

Virtualisation

1) Automatisation démarrage des machines

Pour le démarrage des machines, un simple script bash a été réalisé. L'exécution du script se fait sur la machine hôte, avec le compte root, à l'aide de virsh.

```
root@debian:/home/toto# ./startMachine.sh
Domain machinePox started
```

```
Domain machineMininet started
```

```
root@debian:/home/toto# █
```

2) Démarrage automatique des services

Pour démarrer sur les machines virtuelles les services associés (Pox et Mininet), j'ai utilisé les crontable de ces machines afin de lancer les scripts python réalisés, dès le démarrage de la machine, comme suit :

```
""
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow  command
@reboot python /root/Doment/pox/pox.py ctrlr█
```

3) Partie Mininet

Le réseau donné en consigne a été réalisé avec un script python :

```

if __name__=="__main__":
    topo=Topo()
    ##-----SUBNET 1
    s1=topo.addSwitch("s1")
    webServTarg=topo.addHost("wst")
    tg1=topo.addHost("tg1")
    tg2=topo.addHost("tg2")
    topo.addLink(webServTarg,s1)
    topo.addLink(tg1,s1)
    topo.addLink(tg2,s1)

    ##-----SUBNET 2
    s2=topo.addSwitch("s2")
    webCli1=topo.addHost("wc1")
    webCli2=topo.addHost("wc2")
    dosLaunch=topo.addHost("dosL")
    topo.addLink(s2,webCli1)
    topo.addLink(s2,webCli2)
    topo.addLink(s2,dosLaunch)

    ##-----Switch to Switch
    topo.addLink(s2,s1)

    ##-----CONTROLLEUR
    opFlowSwitch= RemoteController('c','192.168.122.118',6633)
    net=Mininet(topo=topo,controller=opFlowSwitch)
    print("Starting Network")
    net.start()
    print("Network started\n")

```

La suite du TP est l'attaque d'un service web que nous démarrons de la manière suivante :

```
def startService(net):
    for host in net.hosts:
        if host.name=="wst":
            print("Starting Web Service target")
            host.cmd('python -m SimpleHTTPServer 80 &')
            print("Web Service Started")
```

Dans ce script python a été ajouté deux parties : la génération d'un trafic réseau « normal » et la génération d'une attaque. Chaque partie est faite pour mettre à l'épreuve la partie suivante qui concernera Pox.

Pour la génération de trafic un simple ping sera utilisé comme suit :

```
def generateTraffic(net):
    print("Traffic generation :")
    net.pingAll()
    print("\n")
```

La partie attaque consiste à faire des requêtes HTTP sur la chaîne vers la machine cible :

```
def attack(net):
    for host in net.hosts:
        if host.name=="dosL":
            dosl=host
        if host.name=="wst":
            targ=host
    print("Starting attack service")
    for i in range(100):
        dosl.cmd('wget -o - %s &' % targ.IP())
    print("attack done")
```

4) Partie Pox

Pox est le contrôleur mis en place, il servira à la détection d'attaque de type déni de service sur le réseau.

Voici comment est géré la détection :

Pour chaque paquet qui transite dans le réseau, on stock le couple source+destination ainsi que l'heure de transmission. La façon dont sont stocké les informations est la suivante :

Un dictionnaire ayant pour clé le couple source+destination et pour valeur un tableau stockant les heures

Pour l'exercice, j'ai arbitrairement choisi une tolérance à 10 requêtes par seconde.

Pour chaque paquet émis, on vérifie si les 10 derniers paquets ont été envoyé dans la seconde.

Si non : le paquet est transmis et les informations sauvegardées.

Si oui : le paquet est « droppé » mais les informations le concernant sont malgré tout stocké afin de suivre la continuité de l'attaque.

Afin d'éviter les problèmes que pourrai engendrer le stockage des logs, seul une quantité nécessaire à l'analyse de donnée est gardée (10 logs par couple).

Activité avec trafic avant l'attaque :

```
46:4f:9f:ea:1d:93 destination known. only send message to it
33:33:00:00:00:16 not known, resend to everybody
33:33:00:00:00:16 not known, resend to everybody
b6:8d:5e:4c:31:91 destination known. only send message to it
b6:8d:5e:4c:31:91 destination known. only send message to it
12:79:6e:7b:90:7d destination known. only send message to it
12:79:6e:7b:90:7d destination known. only send message to it
9a:b1:15:d9:66:66 destination known. only send message to it
33:33:00:00:00:16 not known, resend to everybody
12:79:6e:7b:90:7d destination known. only send message to it
33:33:00:00:00:16 not known, resend to everybody
4e:53:8c:a3:eb:6a destination known. only send message to it
12:79:6e:7b:90:7d destination known. only send message to it
1e:24:17:b8:2e:bc destination known. only send message to it
1e:24:17:b8:2e:bc destination known. only send message to it
12:79:6e:7b:90:7d destination known. only send message to it
12:79:6e:7b:90:7d destination known. only send message to it
46:4f:9f:ea:1d:93 destination known. only send message to it
46:4f:9f:ea:1d:93 destination known. only send message to it
12:79:6e:7b:90:7d destination known. only send message to it
12:79:6e:7b:90:7d destination known. only send message to it
33:33:00:00:00:16 not known, resend to everybody
33:33:00:00:00:02 not known, resend to everybody
33:33:00:00:00:16 not known, resend to everybody
33:33:00:00:00:02 not known, resend to everybody

root@mininet:/home/mininet/Documents# python test.py
Starting Network
Network started

Starting Web Service target
Web Service Started
Traffic generation :
*** Ping: testing ping reachability
dosL -> tg1 tg2 wc1 wc2 wst
tg1 -> dosL tg2 wc1 wc2 wst
tg2 -> dosL tg1 wc1 wc2 wst
wc1 -> dosL tg1 tg2 wc2 wst
wc2 -> dosL tg1 tg2 wc1 wst
wst -> dosL tg1 tg2 wc1 wc2
*** Results: 0% dropped (30/30 received)
```

Activité pendant l'attaque :

