Une image contenant flou, ciel nocturne

Description générée automatiquement

Networks Architectures project

Server project (DNS / DHCP trace)

Becker lucas, Ngo Thomas, Bue Mathieu, cordier Benoît

IOS 1 – Promo 2022

## description

Les fonctionnalités demandées pour le serveur sont :

Server under (Windows / Linux):

1) Connection to your DNS / DHCP log server (TCP)

2) Possibility to log in several @MACs managed by SQLite

3) Retrieval of DNS and DHCP requests in an SQLite database by @MAC

4) Possibility to display the logs by @MAC or by the OUI file (Manufacturer)

5) Possibility of sorting logs by (@IP, Date, Time ...)

6) Managing a list of unauthorized DNS

7) Notification if a DNS is in the list of unauthorized DNS

8) Notification if a DHCP @IP is in the list of unauthorized @MACs

9) Save logs to one file per day

10) Your original feature

Bonus:

1) Ability to read and send an alert email if for example DDOS attempt

## Fonctionnement

L’idée est de sniffer les paquets du réseau avec la librairie Python : Scapy.

Cette librairie nous permet de filtrer les paquets entrants pour ne récupérer que les paquets DNS et les paquets DHCP.

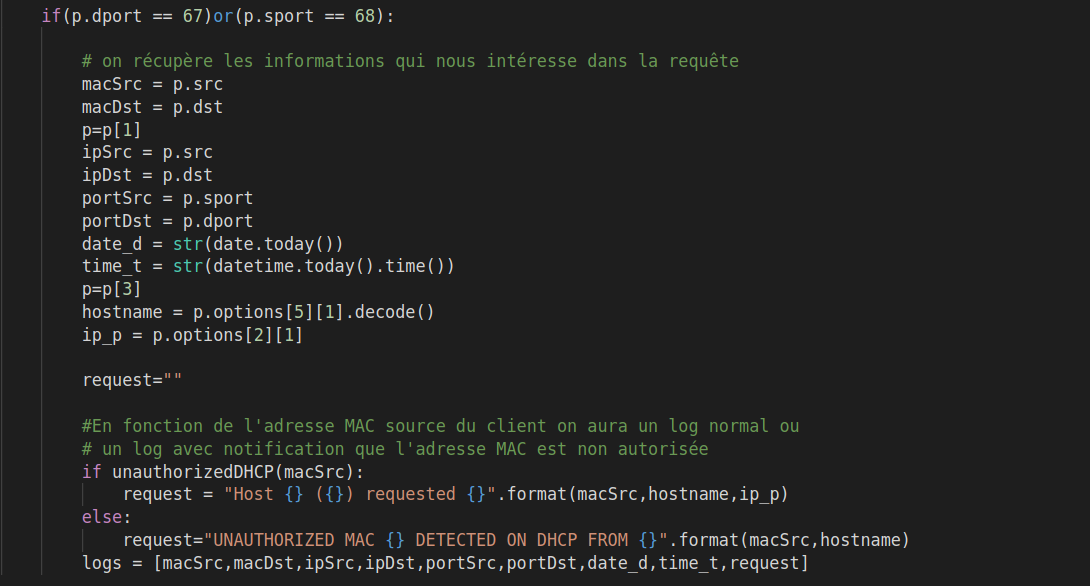


Grâce à l’argument prn de la fonction sniff(), on peut appliquer une fonction à chacun des paquets entrant. C’est dans cette fonction principale mainSniff() que se déroule le traitement des paquets pour l’utilisation du serveur de logs.

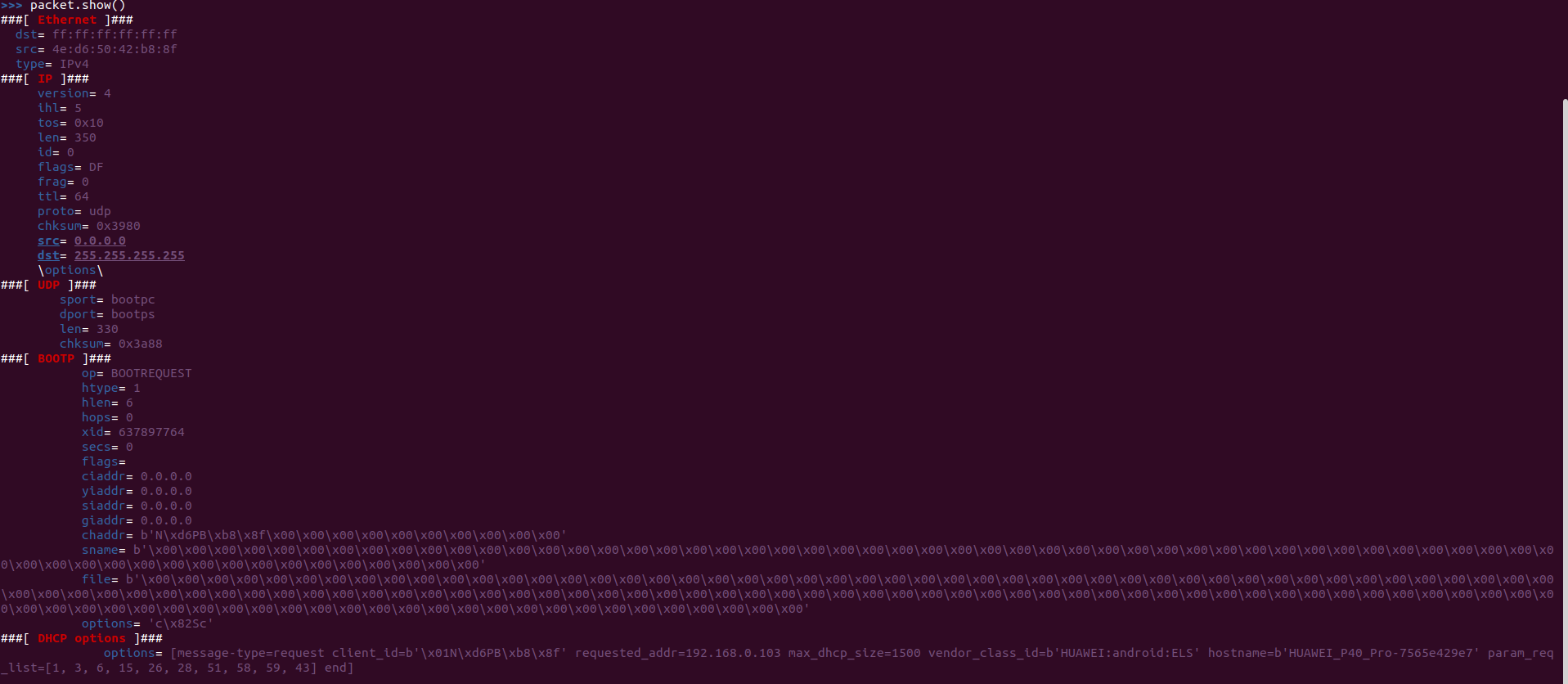
## Traitement des paquets dhcp

Comme nous le savons, le DHCP est un protocole qui permet à un ordinateur qui se connecte sur un réseau local d’obtenir dynamiquement et automatiquement sa configuration IP. On pourrait croire qu’un seul aller-retour peut suffire à la bonne marche du protocole mais il existe plusieurs messages DHCP qui permettent de compléter une configuration, la renouveler… Ces messages sont susceptibles d’être émis soit par le client pour le ou les serveurs, soit par le serveur vers un client : DHCP Discovery, DHCP Offer, DHCP Request, DHCP Ack etc.

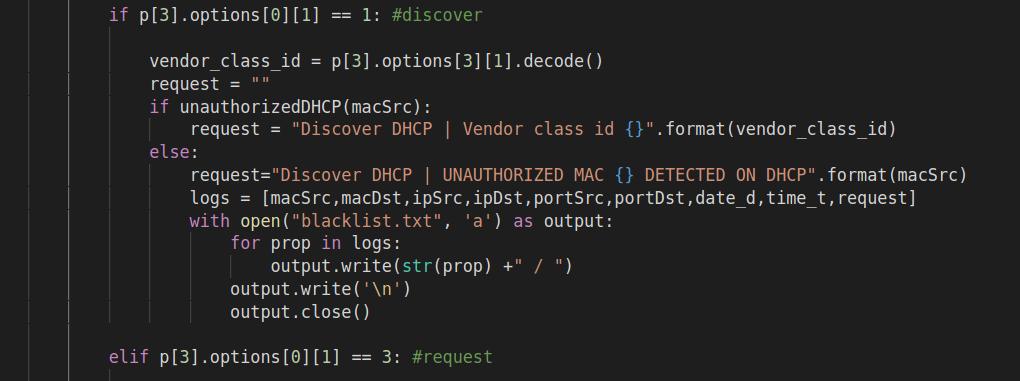
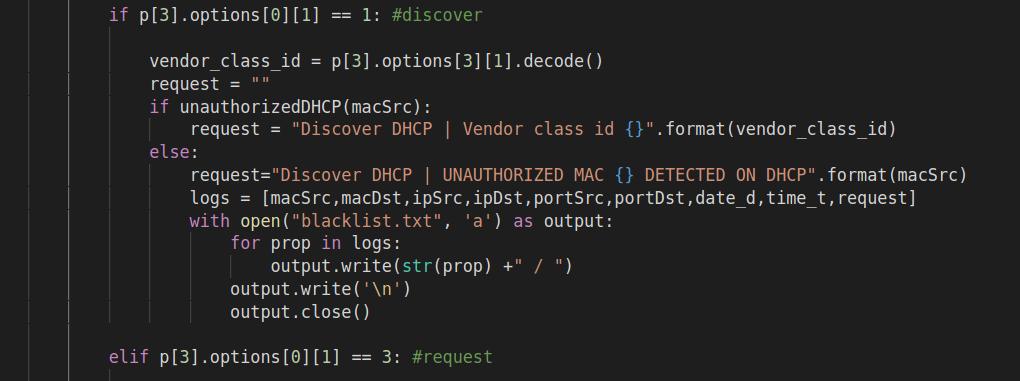
Ici nous nous intéresserons aux requêtes DHCP Discovery et DHCP Request, l’une permet à un client de trouver le serveur puis lui demander une réponse et l'autre de demander au serveur de lui attribuer une IP choisie parmi celle proposées.

Pour différencier les paquets DHCP des paquets DNS capturés par Scapy, on vérifiera les ports source et destination du paquet capturé. Un paquet DHCP utilise le port 67 et 68, 68 sur la source et 67 sur le destinataire. On a aura juste une condition if à rajouter dans le code :  


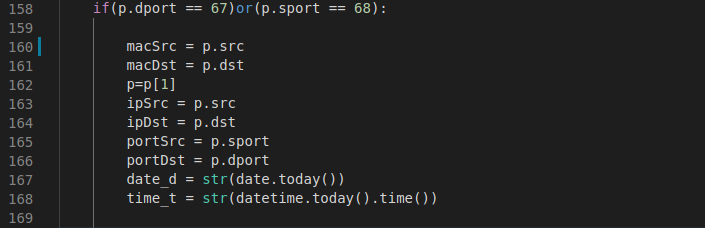
Ensuite pour différencier les paquets les DHCP Discovery et Request des autres paquets DHCP il faudra s’intéresser à la structure d’un paquet DHCP. Globalement la structure des deux paquets qui nous intéresse est presque la même. Comme nous le savons, un paquet est structuré sous forme de couche. On a dans un paquet DHCP 5 couches (Ether, IP, UDP, BOOTP et DHCP options) comme montré sur la photo ci-dessous :



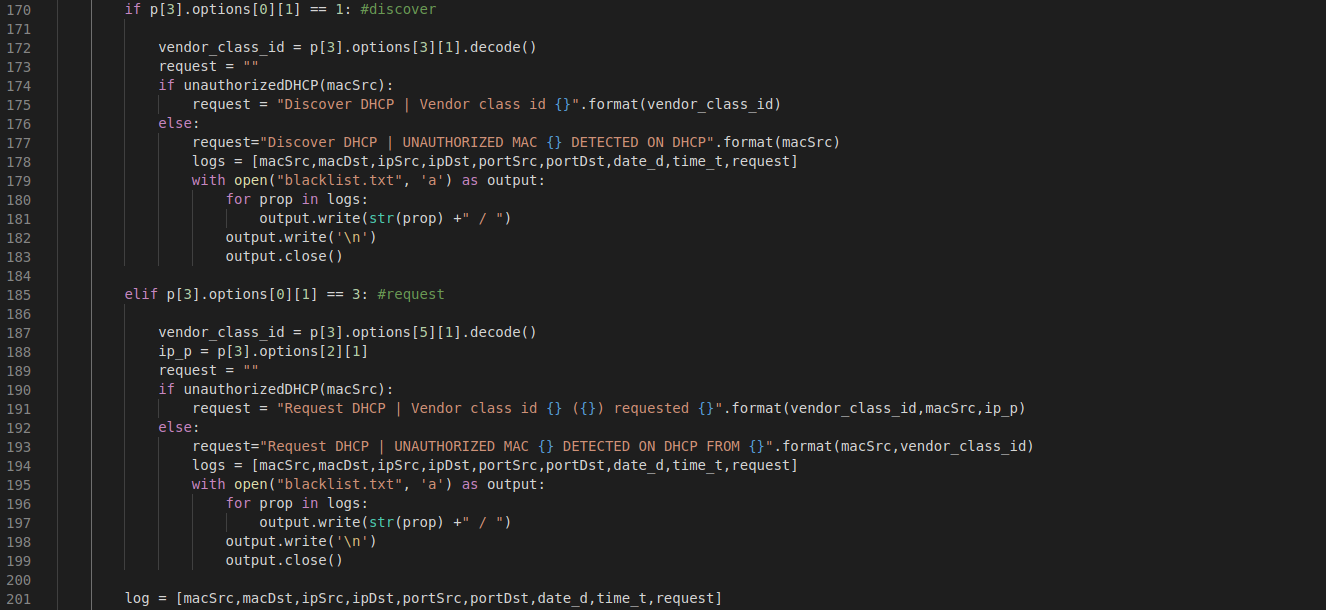
L’information qui permet de différencier les deux paquets se trouve dans la liste **options** dans la couche 5 **[DHCP options]**, c’est le “**message-type = request**”. S’il est égal à 1 le paquet est un Discovery, s’il est égal à 3 c’est un Request. Donc encore une fois on rajoutera une condition **if** pour traiter les informations qui diffèrent entre les deux paquets.



Une fois qu’on connait la structure d’un paquet DHCP il est facile d’en récupérer les informations qui nous intéresse. Les adresses MAC, IP et les ports source et destination des paquets sont au même endroit.



Mais c’est pour le traitement de l’enregistrement du type de la requête que va nous servir les conditions du dessus.



Pour un paquet DHCP Discovery on aura un type de requête différent de celui d’un paquet DHCP Request. Dans les deux cas on rajoutera une condition pour notifier le client lorsque l’on détectera une adresse MAC source dans un paquet qui correspond à une adresse MAC non autorisée de notre base de données.

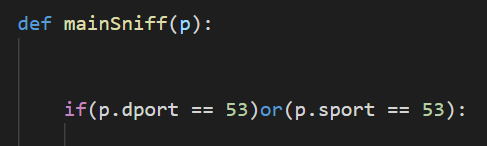
Une fois toutes les informations nécessaires récupérées on les enregistre dans la liste **log**, et on insérera cette liste dans la base de données grâce à la fonction **insertLog()**.

## Traitement des paquets DNS

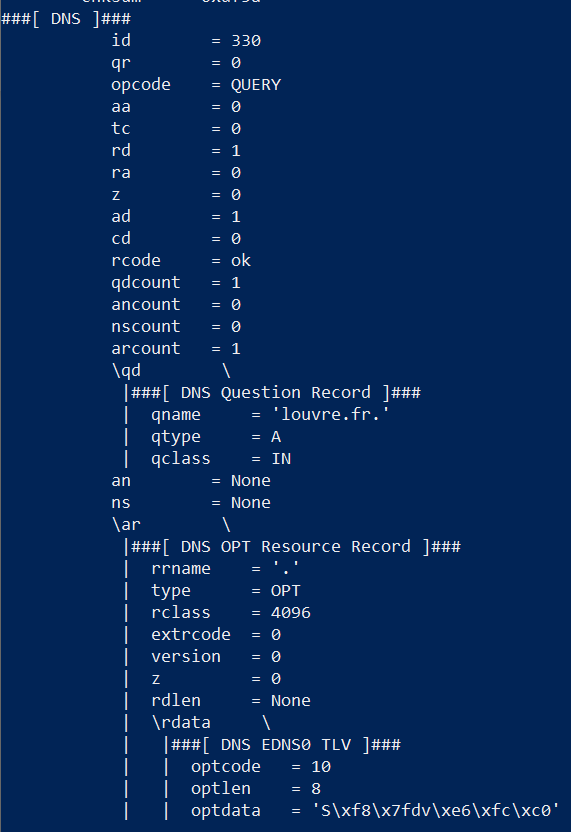
Les paquets DNS permettent de traduire les noms de domaine Internet en adresse IP. Les paquets DNS peuvent jouer différents rôles : Standard Query, Inverse Query, Server Status Request…

Mais la plupart du temps un paquet DNS va demander l’adresse IP d’un nom de domaine, et le serveur DNS lui renvoie l’adresse IP.

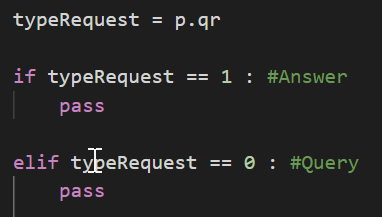
Les paquets DNS communiquent via le port 53 du serveur DNS. Donc lors d’une Query le dport = 53 et pour une réponse du serveur sport = 53.



On peut retrouver tous les champs d’un paquet au format DNS avec Scapy : (Voir partie 10 pour une étude approfondie des paquets DNS)

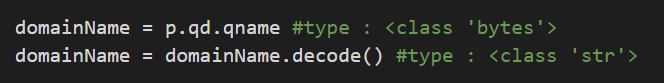


Il faut ensuite distinguer les paquets Query des paquets Answer :

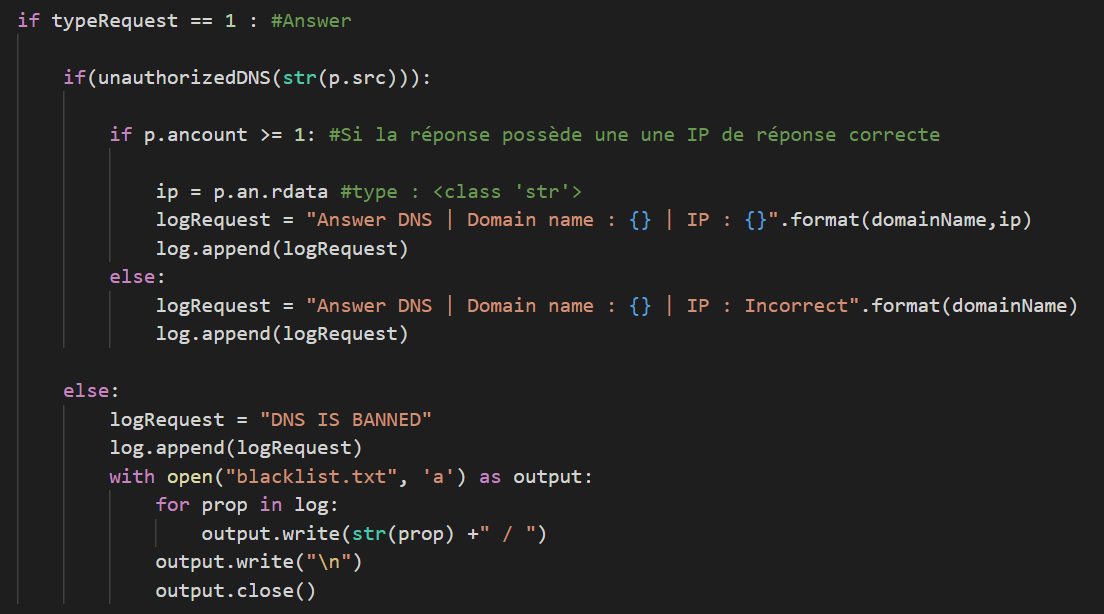


L’information se trouve dans le champ qr du paquet.

On récupère le nom de domaine qui est présent dans les Query ainsi que dans les Anwer :



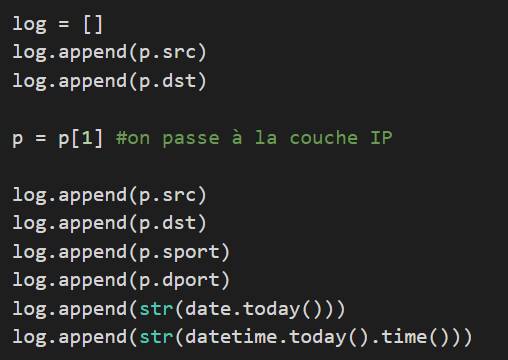
A partir de là on traite les deux paquets différemment. Premièrement on vérifie que le serveur DNS ne fait pas partie de la liste des DNS non autorisés. Ensuite, pour un paquet Answer, il faut également vérifier que le serveur possède bien une IP pour le nom de domaine recherché, sinon il renvoie None et le serveur peut crasher. On prend donc cette possibilité en considération et on vérifie que le numéro de réponse(s) est au moins supérieur à 1. Sinon on indique que l’IP renvoyée est incorrect.



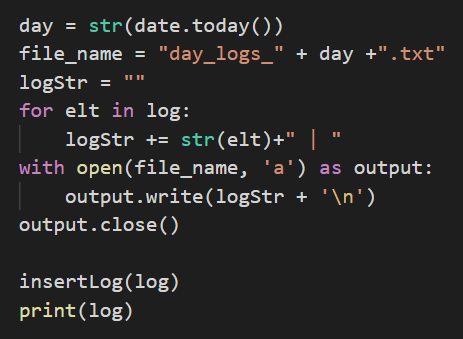
Pour une Query, le traitement est plus simple puisque à part la TransactionID et le nom de domaine, la partie DNS ne change pas vraiment entre les paquets :



Il faut ensuite regrouper toutes les informations nécessaires pour les logs (adresses MAC, adresses IP, la date…) et ajouter le champ Request de la table de données qui change selon les types de paquets DNS.



On insert ce log dans le fichier de log du jour puis dans la base de données. On termine en affichant le log pour avoir une trace de l’activité côté serveur.



## Traitement des logs

**2/ Possibility to log in several @MACs managed by SQLite**

Une image contenant table

Description générée automatiquementComme précisé dans le point précédent, lors de la récupération d’un paquet DNS ou DHCP quel qu’il soit, l’adresse Mac de la source et de la destination de la requête sont récupérés. En effet, ces adresses sont accessibles dans la trame Ethernet des paquets. :

Une image contenant texte

Description générée automatiquement

Ainsi, ces adresses Macs sont insérées dans la base de données grâce à la fonction insert Log (en fonction de si elle est autorisée ou non, détaillé en point 8).

**3/ Retrieval of DNS and DHCP requests in an SQLite database by @MAC**

Une image contenant texte

Description générée automatiquementAfin de récupérer les différentes requêtes enregistrées dans notre base de données, nous avons créé un second programme Python : ClientDatabse.py. Ce fichier va permettre à l’utilisateur de lire les différents logs selon des paramètres et des filtres de son choix :

Une image contenant texte

Description générée automatiquementConcernant les requêtes DHCP et DNS, il est nécessaire dans ce fichier de les récupérer avec toutes leurs informations précisées dans le point 1 du rapport. Pour cela, on se connecte dans un premier temps à la base de données dans une fonction prenant en paramètre le type de requête : si l’utilisateur choisit de visualiser les requêtes DNS, alors la chaîne de caractère « DNS » est passée en paramètre, et va donc rentrer dans le « if » concernant le DNS. Dans ce if, on va exécuter une requête SQL, retournant toutes les requêtes où le port source, ou le port destination est égal à 53 :

On va alors récupérer dans une liste toutes les lignes de la table concernant les requêtes qui seront retournées au client pour affichage.

On fait de même avec les requêtes DHCP, avec les paramètres « DHCP », qui, en rentrant dans le if approprié, va sélectionner tous les logs où les ports sources et destinations seront égaux à 67 ou 68. Cela va nous retourner toutes les lignes des requêtes DHCP.

La connexion à la base de données est fermée à chaque fin de requête.

**4/ Possibility to display the logs by @MAC or by the OUI file (Manufacturer)**

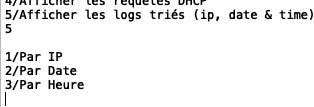
Une fois les requêtes récupérées comme précisé dans le point précédent, il faut les afficher. On souhaite les trier par adresses Mac, ou par le OUI File, qui contient tous les OUIs (Organizationally Unique Identifiers) enregistrés aux IEEE. Pour cela, dans nos requêtes précédentes il est nécessaire de rajouter « ORDER BY », avec les adresses mac sources, afin de les trier comme tel.

Une fois la liste triée et récupérée, on l’affiche à l’utilisateur, en faisant défiler chacun des index de notre liste, et en récupérant chaque paramètres (index 0 à 8) de notre liste afin de présenter toutes les informations. L’affichage final se présente comme tel.

Une image contenant texte

Description générée automatiquement

**5/ Possibility of sorting logs by (@IP, Date, Time ...)**

L’utilisateur a, dans le menu principal, la possibilité de choisir d’afficher les logs triés selon différents paramètres : l’IP, la date et l’heure. En choisissant cette option, il peut alors choisir le tri qu’il souhaite, qui va appeler une fonction avec le filtre correspondant en paramètre.

Cette fonction va alors en fonction du paramètre, exécuter une query « SELECT \* FROM logs», avec l’attribut « ORDER BY » correspondant en paramètres. Cela va alors nous placer dans une liste les différentes lignes, triées selon le paramètre voulu, liste qui sera ensuite retournée pour affichage à l’utilisateur. Pour un affichage par date, on a donc :

Une image contenant texte

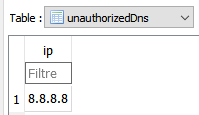
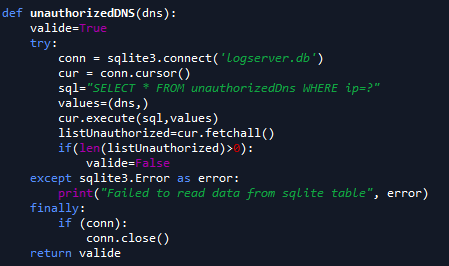
Description générée automatiquement

**Bonus: Ability to read and send an alert email if for example DDOS attempt**

Pour ce bonus, on souhaite envoyer un email lorsqu’il y a une tentative d’attaque DDOS par exemple, c’est-à-dire quand le trafic est sur le point de saturer.

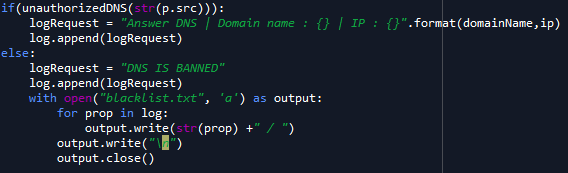
**6/ Managing a list of unauthorized DNS**

La liste des DNS non autorisés est référencée dans la base de données dans la table unauthorisedDns. Lors de la capture d’un paquet on vérifie que l’ip source et l’ip de destination ne sont pas dans cette table.

**7/ Notification if a DNS is in the list of unauthorized DNS**

Si le programme trouve une ip non autorisée, il change la nature de la requête et ajoute le log dans le fichier blacklist.txt pour avoir une trace des actions non autorisées.





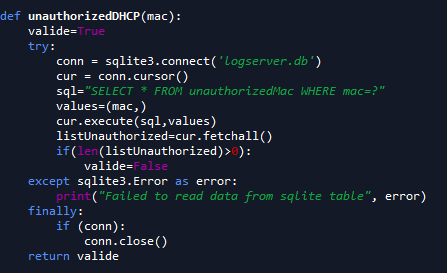
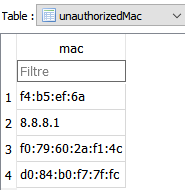
Console serveur



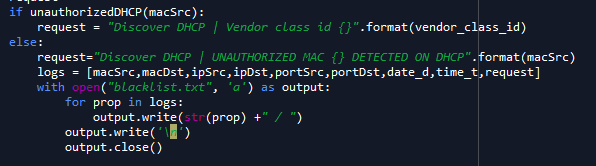
blacklist.txt

**8/ Notification if a DHCP @IP is in the list of unauthorized @MACs**

De la même manière que pour les DNS non autorisés, une table sqlite unauthorisedMac référence les adresses mac non autorisées.



Si une des adresses mac de la table est reçue dans la fonction unauthorizedDHCP(mac), on change la nature de la requete et on ajoute le log dans le fichier blacklist.txt .





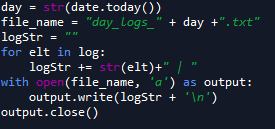
Console serveur



Blacklist.txt

**9/ Save logs to one file per day**

Avant chaque insertion dans la base de données, on écrit également le log dans le fichier du jour.



On récupère la date du jour pour nommer le fichier. Le log est récupéré à partir des de la liste « log » transformée en string, puis on ouvre le fichier en mode append (‘a’) pour ajouter le log à la suite du fichier.

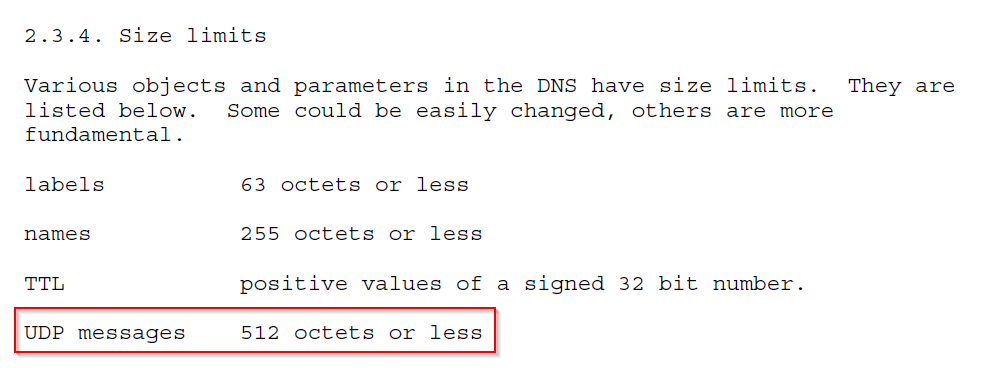
## 10) Notre caracteristique originale

Après échanges avec M. Livolsi nous nous sommes particulièrement intéressés aux serveurs DNS. Il a évoqué l’existence du DNS over HTTPS et du DNSSEC qui sont des évolutions plus sécurisées du DNS de base. Il est vrai que les informations transmises dans les paquets DNS sont sensibles, il parait ainsi logique de chercher à les sécuriser. Cependant la quasi-totalité des routeurs et servers DNS grand public n’implémentent pas ces sécurités.

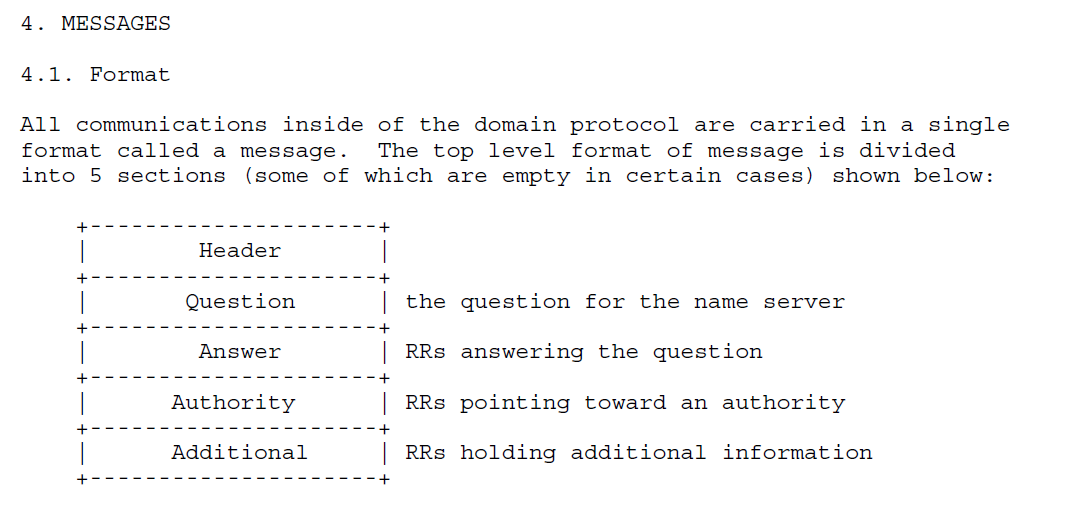
Nous avons donc décider de construire un vrai serveur DNS basique en Python pour se familiariser plus en profondeur avec les DNS.

Ce serveur communique avec des clients DNS (comme les navigateurs ou la commande **DIG**) via les sockets.

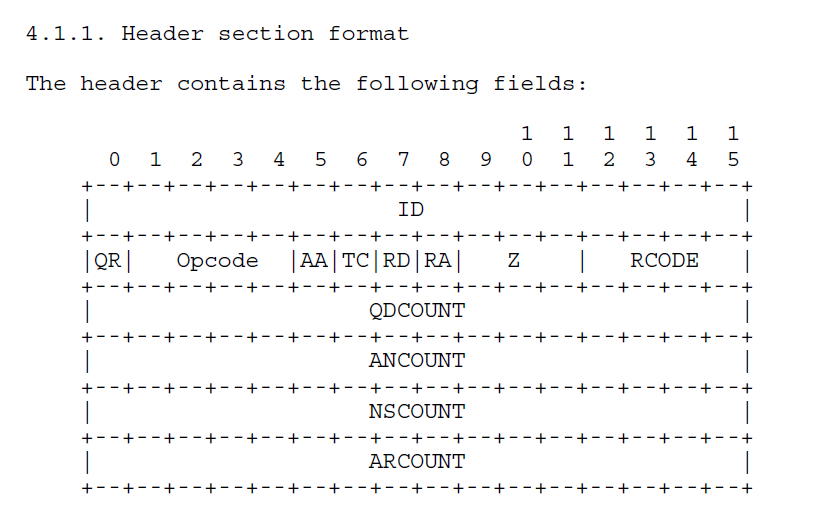
Pour commencer il faut comprendre la structure d’un paquet DNS. Pour ce faire, il faut lire la documentation des RFC 2929 et 1035.



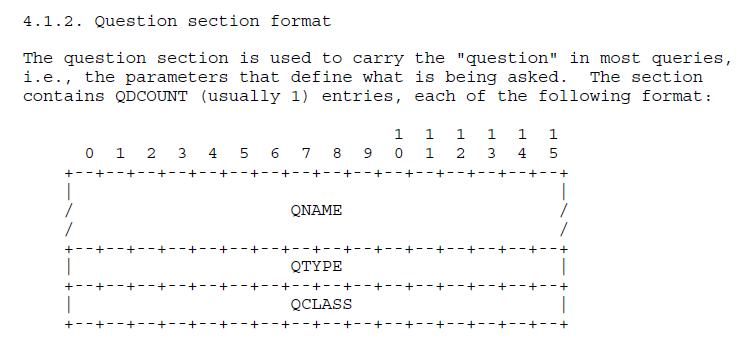
Notre serveur est assez basique et se contente de récupérer une Query et d’envoyer une Answer. Les tailles des paquets sont relativement petites, on utilise donc le protocole UDP.



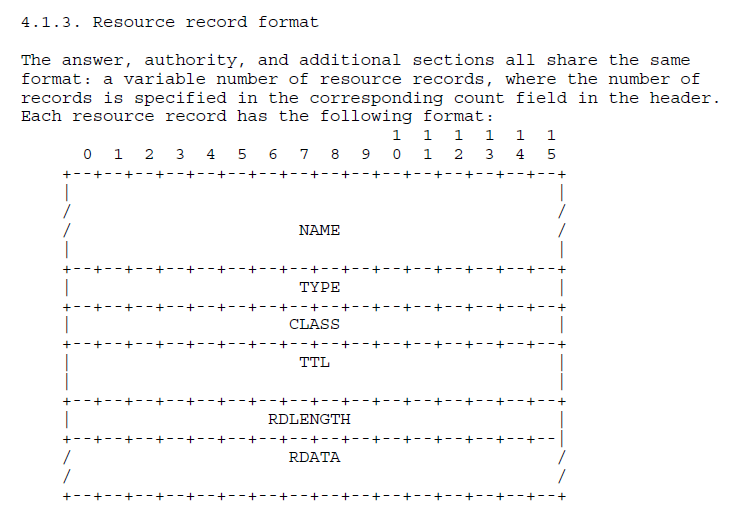
Les Query DNS se composent des sections **Header** et **Question**. Pour les Answer DNS, il faut ajouter à cela la section **Answer**. Nous n’utilisons pas les autres sections.



Seul l’ID change entre les requêtes DNS dans notre utilisation. Ce qui nous permet d’entrer tous les autres champs en dur lors de la construction du paquet de réponse.

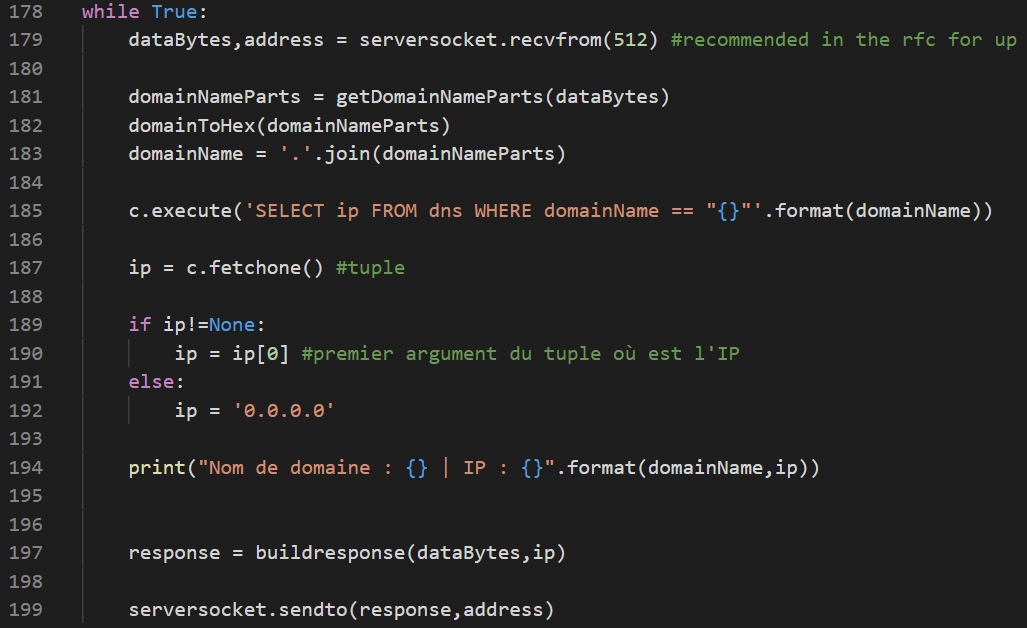


Ici c’est seulement le nom de domaine (QNAME) que l’on doit modifier entre les paquets.

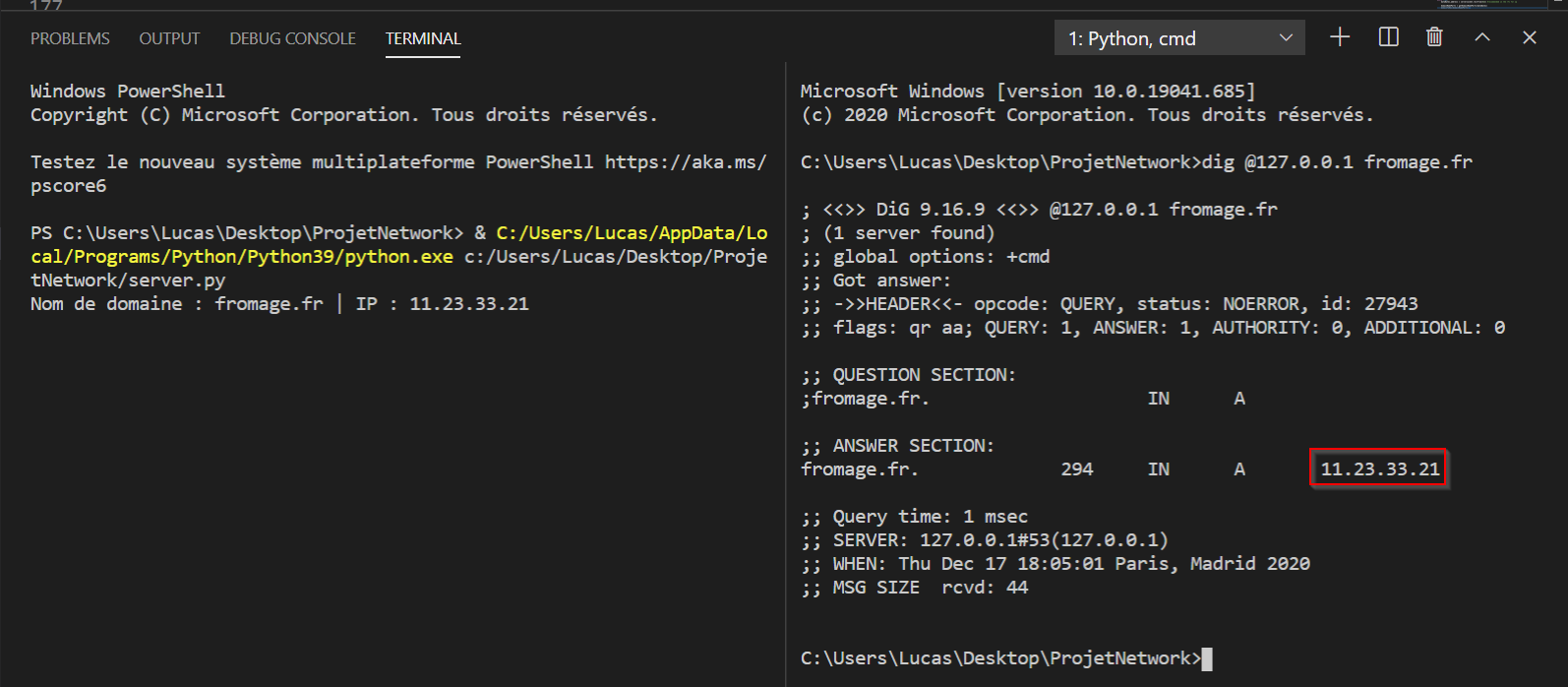


Après avoir renseigné tous ces champs, le paquet de réponse est prêt à être envoyer.

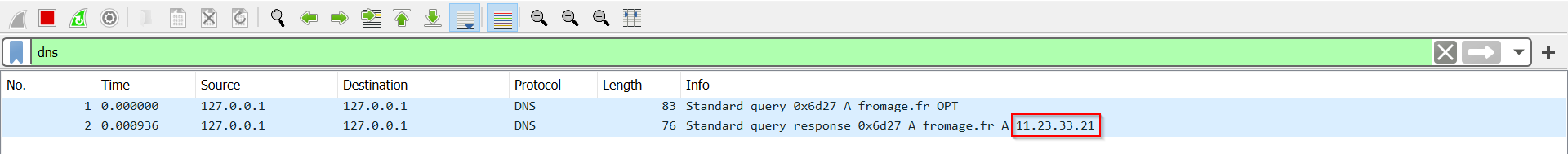
Voici la boucle principale du programme :



Le serveur cherche dans sa base de données s’il possède l’IP du nom de domaine dans la Query. S’il ne possède pas l’IP, il renvoie 0.0.0.0. Sinon il renvoie un paquet de réponse avec l’IP.



On voit ici que le client **DIG** reçoit un paquet DNS en bonne et due forme en tant que réponse.



On peut vérifier sur Wireshark que l’échange s’est bien déroulé.

Grâce à cette caractéristique originale du projet global, nous avons pu approfondir notre connaissance des serveurs DNS. Maintenant que nous comprenons en détails comment les échanges se déroulent, nous pouvons maintenant nous intéresser aux évolutions plus sécurisées telles que DNS over HTTPS ou DNSSEC.

BONUS

**1/ Envoi de l’email**

Concernant l’envoi de l’email, nous avons implémenter une fonction qui sera appelé au moment de l’attaque. Cette fonction utilise la libraire Python smtplib qui permet d’envoyer des emails en utilisant le protocole SMTP, qui utilise le RFC 821. Ici, nous allons utiliser le server Gmail SMTP pour envoyer nos mails.

Une image contenant texte

Description générée automatiquementPour envoyer le mail, il faut bien évidemment insérer son adresse mail, le mot de passe du compte (nous avons créé une nouvelle adresse mail pour l’occasion, [projetnetwork9@gmail.com](mailto:projetnetwork9@gmail.com)) et enfin l’adresse mail du destinataire. Il faut ensuite créer le contexte SSL sécurisé, grâce à la fonction ssl. create\_default\_context(), puis on peut procéder à l’envoi du mail.

On utilise SMTP\_SSL() pour envoyer le mail, avec en paramètre le server SMTP utilisé (Gmail), le port (465, port pour SSL) et enfin le contexte.



Avec cette fonction, le serveur va se login en utilisant les identifiants renseignés, puis envoyer le message en utilisant cette adresse mail. Une fois lancé, le script envoi bien le mail !

Une image contenant texte

Description générée automatiquement