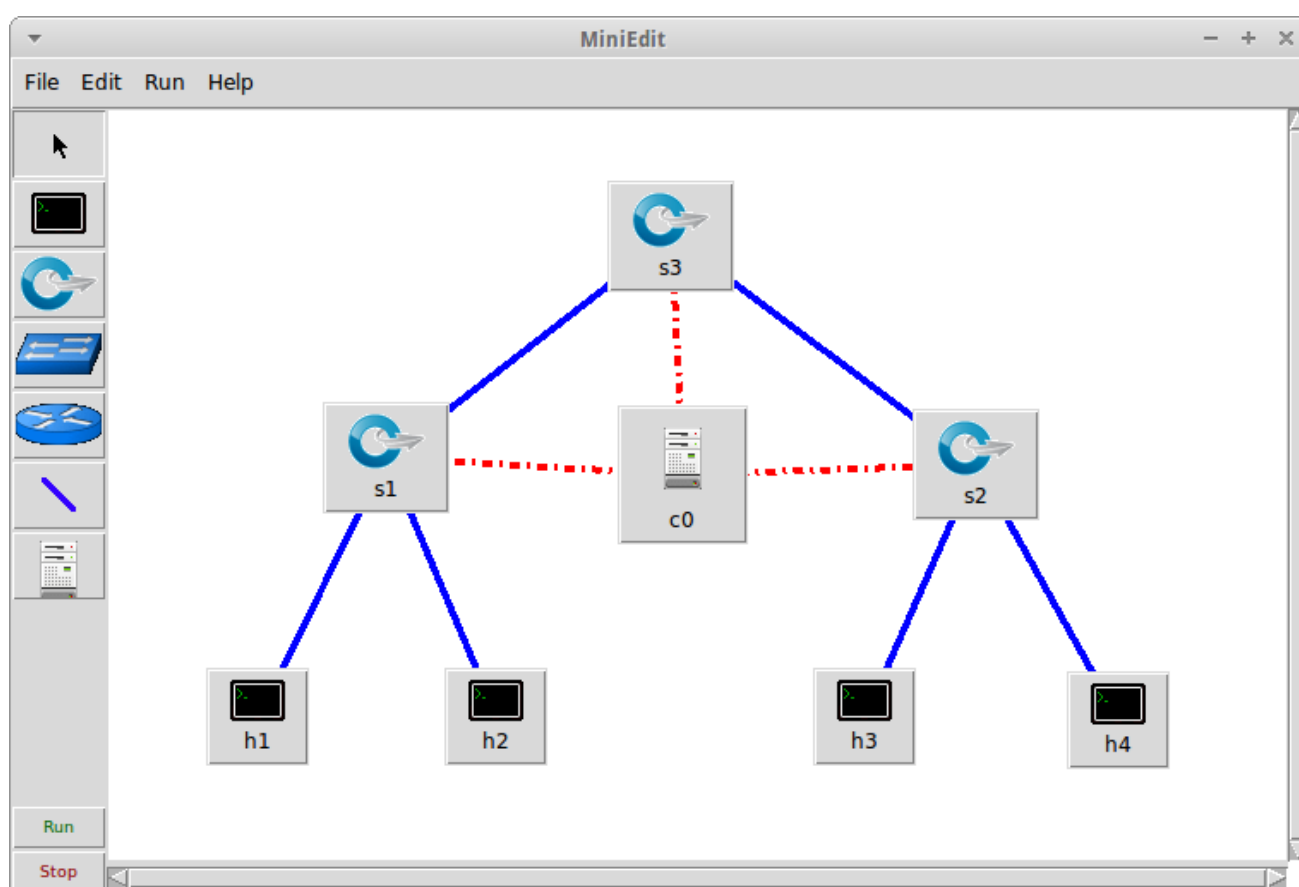


## Projet Privacité et Sécurité des Données



Aubert Antoine

Birembaux Yann

10/05/2024

## Table des matières

Objectif .....	3
Partie 1 : Mise en œuvre de l'environnement virtualisé.....	3
Introduction de l'environnement virtualisé.....	3
Installation et configuration de l'environnement virtualisé .....	4
Partie 2 : Simulation et gestion du réseau avec Mininet et POX .....	6
Configuration de la topologie réseau avec Mininet.....	6
Contrôle du réseau avec POX .....	7
Intégration avec Mininet.....	8
Scripts et automatisation.....	6
Partie 3 : Détection et blocage des attaques avec POX .....	8
Détection des Attaques .....	8
Détection de l'Attaque ARP Spoofing.....	8
Détection de l'Attaque DoS (Denial of Service) .....	9
Implémentation Technique .....	8
Conclusion .....	10

## Objectif

Dans le cadre de l'étude de la privacité et de la sécurité des données, ce rapport documente la création d'un environnement utilisant Docker et Docker Compose permettant de simuler un réseau informatique avec Mininet et POX et les processus déployés pour réaliser des attaques ARP Spoofing et DoS. Il explique en détail les techniques de détection et de riposte, en utilisant le contrôleur SDN POX.

## Partie 1 : Mise en œuvre de l'environnement virtualisé

### Introduction de l'environnement virtualisé

Docker permet de conteneuriser les applications, offrant ainsi une solution légère et portable pour exécuter des logiciels de manière cohérente sur différentes machines.

Docker Compose est un outil qui permet de définir et de gérer des applications multi-conteneurs. Grâce à un fichier de configuration unique (docker-compose.yml), il est possible de décrire les services nécessaires et de les orchestrer facilement.

Mininet est un émulateur de réseau qui crée un réseau virtuel comprenant des hôtes, des switches, des routeurs et des liens. Il est largement utilisé pour tester des protocoles réseau et des applications dans un environnement contrôlé.

POX est un contrôleur SDN (Software Defined Networking) écrit en Python. Il permet de contrôler et de gérer les switches OpenFlow dans un réseau, offrant une grande flexibilité pour la gestion du trafic et l'implémentation de règles de sécurité.

## Installation et configuration de l'environnement virtualisé

### Installation de Docker :

Mettre à jour les paquets existants, Installer les dépendances nécessaires, Ajouter la clé GPG et le dépôt Docker, Installer Docker, Vérifier l'installation.

Utilisation de la commande snap pour installer le docker

```
apt-get update
apt-get upgrade
sudo snap install docker
```

### Installation de Mininet et POX :

Création du fichier dockerfile pour mininet :

```
FROM ubuntu:latest
RUN apt-get update \
    && apt-get install -y python3 python3-pip openvswitch-switch iproute2 \
    && apt-get clean \
    && apt-get install iputils-ping -y \
    && rm -rf /var/lib/apt/lists/*
ENV PIP_BREAK_SYSTEM_PACKAGES 1
RUN pip3 install mininet
RUN pip3 install scapy
COPY requirements.txt requirements.txt
COPY CodePythonMininet.py CodePythonMininet.py
COPY poison_h2.py poison_h2.py
RUN service openvswitch-switch start

EXPOSE 6634
ENTRYPOINT ["sh"]
```

Création du fichier dockerfile pour Pox :

```
FROM ubuntu:latest

RUN apt-get update \
    && apt-get install -y python3 python3-pip mininet openvswitch-switch \
    && apt-get install -y git \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/*
RUN apt-get update && apt-get install -y iproute2
RUN git clone 'https://github.com/noxrepo/pox.git'
WORKDIR /pox
EXPOSE 6634
COPY . /pox

COPY fichier.py /pox/pox/forwarding/fichier.py
CMD ["/pox.py", "of_tutorial", "openflow.spanning_tree", "openflow.discovery", "openflow.of_01", "--address=192.168.12.25", "--port=6634", "forwarding.l2_learning"]
#CMD ["/pox.py", "forwarding.fichier"]

ENTRYPOINT ["sh"]
```

## Configuration avec Docker Compose :

Création du fichier docker-compose.yml avec le contenu suivant

```
version: '3'
networks:
  POXetMININET:
    ipam:
      config:
        - subnet: 192.168.12.0/24
          gateway: 192.168.12.1
services:
  mininet:
    user: root
    tty: true
    stdin_open: true
    build:
      context: ./mininet
    volumes:
      - /lib/modules:/lib/modules
    hostname: mininet
    networks:
      POXetMININET:
        ipv4_address: 192.168.12.45
    links:
      - pox
    privileged: true
  pox:
    build:
      context: ./pox
    ports:
      - 6634:6634
    hostname: pox
    networks:
      POXetMININET:
        ipv4_address: 192.168.12.25
    tty: true
```

## **Lancement et vérification :**

Lancement de la commande pour crée ou recrée les différents conteneurs :

```
docker compose up -d --force-recreate
```

Vérification du bon fonctionnement des dockers :

```
docker container ls
```

```
root@VM-ubuntu:/home/ubuntu/LesGolmons/mininet# docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1ab1cd7aed57	lesgolmons-mininet	"sh"	35 seconds ago	Up 33 seconds	6634/tcp	lesgolmons-mininet-1
c6dief7d6e6d	lesgolmons-pox	"sh"	35 seconds ago	Up 34 seconds	0.0.0.0:6634->6634/tcp, :::6634->6634/tcp	lesgolmons-pox-1

Vérification du ping entre les deux dockers :

```
docker exec -i lesgolmons-mininet-1 sh
```

```
ping 192.168.12.25
PING 192.168.12.25 (192.168.12.25) 56(84) bytes of data.
64 bytes from 192.168.12.25: icmp_seq=1 ttl=64 time=0.133 ms
64 bytes from 192.168.12.25: icmp_seq=2 ttl=64 time=0.089 ms
64 bytes from 192.168.12.25: icmp_seq=3 ttl=64 time=0.083 ms
64 bytes from 192.168.12.25: icmp_seq=4 ttl=64 time=0.077 ms
64 bytes from 192.168.12.25: icmp_seq=5 ttl=64 time=0.070 ms
64 bytes from 192.168.12.25: icmp_seq=6 ttl=64 time=0.077 ms
64 bytes from 192.168.12.25: icmp_seq=7 ttl=64 time=0.073 ms
```

## Partie 2 : Simulation et gestion du réseau avec Mininet et POX

### Configuration de la topologie réseau avec Mininet

On créer une topologie simple :

```
#!/usr/bin/python

from mininet.net import Mininet
from mininet.node import Controller, RemoteController, OVSwitch, Node
from mininet.cli import CLI
from mininet.log import setLogLevel, info
from mininet.link import TCLink, Intf

def myNetwork():

    net = Mininet(topo=None,
                  build=False,
                  ipBase='192.168.12.0/8')

    info('*** Adding controller\n')
    # Add a remote controller
    c0 = net.addController(name='c0',
                          controller=RemoteController,
                          ip='192.168.12.1',
                          protocol='tcp',
                          port=6633)

    info('*** Add switches\n')
    s1 = net.addSwitch('s1', cls=OVSwitch)
    s2 = net.addSwitch('s2', cls=OVSwitch)
    r1 = net.addHost('r1', cls=Node, ip='192.168.12.1/24')
    r1.cmd('sysctl -w net.ipv4.ip_forward=1')

    info('*** Add hosts\n')
    h1 = net.addHost('h1', cls=Node, ip='192.168.12.2/24', defaultRoute='via 192.168.12.1')
    h2 = net.addHost('h2', cls=Node, ip='192.168.12.3/24', defaultRoute='via 192.168.12.1')
    h3 = net.addHost('h3', cls=Node, ip='192.168.12.4/24', defaultRoute='via 192.168.12.1')
    h4 = net.addHost('h4', cls=Node, ip='192.168.12.5/24', defaultRoute='via 192.168.12.1')
    h5 = net.addHost('h5', cls=Node, ip='192.168.12.6/24', defaultRoute='via 192.168.12.1')
    h6 = net.addHost('h6', cls=Node, ip='192.168.12.7/24', defaultRoute='via 192.168.12.1')

    info('*** Add links\n')
    net.addLink(s1, r1, intfName2='r1-eth1', params2={'ip': '192.168.12.1/24'})
    net.addLink(s2, r1, intfName2='r1-eth2', params2={'ip': '192.168.12.1/24'})

    net.addLink(s1, h1)
    net.addLink(s1, h2)
    net.addLink(s1, h3)

    net.addLink(s2, h4)
    net.addLink(s2, h5)
    net.addLink(s2, h6)

    info('*** Starting network\n')
    net.build()
    info('*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info('*** Starting switches\n')
    net.get('s1').start([c0])
    net.get('s2').start([c0])

    info('*** Post configure switches and hosts\n')
    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    myNetwork()
```

Contrôleur : Un contrôleur SDN distant est ajouté (RemoteController).

Switches : Deux switches OpenFlow (s1 et s2) sont ajoutés.

Routeur : Un routeur (r1) interconnecte les deux sous-réseaux.

Hosts :

h1, h2, h3 : Serveur web et générateurs de trafic.

h4, h5, h6 : Clients HTTP et lanceur d'attaques.

## Contrôle du réseau avec POX

On démarre POX avec le script forwarding.l2\_learning :

./pox.py forwarding.l2\_learning

Ce script implémente un switch de niveau 2 (L2) qui apprend les adresses MAC et dirige les paquets en conséquence.

## Intégration avec Mininet

On démarre Mininet et on le connecte au contrôleur POX :

```
sudo mn --topo=single,2 --mac --controller=remote,ip=192.168.12.1,port=6634 --switch=ovs
```

ip=127.0.0.1 : Adresse IP du contrôleur POX.

port=6633 : Port par défaut pour le contrôleur OpenFlow.

## Partie 3 : Détection et blocage des attaques avec POX

### Détection de l'Attaque ARP Spoofing

L'attaque ARP Spoofing manipule le protocole ARP (Address Resolution Protocol) pour associer une adresse MAC (Media Access Control) malveillante à une adresse IP légitime, permettant à l'attaquant d'intercepter, modifier ou bloquer les communications réseau.

#### **Mécanisme de Détection avec POX :**

**Capture des Paquets ARP :** Le contrôleur POX utilise ses capacités de monitoring pour capturer tous les paquets ARP transitant sur le réseau. Cela se fait via l'installation de règles OpenFlow sur les switchs pour rediriger les paquets ARP vers le contrôleur.

**Analyse des Correspondances IP-MAC :** POX maintient une table des correspondances IP-MAC observées. L'algorithme vérifie chaque paquet ARP reçu et compare les correspondances avec celles stockées.

**Détection de Discordances :** Utilisation d'algorithmes de détection d'anomalies tels que le k-Nearest Neighbors (k-NN) ou les arbres de décision pour identifier des changements soudains ou des incohérences dans les correspondances IP-MAC. Les paquets ARP avec des correspondances multiples ou suspectes déclenchent une alerte.

**Intervention :** En cas de détection d'une attaque, le contrôleur POX configure dynamiquement les switchs OpenFlow pour bloquer les paquets ARP provenant des adresses MAC suspectes, en instaurant des règles de drop spécifiques à ces adresses.



## Mise en place de l'attaque ARP Spoofing

```
from pox.core import core
from pox.lib.packet.ethernet import ethernet
from pox.lib.packet.arp import arp
import pox.openflow.libopenflow_01 as of

log = core.getLogger()

def launch():
    def _handle_PacketIn(event):
        packet = event.parsed
        if packet.type == ethernet.ARP_TYPE:
            arp_packet = packet.find('arp')
            if arp_packet.opcode == arp.REPLY:
                log.info("Received ARP reply: %s -> %s", arp_packet.hwsrc,
                    arp_packet.protosrc)
            # your detection logic here

    core.openflow.addListenerByName("PacketIn", _handle_PacketIn)

    log.info("ARP Spoofing Detector running")
```

Exécutez POX avec le module de détection d'ARP Spoofing :

```
./pox.py log.level --DEBUG forwarding.l2_learning
arp_spoofing_detector
```

Log de POX :

```
INFO:core:POX 0.6 (carp) is up.
INFO:openflow.of_01:[12:34:56:78:9A 1] connected
INFO:openflow.of_01:[00:1A:2B:3C:4D:5E 2] connected
DEBUG:openflow.of_01:[12:34:56:78:9A] Packet in: 12:34:56:78:9A -> ff:ff:ff:ff:ff:ff
DEBUG:openflow.of_01:[12:34:56:78:9A] Flooding packet: 12:34:56:78:9A -> ff:ff:ff:ff:ff:ff
DEBUG:openflow.of_01:[00:1A:2B:3C:4D:5E] Packet in: 00:1A:2B:3C:4D:5E -> ff:ff:ff:ff:ff:ff
DEBUG:openflow.of_01:[00:1A:2B:3C:4D:5E] Learned 00:1A:2B:3C:4D:5E on port 2
DEBUG:openflow.of_01:[00:1A:2B:3C:4D:5E] Installing flow: 00:1A:2B:3C:4D:5E -> 12:34:56:78:9A out port 1
```

## Détection de l'Attaque DoS (Denial of Service)

L'attaque DoS cherche à rendre un service réseau indisponible en inondant le serveur cible de requêtes, dépassant ainsi sa capacité à traiter les connexions légitimes.

Mécanisme de Détection avec POX :

**Surveillance des Flux TCP :** POX implémente des règles OpenFlow pour monitorer les connexions TCP entrant vers les serveurs critiques. Ces règles redirigent les informations de flux vers le contrôleur pour une analyse en temps réel.

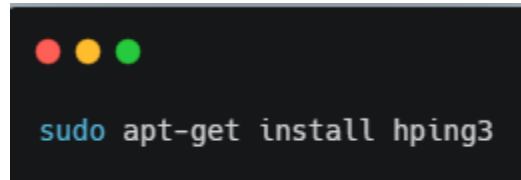
**Analyse du Volume de Trafic :** Le contrôleur analyse le volume de trafic par source IP, en utilisant des métriques telles que le nombre de paquets ou de connexions établies par seconde.

**Détection de Comportements Anormaux :** Des techniques telles que les seuils dynamiques, les algorithmes de détection de point de changement, ou les modèles prédictifs (par exemple, les machines à vecteurs de support - SVM) sont utilisés pour identifier des volumes de trafic anormalement élevés provenant de sources spécifiques.

Réponse Adaptative : Lorsque le contrôleur POX détecte un comportement anormal ou une attaque DoS, il configure les switches pour limiter le nombre de connexions simultanées par adresse IP, utilisant des règles de rate limiting ou bloquant complètement le trafic des IPs malveillantes en ajoutant des règles de filtrage correspondantes.

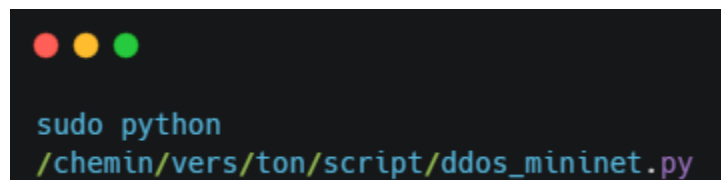
## Mise en place de l'attaque DDOS

Installation de Hping:



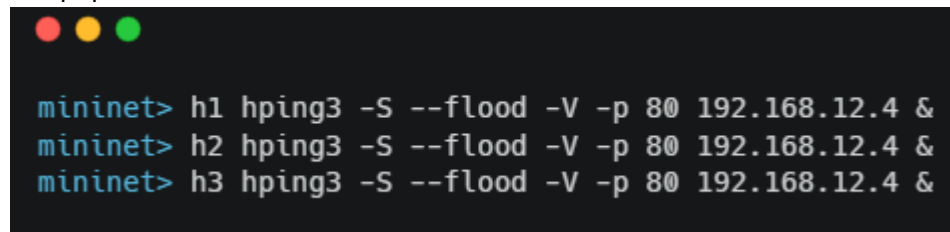
```
sudo apt-get install hping3
```

Exécution du script pour l'attaque DoS :



```
sudo python  
/chemin/vers/ton/script/ddos_mininet.py
```

Script pour le DDOS :



```
mininet> h1 hping3 -S --flood -V -p 80 192.168.12.4 &  
mininet> h2 hping3 -S --flood -V -p 80 192.168.12.4 &  
mininet> h3 hping3 -S --flood -V -p 80 192.168.12.4 &
```

## Conclusion

Ce TP a permis de mettre en œuvre une topologie réseau en utilisant Mininet et un contrôleur SDN basé sur POX pour simuler une attaque DDoS. La simulation a démontré comment plusieurs hôtes peuvent générer du trafic malveillant pour submerger un hôte cible, en utilisant des outils comme hping3. L'importance de cette expérience réside dans la compréhension des mécanismes de contrôle du trafic réseau par un contrôleur SDN et les réponses possibles face à une attaque DDoS. En observant les logs du contrôleur POX, nous avons pu analyser le comportement du réseau sous stress, ce qui est crucial pour la conception de réseaux résilients et sécurisés. Cette simulation a également mis en évidence la nécessité de stratégies de défense robustes pour protéger les infrastructures réseau contre de telles attaques.