

NETWORKS ARCHITECTURES PROJECT

SERVER PROJECT (DNS / DHCP TRACE)

BECKER LUCAS, NGO THOMAS, BUE MATHIEU, CORDIER BENOIT
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.

```
sniff(prn=mainSniff,filter="port 68 or port 67 or port 53",store=0)
```

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.

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 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 connaît la structure d'un paquet DHCP il est facile d'en récupérer les informations qui nous intéressent. Les adresses MAC, IP et les ports source et destination des paquets sont au même endroit.

Networks Architectures project

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

```
170     if p[3].options[0][1] == 1: #discover
171
172         vendor_class_id = p[3].options[3][1].decode()
173         request = ""
174         if unauthorizedDHCP(macSrc):
175             request = "Discover DHCP | Vendor class id {}".format(vendor_class_id)
176         else:
177             request="Discover DHCP | UNAUTHORIZED MAC {} DETECTED ON DHCP".format(macSrc)
178             logs = [macSrc,macDst,ipSrc,ipDst,portSrc,portDst,date_d,time_t,request]
179             with open("blacklist.txt", 'a') as output:
180                 for prop in logs:
181                     output.write(str(prop) + " / ")
182                 output.write('\n')
183                 output.close()
184
185     elif p[3].options[0][1] == 3: #request
186
187         vendor_class_id = p[3].options[5][1].decode()
188         ip_p = p[3].options[2][1]
189         request = ""
190         if unauthorizedDHCP(macSrc):
191             request = "Request DHCP | Vendor class id {} (()) requested {}".format(vendor_class_id,macSrc,ip_p)
192         else:
193             request="Request DHCP | UNAUTHORIZED MAC {} DETECTED ON DHCP FROM {}".format(macSrc,vendor_class_id)
194             logs = [macSrc,macDst,ipSrc,ipDst,portSrc,portDst,date_d,time_t,request]
195             with open("blacklist.txt", 'a') as output:
196                 for prop in logs:
197                     output.write(str(prop) + " / ")
198                 output.write('\n')
199                 output.close()
200
201     log = [macSrc,macDst,ipSrc,ipDst,portSrc,portDst,date_d,time_t,request]
```

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.

```
def mainSniff(p):

    if(p.dport == 53)or(p.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)

```

##### [ DNS ] #####
    id      = 330
    qr      = 0
    opcode   = QUERY
    aa      = 0
    tc      = 0
    rd      = 1
    ra      = 0
    z       = 0
    ad      = 1
    cd      = 0
    rcode    = ok
    qdcount  = 1
    ancount  = 0
    nscount  = 0
    arcount  = 1
    \qd      \
    |##### [ DNS Question Record ]#####
    |  qname   = 'louvre.fr.'
    |  qtype   = A
    |  qclass  = IN
    an       = None
    ns       = None
    \ar      \
    |##### [ DNS OPT Resource Record ]#####
    |  rrname  = '.'
    |  type    = OPT
    |  rclass  = 4096
    |  extrcode = 0
    |  version = 0
    |  z       = 0
    |  rdlen   = None
    |  \rdata   \
    |  |##### [ DNS EDNS0 TLV ]#####
    |  |  optcode = 10
    |  |  optlen  = 8
    |  |  optdata  = 'S\xfb\x7fdv\xe6\xfc\x0'

```

Il faut ensuite distinguer les paquets Query des paquets Answer :

```

typeRequest = p.qr

if typeRequest == 1 : #Answer
    pass

elif typeRequest == 0 : #Query
    pass

```

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 Answer :

```

domainName = p.qd.qname #type : <class 'bytes'>
domainName = domainName.decode() #type : <class 'str'>

```

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.

```
if typeRequest == 1 : #Answer

    if(unauthorizedDNS(str(p.src))):

        if p.ancount >= 1: #Si la réponse possède une IP de réponse correcte

            ip = p.an.rdata #type : <class 'str'>
            logRequest = "Answer DNS | Domain name : {} | IP : {}".format(domainName,ip)
            log.append(logRequest)
        else:
            logRequest = "Answer DNS | Domain name : {} | IP : Incorrect".format(domainName)
            log.append(logRequest)

    else:
        logRequest = "DNS IS BANNED"
        log.append(logRequest)
        with open("blacklist.txt", 'a') as output:
            for prop in log:
                output.write(str(prop) + " / ")
            output.write("\n")
            output.close()
```

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 :

```
elif typeRequest == 0 : #Query

    if(unauthorizedDNS(str(p.dst))):
        logRequest = "Query DNS | Domain name : {} ".format(domainName)
        log.append(logRequest)
    else:
        logRequest = "DNS IS BANNED"
        log.append(logRequest)
        with open("blacklist.txt", 'a') as output:
            for prop in log:
                output.write(str(prop) + " / ")
            output.write("\n")
            output.close()
```

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.

```

log = []
log.append(p.src)
log.append(p.dst)

p = p[1] #on passe à la couche IP

log.append(p.src)
log.append(p.dst)
log.append(p.sport)
log.append(p.dport)
log.append(str(date.today()))
log.append(str(datetime.today().time()))

```

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.

```

day = str(date.today())
file_name = "day_logs_" + day + ".txt"
logStr = ""
for elt in log:
    logStr += str(elt) + " | "
with open(file_name, 'a') as output:
    output.write(logStr + '\n')
output.close()

insertLog(log)
print(log)

```

TRAITEMENT DES LOGS

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

Comme 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. :

```

#macSrc = p.src
macSrc = "d0:84:b0:f7:7f:fc"
macDst = p.dst
- - -

```

	macSrc	macDst
	Filtre	Filtre
1	f4:6b:ef:6a:ad:c7	7c:67:a2:19:ec:e1
2	e5:6z:ef:6a:ad:c7	6t:1a:4r:65:8o:2b
3	7c:67:a2:19:ec:e0	f4:6b:ef:6a:ad:c4
4	f4:6b:ef:6a:ad:c4	7c:67:a2:19:ec:e0
5	7c:67:a2:19:ec:e0	f4:6b:ef:6a:ad:c4
6	f4:6b:ef:6a:ad:c4	7c:67:a2:19:ec:e0

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

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

```
DNS DHCP Log server

1/Afficher les logs du jour
2/Afficher tous les logs
3/Afficher les requêtes DNS
4/Afficher les requêtes DHCP
5/Afficher les logs triés (ip, date & time)
```

Concernant les requêtes DHCP et DNS, il est nécessaire dans ce fichier de les récupérer avec toutes

```
#Display DNS logs
elif(request=="dns"):
    cur.execute("SELECT * FROM logs WHERE portSrc=53 or portDst=53 ORDER BY macSrc")

#Display DHCP logs
elif(request=="dhcp"):
    cur.execute("SELECT * FROM logs WHERE portSrc=67 or portSrc=68 "+
               "or portDst=67 or portDst=68 ORDER BY macSrc")
```

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.

```
if request=="all":
    cur.execute("SELECT * FROM logs ORDER BY macSrc")
```


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.

```
macSrc :f0:79:60:2a:f1:4c macDst :2c:39:96:31:a7:8cIpSrc :192.168.1.26 IpDst :8.8.8.8portSrc :62930 portDst :53Date :2020-12-17 Time :
21:24:32.288037
Request :Query DNS | Domain name : me.apple-dns.net.

macSrc :f0:79:60:2a:f1:4c macDst :2c:39:96:31:a7:8cIpSrc :192.168.1.26 IpDst :8.8.8.8portSrc :57547 portDst :53Date :2020-12-17 Time :
21:24:32.294153
Request :Query DNS | Domain name : me.apple-dns.net.

macSrc :f4:6b:ef:6a:ad:c4 macDst :7c:67:a2:19:ec:e0IpSrc :89.2.0.1 IpDst :192.168.0.10portSrc :53 portDst :50806Date :2020-12-17 Time :
19:38:55.126373
Request :Answer DNS | Domain name : louvre.fr. | IP : 89.185.38.253

macSrc :f4:6b:ef:6a:ad:c4 macDst :7c:67:a2:19:ec:e0IpSrc :89.2.0.1 IpDst :192.168.0.10portSrc :53 portDst :51135Date :2020-12-17 Time :
19:38:58.610921
Request :Answer DNS | Domain name : vortex.data.microsoft.com. | IP : b'asimov.vortex.data.trafficmanager.net.'
```

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.

```
4/Afficher les requetes dnsc
5/Afficher les logs triés (ip, date & time)
5

1/Par IP
2/Par Date
3/Par Heure
|
```

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 :


```
d0:84:b0:f7:7f:fc : DNS IS BANNED (2020-12-18)
d0:ab:d5:da:a6:f1 : Query DNS | Domain name :
d0:84:b0:f7:7f:fc : Answer DNS | Domain name :
central-1.elb.amazonaws.com.' (2020-12-18)
d0:ab:d5:da:a6:f1 : Query DNS | Domain name :
d0:84:b0:f7:7f:fc : Answer DNS | Domain name :
central-1.elb.amazonaws.com.' (2020-12-18)
d0:ab:d5:da:a6:f1 : DNS IS BANNED (2020-12-18)
```

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 unauthorizedDns. Lors de la capture d'un paquet on vérifie que l'ip source et l'ip de destination ne sont pas dans cette table.

Table :  unauthorizedDns

	ip
	<input type="text" value="Filtre"/>
1	8.8.8.8

```
def unauthorizedDNS(dns):
    valide=True
    try:
        conn = sqlite3.connect('logserver.db')
        cur = conn.cursor()
        sql="SELECT * FROM unauthorizedDns WHERE ip=?"
        values=(dns,)
        cur.execute(sql,values)
        listUnauthorized=cur.fetchall()
        if(len(listUnauthorized)>0):
            valide=False
    except sqlite3.Error as error:
        print("Failed to read data from sqlite table", error)
    finally:
        if (conn):
            conn.close()
    return valide
```

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.

```
if(unauthorizedDNS(str(p.src))):
    logRequest = "Answer DNS | Domain name : {} | IP : {}".format(domainName,ip)
    log.append(logRequest)
else:
    logRequest = "DNS IS BANNED"
    log.append(logRequest)
    with open("blacklist.txt", 'a') as output:
        for prop in log:
            output.write(str(prop) + " / ")
            output.write("\n")
        output.close()
```

```
DNS IS BANNED
['d0:ab:d5:da:a6:f1', 'd0:84:b0:f7:7f:fc', '192.168.0.11', '8.8.8.8', 62404, 53, '2020-12-18', '00:29:01.089764', 'DNS IS BANNED']
```


Console serveur

```
d0:ab:d5:da:a6:f1 / d0:84:b0:f7:7f:fc / 192.168.0.11 / 8.8.8.8 / 62404 / 53 / 2020-12-18 / 00:29:01.483734 / DNS IS BANNED /
```

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.

Table :  unauthorizedMac

	mac
	<input type="text" value="Filtre"/>
1	f4:b5:ef:6a
2	8.8.8.1
3	f0:79:60:2a:f1:4c
4	d0:84:b0:f7:7f:fc

```
def unauthorizedDHCP(mac):
    valide=True
    try:
        conn = sqlite3.connect('logserver.db')
        cur = conn.cursor()
        sql="SELECT * FROM unauthorizedMac WHERE mac=?"
        values=(mac,)
        cur.execute(sql,values)
        listUnauthorized=cur.fetchall()
        if(len(listUnauthorized)>0):
            valide=False
    except sqlite3.Error as error:
        print("Failed to read data from sqlite table", error)
    finally:
        if (conn):
            conn.close()
    return valide
```

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 .

```
if unauthorizedDHCP(macSrc):
    request = "Discover DHCP | Vendor class id {}".format(vendor_class_id)
else:
    request="Discover DHCP | UNAUTHORIZED MAC {} DETECTED ON DHCP".format(macSrc)
    logs = [macSrc,macDst,ipSrc,ipDst,portSrc,portDst,date_d,time_t,request]
    with open("blacklist.txt", 'a') as output:
        for prop in logs:
            output.write(str(prop) + " / ")
        output.write('\n')
        output.close()
```

```
['d0:84:b0:f7:7f:fc', 'ff:ff:ff:ff:ff:ff', '0.0.0.0', '255.255.255.255', 68, 67, '2020-12-18', '00:29:14.659732', 'Request DHCP | UNAUTHORIZED MAC d0:84:b0:f7:7f:fc DETECTED ON DHCP FROM MSFT 5.0']
```

Console serveur

```
d0:84:b0:f7:7f:fc / ff:ff:ff:ff:ff:ff / 0.0.0.0 / 255.255.255.255 / 68 / 67 / 2020-12-18 / 00:29:14.659732 / Request DHCP | UNAUTHORIZED MAC d0:84:b0:f7:7f:fc DETECTED ON DHCP FROM MSFT 5.0 /
```

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.

```
day = str(date.today())
file_name = "day_logs_" + day + ".txt"
logStr = ""
for elt in log:
    logStr += str(elt) + " / "
with open(file_name, 'a') as output:
    output.write(logStr + '\n')
output.close()
```

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 paraît ainsi logique de chercher à les sécuriser. Cependant la quasi-totalité des routeurs et serveurs DNS grand public n'implémentent pas ces sécurités.

Nous avons donc décidé 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.

2.3.4. Size limits

Various objects and parameters in the DNS have size limits. They are listed below. Some could be easily changed, others are more fundamental.

labels	63 octets or less
names	255 octets or less
TTL	positive values of a signed 32 bit number.
UDP messages	512 octets or less

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.

4. MESSAGES

4.1. Format

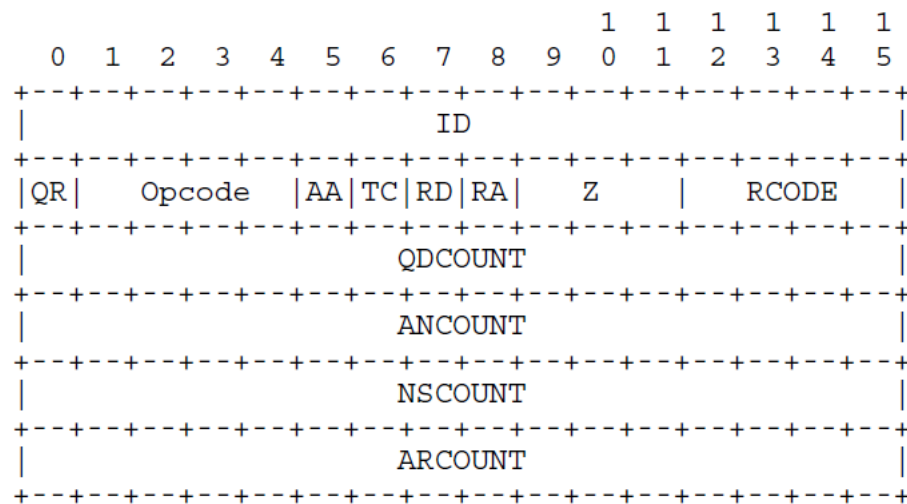
All communications inside of the domain protocol are carried in a single format called a message. The top level format of message is divided into 5 sections (some of which are empty in certain cases) shown below:

Header	
Question	the question for the name server
Answer	RRs answering the question
Authority	RRs pointing toward an authority
Additional	RRs holding additional information

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.

4.1.1. Header section format

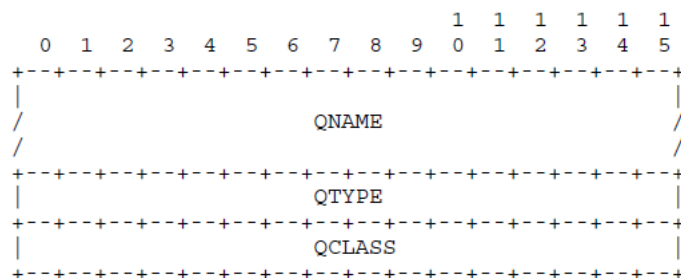
The header contains the following fields:



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.

4.1.2. Question section format

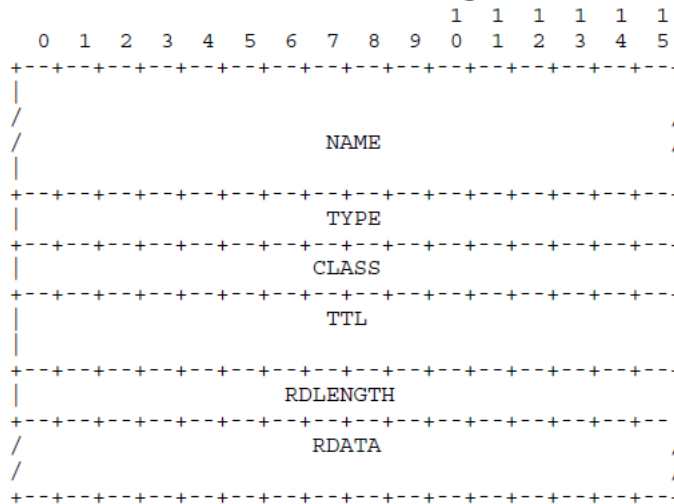
The question section is used to carry the "question" in most queries, i.e., the parameters that define what is being asked. The section contains QDCOUNT (usually 1) entries, each of the following format:



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

4.1.3. Resource record format

The answer, authority, and additional sections all share the same format: a variable number of resource records, where the number of records is specified in the corresponding count field in the header. Each resource record has the following format:

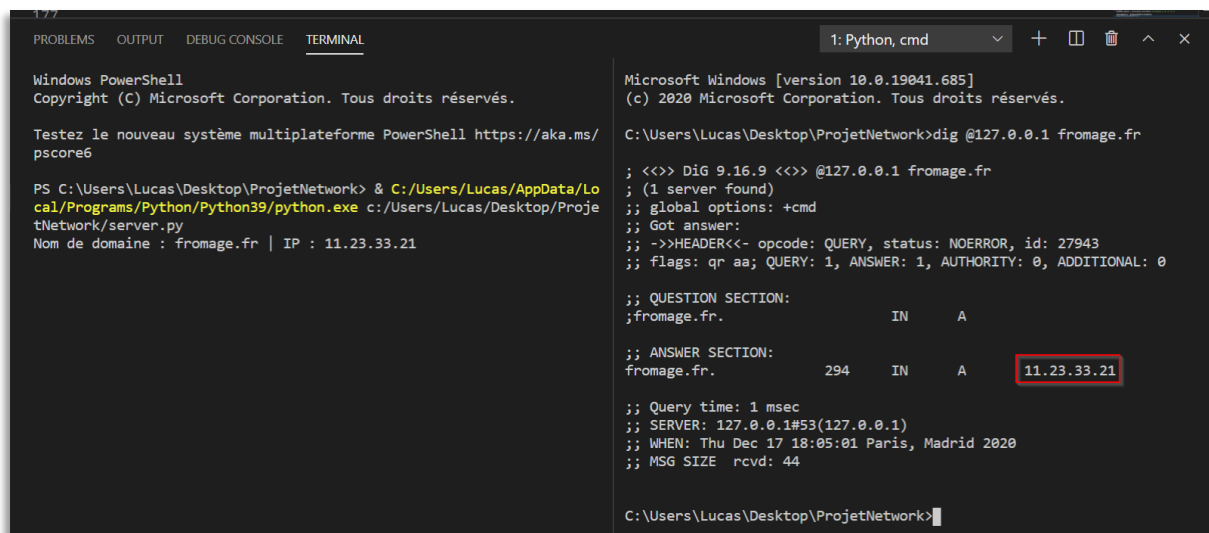


Après avoir renseigné tous ces champs, le paquet de réponse est prêt à être envoyé.

Voici la boucle principale du programme :

```
178 while True:
179     dataBytes,address = serversocket.recvfrom(512) #recommended in the rfc for up
180
181     domainNameParts = getDomainNameParts(dataBytes)
182     domainToHex(domainNameParts)
183     domainName = '.'.join(domainNameParts)
184
185     c.execute('SELECT ip FROM dns WHERE domainName == "{}"'.format(domainName))
186
187     ip = c.fetchone() #tuple
188
189     if ip!=None:
190         ip = ip[0] #premier argument du tuple où est l'IP
191     else:
192         ip = '0.0.0.0'
193
194     print("Nom de domaine : {} | IP : {}".format(domainName,ip))
195
196
197     response = buildresponse(dataBytes,ip)
198
199     serversocket.sendto(response,address)
```

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.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS C:\Users\Lucas\Desktop\ProjetNetwork> & C:/Users/Lucas/AppData/Local/Programs/Python/Python39/python.exe c:/Users/Lucas/Desktop/ProjetNetwork/server.py
Nom de domaine : fromage.fr | IP : 11.23.33.21

C:\Users\Lucas\Desktop\ProjetNetwork>dig @127.0.0.1 fromage.fr

; <<>> DiG 9.16.9 <<>> @127.0.0.1 fromage.fr
; (1 server found)
; global options: +cmd
; Got answer:
; -->HEADER<<- opcode: QUERY, status: NOERROR, id: 27943
; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;fromage.fr.                IN      A

;; ANSWER SECTION:
fromage.fr.                294     IN      A      11.23.33.21

;; Query time: 1 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Thu Dec 17 18:05:01 Paris, Madrid 2020
;; MSG SIZE rcvd: 44

C:\Users\Lucas\Desktop\ProjetNetwork>
```

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



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	DNS	83	Standard query 0x6d27 A fromage.fr OPT
2	0.000936	127.0.0.1	127.0.0.1	DNS	76	Standard query response 0x6d27 A fromage.fr A 11.23.33.21

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: ABILITY SEND AN ALERT EMAIL IF FOR EXAMPLE DDOS ATTEMPT

1/ DDOS

Une attaque par déni de service (DDOS) vise à rendre indisponible un ou plusieurs services. Il en existe de plusieurs types. Nous allons voir dans ce projet les attaques basées sur l'amplification. Elle a pour objectif d'épuiser la bande passante réseau disponible afin de rendre un ou plusieurs services inaccessibles. Pour cela, on va envoyer un très grand nombre de paquets en très peu de temps. Parmi les protocoles permettant cette attaque, le DNS est l'un des plus exploités, d'où notre choix.

Pour mettre en place sur notre serveur un détecteur de DDOS, on va scanner, à chaque réception d'un paquet DNS, les 50 derniers paquets DNS de la base de données (pour un serveur plus grand, le nombre serait bien plus large que 50, mais en guise de test, nous nous limiterons à ce nombre).

```
def preventDDOS():
    timeList=[]
    try:
        conn = sqlite3.connect('logserver.db')
        cur = conn.cursor()
        cur.execute("SELECT date,time FROM logs WHERE portSrc=53 OR portDST=53 ORDER BY date desc,time desc LIMIT 50")
        timeList=cur.fetchall()
    except sqlite3.Error as error:
        print("Failed to read data from sqlite table", error)
    finally:
        if (conn):
            conn.close()
```

La fonction preventDDOS() récupère donc ces paquets, et s'intéresse à leur date et heure. On compare le premier et le dernier des 50 paquets. Les heures étant stockées en string, il est nécessaire de les repasser en format date/heure, puis de faire la différence entre les deux. Si la différence de leur heure d'ajout à la base de données est inférieure à une seconde, limite fixée arbitrairement, alors il y a peut-être une tentative de DDOS et on envoie un mail, partie détaillée par la suite.

```
#On récupère le premier paquet sur 50 et le dernier paquet, qu'on passe format date
time = datetime.strptime(timeList[0][1], '%H:%M:%S.%f')
time2=datetime.strptime(timeList[49][1], '%H:%M:%S.%f')

#On calcule la différence de temps entre les deux
delta=datetime.strptime(str(time-time2), '%H:%M:%S.%f')

#On fixe la limite à une seconde
now=datetime.now()
limit=now.replace(year=1900,month=1,day=1,hour=0,minute=0,second=1,microsecond=0)

#Si la différence de temps est inférieure à la limite, on envoie un mail
if delta<limit:
    mail()
```

2/ 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 librairie 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.

Pour 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) 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.

```
portEmail = 465 # For SSL
smtp_server = "smtp.gmail.com"
sender_email = "projetnetwork9@gmail.com" # Enter your address
receiver_email = "projetnetwork9@gmail.com" # Enter receiver address
message = """\
Subject: DDOS ATTEMPT

Server received a lot of request.
There is a high probability that a ddos is trying to shutdown the server."
```


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.

```
context = ssl.create_default_context()
with smtplib.SMTP_SSL(smtp_server, portEmail, context=context) as server:
    server.login(sender_email, "ProjetNetwork1!")
    server.sendmail(sender_email, receiver_email, message)
```

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 envoie bien le mail !



projetnetwork9@gmail.com

À cci : moi ▾

This message is sent from Python.