est une extension du protocole DNS qui permet
la résolution de noms sur des réseaux locaux sans serveur DNS centralisé.

1. Différences clés avec DNS standard :

2. Utilise l'adresse multicast 224.0.0.251 (IPv4) ou ff02::fb (IPv6)

3. Utilise le port UDP 5353 (au lieu de 53)
4. Les noms se terminent généralement par ".local"
5. Les requêtes peuvent être envoyées en multicast
6. **Structure des paquets :**
7. Similaire à DNS standard, mais avec des flags spécifiques
8. ID de transaction souvent à 0x0000 pour les requêtes multicast
9. Questions de type PTR pour la découverte de services
10. **Découverte de services (DNS-SD) :**
11. Format de nom : `_service._protocole.domaine`
12. Dans la capture : `_googlecast._tcp.local` (service Chromecast)

La partie hexadécimale de la trame montre :

```
01 00 5c 00 00 fb 00 50 56 c0 00 08 08 00 45 00
00 44 7a e2 00 00 01 11 81 22 c0 a8 1c 01 e0 00
00 fb 14 e9 14 e9 00 30 cd fe 00 00 00 00 00 01
00 00 00 00 00 00 0b 5f 67 6f 6f 67 6c 65 63 61
73 74 04 5f 74 63 70 05 6c 6f 63 61 6c 00 00 0c
00 01
```

On y distingue : - L'adresse multicast de destination (e0 00 00 fb = 224.0.0.251) - Le port UDP 5353 (14 e9) - L'ID de transaction à 0 (00 00) - Une question (00 01) - Le nom "_googlecast._tcp.local" encodé - Le type PTR (00 0c) et la classe IN (00 01)

4. Messages TCP/TLS

4.1 Filtres Wireshark utilisés

```
tcp.port == 443
```

4.2 Interprétation des paquets capturés

La capture suivante montre l'analyse de paquets TCP avec le filtre "tcp.port == 443" :

Activités Wireshark 20 mai 10:22 *ens33

Fichier Editer Vue Aller Capture Analyser Statistiques Telephonie Wireless Outils Aide

tcp.port == 443

No.	Time	Source	Destination	Protocol	Length	Info
256	140.650816847	192.168.28.131	192.168.28.130	TCP	74	39492 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=726171297 TSecr=0 WS=128
257	140.652027177	192.168.28.130	192.168.28.131	TCP	74	443 → 39492 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=2443455499 TSecr=726171297 WS=128
258	140.652118790	192.168.28.131	192.168.28.130	TCP	66	39492 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=726171298 TSecr=2443455499
259	140.665142100	192.168.28.131	192.168.28.130	TLSv1.3	583	Client Hello
260	140.666350974	192.168.28.130	192.168.28.131	TCP	66	443 → 39492 [ACK] Seq=1 Ack=518 Win=64768 Len=0 TSval=2443455513 TSecr=726171311
261	140.671877021	192.168.28.130	192.168.28.131	TLSv1.3	1392	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data
262	140.671917696	192.168.28.131	192.168.28.130	TCP	66	39492 → 443 [ACK] Seq=518 Ack=1327 Win=64128 Len=0 TSval=726171318 TSecr=2443455519
263	140.727227432	192.168.28.131	192.168.28.130	TLSv1.3	73	Alert (Level: Fatal, Description: Unknown CA)
264	140.728200020	192.168.28.130	192.168.28.131	TCP	66	39492 → 443 [RST, ACK] Seq=525 Ack=1327 Win=64128 Len=0 TSval=726171375 TSecr=2443455519
265	140.728984511	192.168.28.131	192.168.28.130	TCP	66	443 → 39492 [FIN, ACK] Seq=1327 Ack=525 Win=64768 Len=0 TSval=2443455570 TSecr=726171373
266	140.729886869	192.168.28.131	192.168.28.130	TCP	54	39492 → 443 [RST] Seq=525 Win=0 Len=0

Frame 264: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface ens33, 1 Ethernet II, Src: VMware_dc:e4:59 (00:0c:29:dc:e4:59), Dst: VMware_f5:e1:2b (00:0c:29:f5:e1:2b), Internet Protocol Version 4, Src: 192.168.28.131, Dst: 192.168.28.130, Transmission Control Protocol, Src Port: 39492, Dst Port: 443, Seq: 525, Ack: 1327, Len: 0

```

0000  00 0c 29 f5 e1 2b 00 0c 29 dc e4 59 08 00 45 00  ..).+... )...Y..E.
0010  00 34 c5 6a 40 00 40 06 bb 03 c0 a8 1c 83 c0 a8  .4.j@.@.....-4.j@.@.....
0020  1c 82 9a 44 01 bb e0 b2 f9 14 62 14 e8 3c 00 14  .D.....-b.<...
0030  01 f5 ba 7c 00 00 01 01 08 0a 2b 48 76 ef 81 a4  .|.....+H....
0040  2c 1f                                     ,.

```

Dans cette capture, on observe une séquence complète d'établissement de connexion TCP, suivie d'une négociation TLS et d'échanges de données :

Établissement de connexion TCP (Three-way handshake) : - Trame 256 : SYN de 192.168.28.131 vers 192.168.28.130, port 443 - Flags : SYN - Seq=0, Win=64240, Options: MSS=1460, SACK_PERM, TSval=726171297 - Trame 257 : SYN+ACK de 192.168.28.130 vers 192.168.28.131 - Flags : SYN, ACK - Seq=0, Ack=1, Win=65160, Options: MSS=1460, SACK_PERM, TSval=2443455499 - Trame 258 : ACK de 192.168.28.131 vers 192.168.28.130 - Flags : ACK - Seq=1, Ack=1, Win=64256

Négociation TLS : - Trame 259 : Client Hello (TLSv1.3) de 192.168.28.131 vers 192.168.28.130 - Contient les suites cryptographiques supportées et les extensions - Trame 261 : Server Hello, Change Cipher Spec, Application Data (TLSv1.3) - Sélection de la suite cryptographique et établissement des paramètres de sécurité

Échange de données chiffrées : - Trames 260, 262, 263 : Échanges de données applicatives chiffrées

Terminaison de connexion : - Trame 264 : RST+ACK de 192.168.28.130 vers 192.168.28.131 - Flags : RST, ACK - Seq=525, Ack=1327, Win=64128 - Trame 265 : FIN+ACK de 192.168.28.130 vers 192.168.28.131 - Flags : FIN, ACK - Seq=1327, Ack=525, Win=64768 - Trame 266 : RST de 192.168.28.131 vers 192.168.28.130 - Flags : RST - Seq=525, Win=0

4.3 Analyse selon les spécifications

Protocole TCP (RFC 793) :

1. **Établissement de connexion (Three-way handshake) :**
2. SYN : Synchronisation des numéros de séquence

3. SYN+ACK : Accusé de réception et synchronisation

4. ACK : Confirmation finale

5. Structure de l'en-tête TCP :

6. Ports source et destination (16 bits chacun)

7. Numéro de séquence (32 bits)

8. Numéro d'acquittement (32 bits)

9. Offset de données (4 bits)

10. Réserve (3 bits)

11. Flags (9 bits) : NS, CWR, ECE, URG, ACK, PSH, RST, SYN, FIN

12. Taille de fenêtre (16 bits)

13. Checksum (16 bits)

14. Pointeur urgent (16 bits)

15. Options (variable)

16. Terminaison de connexion :

17. Normalement par échange FIN/ACK dans les deux sens

18. Ici, terminaison abrupte avec RST (reset)

Protocole TLS (RFC 8446 pour TLSv1.3) :

1. Handshake TLS :

2. Client Hello : Propose versions, suites cryptographiques, extensions

3. Server Hello : Sélectionne version, suite cryptographique, extensions

4. Key Exchange : Échange de clés pour établir une clé de session

5. Finished : Vérification de l'intégrité du handshake

6. Record Protocol :

7. Encapsule les données applicatives

8. Assure confidentialité (chiffrement) et intégrité (MAC)

Dans la trame 264, on peut voir l'en-tête TCP avec les flags RST et ACK activés :

```
TCP 66 39492 → 443 [RST, ACK] Seq=525 Ack=1327 Win=64128 Len=0
TSval=2261713187 TSecr=2443455519
```

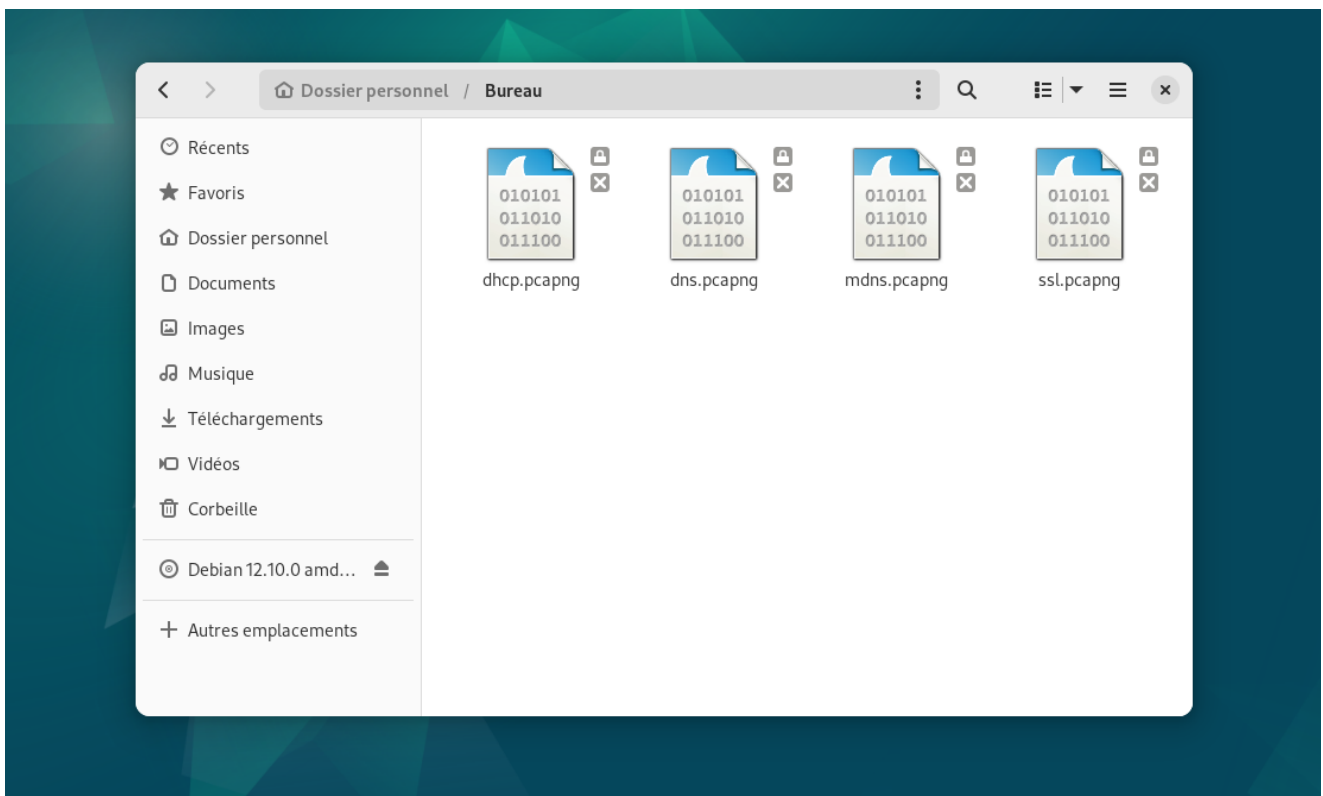
La partie hexadécimale montre :


```
00 0c 29 f5 e1 2b 00 0c 29 dc e4 59 08 00 45 00
00 34 c5 6a 40 40 06 bb 03 c0 a8 1c 01 e0 00 00
1c 82 9a 44 01 bb 60 b2 f9 14 c0 14 e8 3c 80 14
01 f5 ba 7c 00 00 01 01 08 0a 2b 48 7e ef 91 a4
2c 1f
```

On y distingue : - Les ports source (9a 44 = 39492) et destination (01 bb = 443) - Les flags TCP (14 = ACK + RST) - Le numéro de séquence et d'acquittement - La taille de fenêtre (01 f5 = 64128)

5. Organisation des fichiers de capture

La capture suivante montre l'interface de gestion de fichiers avec les différentes captures Wireshark sauvegardées :



Cette capture montre les fichiers de capture Wireshark organisés par protocole : -

dhcp.pcapng : Capture des paquets DHCP - dns.pcapng : Capture des paquets DNS -

mdns.pcapng : Capture des paquets mDNS - ssl.pcapng : Capture des paquets SSL/TLS

Ces fichiers représentent les captures filtrées pour chaque protocole, conformément aux exigences du projet.

6. Sécurité des échanges

6.1 Analyse de la sécurité des échanges

Bien que les captures ne montrent pas d'échanges FTP sans TLS, nous pouvons analyser la sécurité des protocoles observés :

1. **DHCP :**
2. Aucun chiffrement des données
3. Les informations de configuration réseau sont transmises en clair
4. Vulnérable aux attaques d'écoute passive et de DHCP spoofing
5. **DNS :**
6. Requêtes et réponses en clair
7. Les noms de domaine consultés sont visibles
8. Vulnérable aux attaques d'empoisonnement de cache DNS
9. **mDNS :**
10. Similaire au DNS, informations en clair
11. Limité au réseau local, ce qui réduit l'exposition
12. **TLS :**
13. Données applicatives chiffrées
14. Handshake sécurisé avec échange de clés
15. Protection contre l'écoute passive et l'interception active

6.2 Comparaison FTP vs SSL/TLS

Bien que non présent dans les captures, voici une comparaison :

FTP sans TLS : - Transmet les identifiants (nom d'utilisateur, mot de passe) en clair - Commandes et données visibles pour quiconque peut capturer le trafic - Un attaquant peut facilement récupérer les données sensibles de connexion

Échanges SSL/TLS (comme observé dans les captures) : - Données chiffrées après le handshake - Identifiants et contenu protégés contre l'écoute passive - Vérification d'intégrité pour prévenir la manipulation des données - Un attaquant ne peut pas récupérer les données sensibles sans compromettre les clés cryptographiques

Conclusion

L'analyse des captures Wireshark pour les protocoles DHCP, DNS, mDNS et TCP/TLS montre clairement les différences en termes de structure et de sécurité entre ces protocoles. Les protocoles non chiffrés comme DHCP, DNS et mDNS exposent leurs données en clair, tandis que TLS offre une protection des données par chiffrement.

Cette analyse souligne l'importance d'utiliser des protocoles sécurisés comme TLS pour protéger les données sensibles, particulièrement les identifiants de connexion qui seraient exposés dans des protocoles non sécurisés comme FTP sans TLS.

Les captures d'écran fournies ont permis d'illustrer non seulement les échanges de paquets, mais aussi la configuration des services correspondants, offrant ainsi une vision complète de l'infrastructure réseau mise en place pour ce projet.