

Equipe 1 :

DAMONNEVILLE Nicolas

PLEDEL Laurent


SZAJA Maxime

Cursus Réseau

Diplôme d'ingénieur - EICNAM

Projet RSX217 : IDS + SDN dans ODL

Ecole d'Ingénieur du CNAM

Rev :	Date :	Désignation :				
0	21/01/2021	Document initial				
1	04/02/2021	Document modifié				
2	04/02/2021	Version validée				
Etabli par :					Unité d'enseignement	
		N. Damonville	Le :	21/01/2021	RSX217	
		L. Pledel	Le :	04/02/2021	Nb pages	Révision
		M. Szaja	Le :	03/02/2021	30	2



Introduction

Dans le cadre du Cours Réseau de l'Ecole d'Ingénieur du Conservatoire National des Arts et Métiers, notre équipe se doit de valider l'unité d'enseignement RSX217.

Cette UE nécessite la réalisation d'un projet choisi parmi une liste de projets et peut s'effectuer seul ou en équipe.

Notre équipe, composée de Mr. PLEDEL Laurent, Mr. SZAJA Maxime et Mr. DAMONNEVILLE Nicolas a choisi comme projet le sujet numéro 1 intitulé « IDS + SDN dans ODL ».

A travers ce document, vous pourrez trouver toutes les informations nécessaires à la compréhension de notre projet, de l'analyse fonctionnelle à la réalisation du projet sans oublier le détail des technologies utilisées.

Vous pourrez ainsi comprendre quelles ont été les étapes de notre projet, pourquoi nous avons fait certains choix technologiques, comment fonctionne le projet ou encore, comment refaire le projet.

En effet, vous trouverez tous les scripts, documents, détails et explications du projet afin de pouvoir le recréer grâce à l'aide de ce support.

Une archive contenant tout le nécessaire pour reproduire ce projet, ainsi qu'une vidéo de démonstration, accompagnent ce rapport.



Table des matières

Introduction	2
1\ Description du projet.....	5
1.1\ Technologies	5
1.1.1\ Snort.....	5
1.1.2\ OpenVSwitch	6
1.1.3\ OpenDayLight.....	6
1.1.4\ Docker.....	7
1.2\ Environnement fonctionnel.....	7
1.3\ Environnement technique	9
1.3\ Spécifications logiciel	11
1.4\ Chantiers	12
2.\ Réalisation du projet	13
2.1\ Planification	13
2.2\ Les différents scripts	13
2.2.1\ Nettoyage de la configuration	13
2.2.2\ Création de l'architecture	14
2.2.3\ Initialisation des flows du switch	14
2.2.4\ Envoi d'un flow vers ODL.....	14
2.2.5\ DROP d'une IP depuis Snort vers ODL	15
2.3\ Snort	15
2.4\ ODL	16
2.4.1\ Flows de commutation des paquets IP	16
2.4.2\ Flows de commutation des paquets ARP	17
2.4.3\ Flow de DROP d'une IP.....	18
3\ Mise en production	18
4\ Conclusion	23
Annexes	24

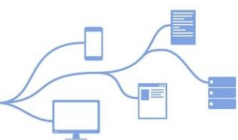


Table des tableaux

Tableau I - Technologies utilisées	7
Tableau II - Equipements de la topologie.....	8
Tableau III - Configuration réseau	10
Tableau IV - Spécifications logicielles	11
Tableau V - Chantiers	12

Table des figures

Figure 1 - Topologie fonctionnelle.....	8
Figure 2 - Topologie technique	9
Figure 3 - Diagramme de GANTT.....	13
Figure 4 - Création de l'OVS et des conteneurs	18
Figure 5 - Echo positionnement des scripts et règle DROP de Snort	18
Figure 6 – Echo création des interfaces et interfaçage à l'OVS	18
Figure 7 - Echo déconnexion des interfaces eth0	19
Figure 8 - Echo démarrage de karaf	19
Figure 9 - Echo installation plugin RESTCONF	19
Figure 10 – résultat de la configuration Openflow du switch.....	19
Figure 11 - Erreurs de connexion volontaires	21
Figure 12 - Bannissement du poste A	22
Figure 13 - Connexion réussie en FTP au serveur web.....	22
Figure 14 - Bannissement du poste A dans la table de flows	22



1\ Description du projet

Le Projet « IDS + SDN dans ODL » nous demande de pouvoir observer une attaque qui se fait détecter par un IDS ; ce dernier devra instruire le contrôleur afin de mettre à jour les tables de flux des commutateurs du réseau de manière à bloquer l'attaque.

Pour arriver à ce but, nous aurons besoin de différentes technologies, certaines sont imposées par le sujet, comme par exemple ODL, d'autres sont libres, comme par exemple le choix de l'IDS.

Afin de bien comprendre le projet, un point semble nécessaire sur les technologies utilisées.

1.1\ Technologies

1.1.1\ Snort

Snort est un IDS (Intrusion Détection System) publié sous licence GNU GPL. Il permet d'effectuer toutes sortes de gestion de trafic, nous permettant ainsi de savoir ce qu'il se passe sur notre réseau.

On peut le qualifier de « Sniffer » comme par exemple WireShark, c'est d'ailleurs pour cette raison que son créateur, Marty Roesch l'a appelé Snort, qui signifie « Renifler » en Français.

Ce dernier peut également effectuer une analyse du trafic en temps réel et enregistrer des paquets de données sur des réseaux IP. Il peut effectuer une analyse de protocole, une recherche / correspondance de contenu et peut être utilisé pour détecter diverses attaques et détections, telles que le débordement de tampon, l'analyse, les attaques sur CGI, la détection SMB, les tests d'empreintes digitales du système d'exploitation et bien plus encore.

Snort effectue une analyse basée sur des règles. Celles-ci sont écrites par Sourcefire ou écrites par la communauté. Il est livré avec quelques règles de base, mais il est possible de créer ses propres règles ou d'en télécharger des nouvelles sur le site de la communauté.

Du fait de sa facilité d'installation, Snort sera choisi comme IDS pour notre projet.



1.1.2\ OpenVSwitch

OpenVSwitch est une implémentation logicielle d'un commutateur réseau virtuel multicouche. Il est conçu pour permettre une automatisation efficace du réseau via des extensions programmatiques ; ce dernier prend en charge les interfaces des switchs et les protocoles de gestion tels que NetFlow, sFlow, SPAN, RSPAN, LACP, CLI et 802.1ag.

De plus, OpenVSwitch est conçu pour prendre en charge une distribution transparente sur plusieurs serveurs physiques en permettant la création de commutateurs inter-serveurs.

Il peut fonctionner comme un commutateur de réseau logiciel en s'exécutant dans un hyperviseur de VM, mais aussi comme contrôleur pour le matériel de commutation.

Nous utilisons cette technologie afin de créer un commutateur virtuel pour notre réseau qui nous permettra de relier nos différents équipements informatiques dans le but d'obtenir un réseau commuté virtuel.

1.1.3\ OpenDayLight

OpenDayLight (ODL) est une plateforme modulaire pour l'automatisation et la personnalisation de réseaux de différentes échelles. ODL est né du mouvement SDN et de la programmabilité du réseau. ODL est conçu pour offrir une flexibilité maximale afin de créer un contrôleur réseau adapté à la situation. Du fait de sa conception modulaire, ODL permet à n'importe qui de tirer parti des services créés par d'autres utilisateurs ou encore d'en créer soi-même et de les partager avec d'autres personnes. ODL prend en charge un large choix de protocoles, comme OpenFlow, NETCONF ou encore BGP, ce qui permet une facilité de programmabilité des réseaux modernes.

Pour notre projet, ODL nous servira de contrôleur SDN, il s'occupera de mettre à jour la topologie réseau en fonction des différentes alarmes que Snort lui enverra. Ainsi, lors de la détection d'une attaque, le contrôleur ODL pilotera le switch OpenVSwitch afin de changer la topologie réseau et, en résultante, contrer l'attaque.



1.1.4\ Docker

Docker est un logiciel permettant de lancer des applications dans des conteneurs logiciels.

L'utilité d'un conteneur est de pouvoir héberger des services sur un même serveur physique tout en isolant les conteneurs les uns des autres. La différence avec une machine virtuelle est qu'un conteneur reste moins figé qu'une machine virtuelle en termes de taille de disques et de ressources allouées.

Les conteneurs permettent l'isolation de chaque service, que ça soit une application, une base de données ou encore un serveur web, ces derniers peuvent être exécutés de façon indépendante dans leur conteneur dédié.

Il est possible de connecter les conteneurs à un réseau virtuel, permettant ainsi la communication réseau.

Utiliser Docker permet la simplification de la mise en œuvre de systèmes distribués en permettant à de multiples applications et autres processus de s'exécuter de façon autonome dans un conteneur.

Pour notre projet, l'utilisation de la technologie Docker nous permettra d'isoler nos applicatifs car l'ensemble du projet se fera sur une seule VM. Il sera alors possible de relier les conteneurs à notre réseau virtuel, permettant ainsi une commutation entre nos conteneurs tout en conservant l'isolement applicatif de ces derniers.

1.2\ Environnement fonctionnel

Pour rappel, nous avons comme technologies utilisées :

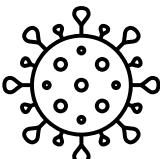

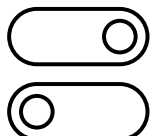
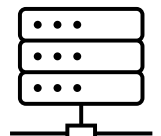
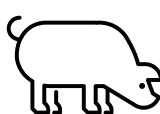
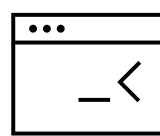
Tableau I - Technologies utilisées

Docker	Snort	ODL	OpenVSwitch
Permet la création de conteneurs pour isoler nos applicatifs	Notre IDS nous permettant d'analyser le trafic.	Notre contrôleur SDN qui nous permet la modification du trafic.	Simule un switch virtuel et permet l'interconnectivité

Le sujet étant volontairement très ouvert, une infinité de topologies s'offrent à nous ; nous avons choisi de réduire au maximum la topologie dans le but de faciliter la compréhension du projet.



Tableau II - Equipements de la topologie

Poste contaminé	Poste témoin	Commutateur	Serveur cible	Snort	Contrôleur SDN
					

Tous les équipements seront reliés au commutateur, le poste contaminé sera le poste que Snort devra détecter et interrompre.

Le poste témoin est présent afin de confirmer la continuité de service pour les autres utilisateurs, cela permet de montrer que seul l'attaquant sera déconnecté du réseau.

Le contrôleur SDN mettra à jour la topologie.

Avec tous ces éléments, nous avons proposé la topologie suivante :

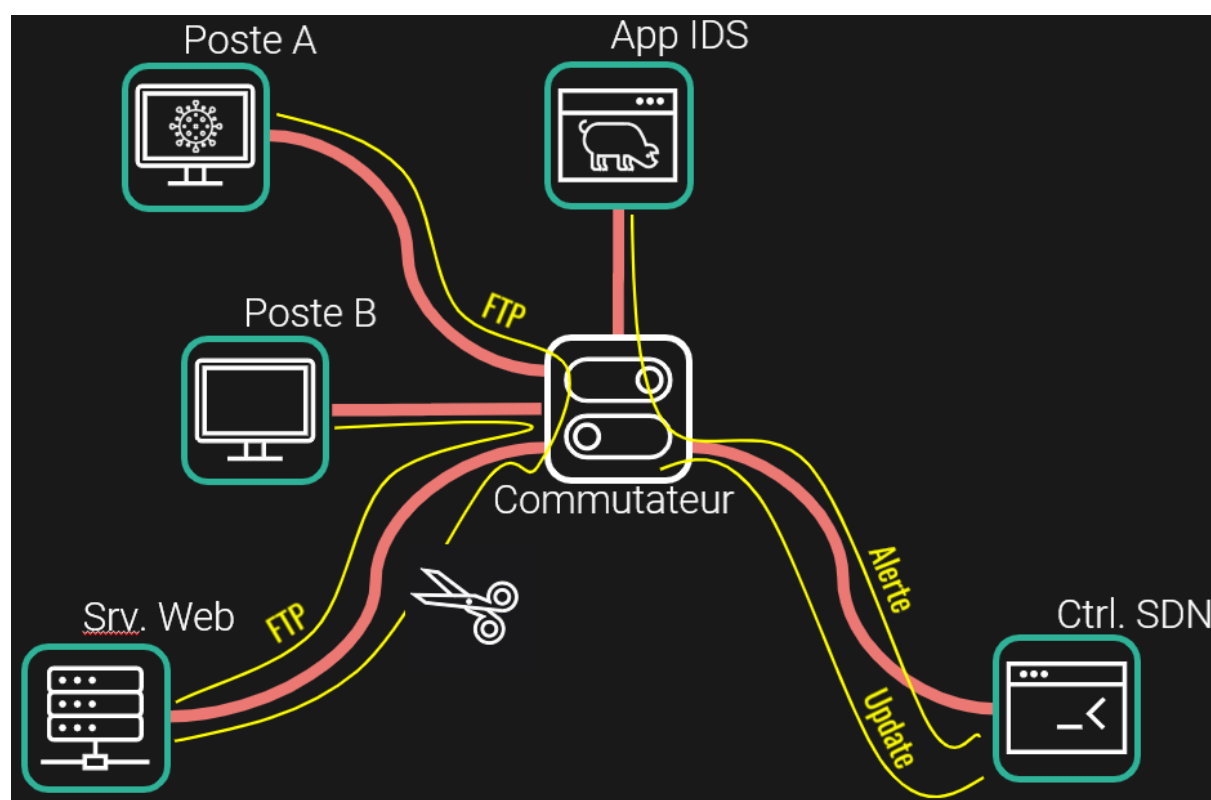


Figure 1 - Topologie fonctionnelle



Le scénario est le suivant :

Le poste A est contaminé, il tente d'accéder au serveur web, Snort détecte la fraude et informe le contrôleur SDN, ce dernier coupe le lien du poste A. Le poste B quant à lui, n'est pas impacté par la mise à jour de la topologie.

Cette topologie à l'avantage d'être très simple, cependant, elle soulève une question : comment Snort peut-il analyser le trafic. En effet, nous sommes en commutation de niveau 2, les paquets du poste A vont directement sur le serveur web, ils ne passent pas par Snort. Ce dernier ne peut donc pas analyser les paquets et détecter l'attaquant.

Afin de palier à ce problème, deux choix s'offrent à nous :

Nous pouvons rajouter un pare-feu et installer Snort dessus puis mettre les postes dans un VLAN 10 et le serveur dans un VLAN 20. Cette modification oblige les paquets des postes à venir se router sur le pare-feu où est installé Snort, il pourra donc analyser le trafic sans problème

L'autre solution serait de dire au commutateur de rediriger tout le trafic vers Snort afin qu'il puisse l'analyser.

Nous avons opté pour cette dernière solution qui aboutit à une topologie plus simple.

1.3\ Environnement technique

Maintenant que nous avons un environnement fonctionnel, nous pouvons passer à un environnement technique.

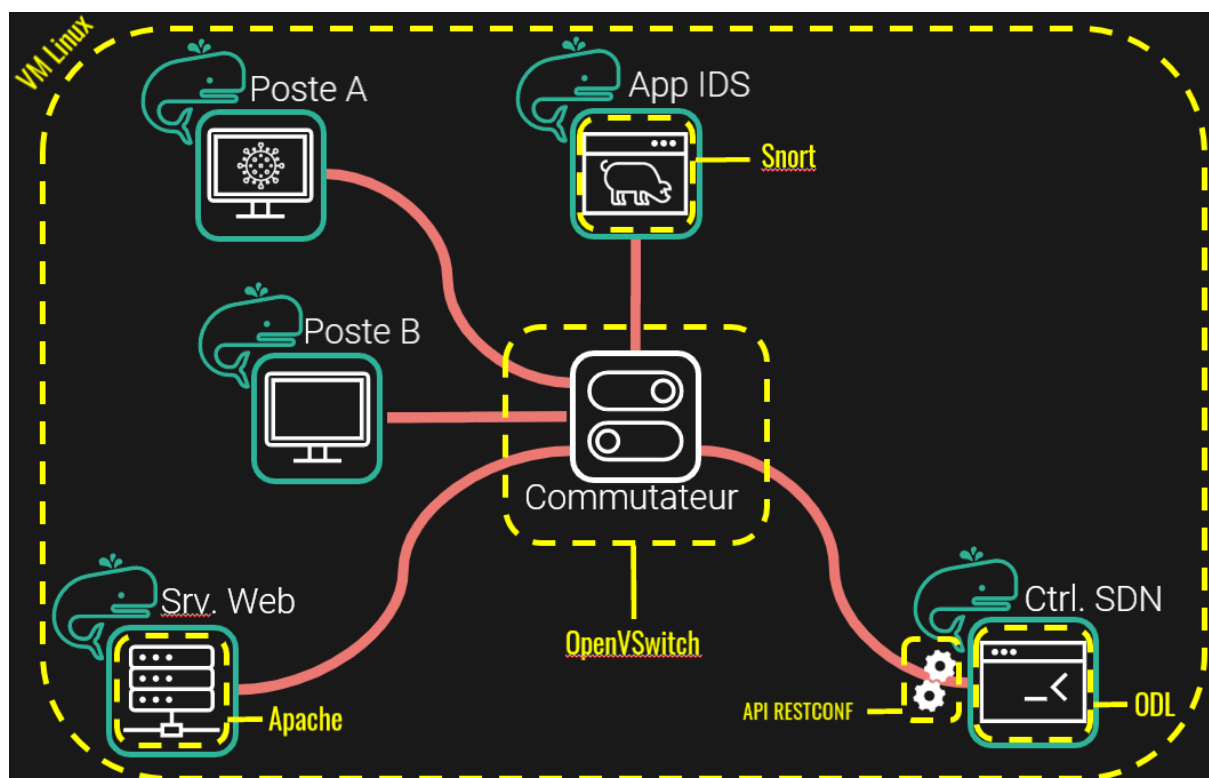


Figure 2 - Topologie technique



Host	IP	MAC	Interface
Poste A	192.168.16.2	10:00:00:00:00:02	Eth1
Poste B	192.168.16.1	10:00:00:00:00:01	Eth1
IDS Snort	192.168.16.3	10:00:00:00:00:03	Eth1
Serveur Web (FTP)	192.168.16.4	10:00:00:00:00:04	Eth1
Contrôleur SDN (ODL)	192.168.16.5	10:00:00:00:00:05	Eth1
	172.17.0.2	-	Eth0

Tableau III - Configuration réseau

Nous avons choisi d'héberger l'ensemble du projet sur une VM Linux, les applicatifs seront installés dans des dockers afin d'assurer l'isolement entre eux.

Le serveur sera un serveur apache et la commutation se fait via OpenVSwitch comme indiqué précédemment.

Le contrôleur SDN pilote la topologie réseau via OpenFlow ; quant à Snort, il pilote le contrôleur SDN via une API RESTCONF.

Snort devra détecter plusieurs tentatives de connexions échouées en FTP au serveur Apache par le Poste A, par suite de cela, il ajoutera une règle demandant la perte de connexion du poste A au réseau.

Le contrôleur SDN, via OpenFlow, mettra à jour le commutateur OpenVSwitch qui coupera le lien du poste A au reste de la topologie.

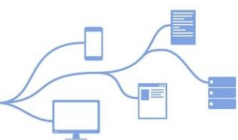


1.3\ Spécifications logiciel

Voici les différentes versions logicielles ainsi que les images docker utilisées.

Tableau IV - Spécifications logicielles

Logiciel	Version
CentOS (Host)	8.0
Docker	20.10.2 (Host)
OpenVSwitch	2.12.0 (Host)
Python	3.6.8 (Host) et 3.4.3 (IDS)
Module Python « request »	2.20.0 (Host) et 2.25.1 (IDS)
OpenDayLight docker image	6.1.0 (opendaylight/odl)
Applications ODL	odl-openflowplugin-flow-services-rest 0.4.1-carbon
	odl-restconf-1.5.1-carbon
Snort docker image	2.9.8.0 (linton/docker-snort)
Poste client A et B docker image	Latest net-tools (for ping, ifconfig, etc)
PureFTP docker image	Latest (panubo/vsftpd)



1.4\ Chantiers

En nous basant sur nos schémas précédents, nous avons imaginé 3 chantiers qui seront 3 grande étapes du projet.

Tableau V - Chantiers

Chantiers	Tâche	Difficulté
ODL	Configuration initiale du switch à travers le GUI ODL	4/5
	Mettre à jour une table de flow via API RESTCONF	4/5
Snort	Déclenchement d'une commande / script permettant de couper le flow via le contrôleur	3/5
	Implémentation des règles lors de la détection d'évènement	2/5
Docker / OpenVSwitch	Interconnecter les containers au sein d'un réseau OpenVSwitch	3/5
	Utilisation de dockerfiles	1/5

Les indices de difficultés ont été choisis par rapport à la connaissance du groupe dans le domaine concerné et par la présence ou non de documentation sur le sujet.

L'API RESTCONF et le GUI ODL n'étant que peu documentés, ils possèdent des indices de difficulté élevés.

La définition des chantiers ayant été faite, nous avons pu passer à la réalisation du projet.

2.1\ Planification

Le diagramme de Gantt pour la semaine 10 (du 24 décembre au 31 décembre 2020) est structuré comme suit :

- Lancement** (1 jour, S0 J0)
- Analyse/modélisation** (19 jours, S0 J0 à S1 J6)
 - Étudier le besoin exprimé (3 jours, S0 J0 à S0 J3)
 - Lister les ressources requises (3 jours, S0 J3 à S0 J6)
 - S'approprier/compléter le modèle proposé (3 jours, S0 J6 à S1 J0)
 - Étudier IDS Snort (10 jours, S1 J0 à S1 J6)
 - Étudier Controller OD (10 jours, S1 J0 à S1 J6)
 - Étudier OpenSwitch (10 jours, S1 J0 à S1 J6)
- Réalisation** (21 jours, S1 J6 à S2 J6)
 - Création de la VM Linux-Hôte (1 jour, S1 J6 à S1 J7)
 - Installation de Snort/OD/ OpenSwitch (Docker) (4 jours, S1 J7 à S1 J0)
 - Création de la topologie réseau (5 jours, S1 J0 à S1 J5)
 - Configuration Snort (5 jours, S1 J5 à S1 J0)
 - Configuration OD (5 jours, S1 J0 à S1 J5)
 - Configuration OpenSwitch (5 jours, S1 J5 à S1 J0)
 - Interface OD/Snort (8 jours, S1 J0 à S1 J7)
 - Test de bon fonctionnement (2 jours, S1 J7 à S1 J0)
 - Vérifier/Finaliser la configuration (1 jour, S1 J0 à S1 J1)
- Intégration/Mise en place** (8 jours, S2 J0 à S2 J7)
 - Mettre en place un environnement fonctionnel (12 jours, S2 J0 à S2 J6)
 - Simulation attaque FTP (6 jours, S2 J6 à S2 J0)
- Documentation** (40 jours, S2 J0 à S3 J6)
 - Rédiger la présentation (40 jours, S2 J0 à S3 J6)
 - Fin du projet (1 jour, S3 J6 à S3 J7)

Figure 3 - Diagramme de GANTT

La mise en place de la configuration est totalement automatisée. L'ensemble des scripts et données de configuration seront livrés dans une archive « miniprojetv3.7z » Une fois décompressée on obtient deux sous dossiers.

Le dossier flow qui contient la bibliothèque de flow.

Un script batch enchaîne l'ensemble des commandes permettant de créer l'architecture présentée dans le chapitre 1.3. Le script s'appuie sur un programme python pour injecter les flows dans le switch ».

Ces différents scripts et programmes sont décrit dans les parties suivantes du document.

Le script Cleanminiprojet.sh effectue une remise à zéro de la topologie OVS et Docker

Voici la commande sudo à exécuter depuis le répertoire du script :

```
sudo ./cleanminiprojet.sh
```



2.2.2\ Création de l'architecture

Le script `createminiprojet.sh` effectue une remise à zéro puis crée la topologie complète.

D'une part, il initialise le switch OVS « miniprojet » ainsi que le réseau, puis crée les containers Dockers et attache les containers au réseau.

Ensuite, il active le protocole `open_flow` sur le switch, puis gère la connexion au docker du contrôleur ODL, puis lance ODL et les applications requises (API rest)

Enfin, il effectue la connexion au contrôleur SDN et la configuration initiale des flows via le programme python « `initswitchb.py` »

Voici la commande `sudo` à exécuter depuis le répertoire du script :

```
sudo ./createminiprojet.sh
```

2.2.3\ Initialisation des flows du switch

2.2.4\ Envoi d'un flow vers ODL

Ce programme python est utilisé dans « `createminiprojet.sh` » pour envoyer les flows au contrôleur.

Le script récupère l'id openflow par une commande REST http « GET » sur le port du contrôleur dont l'adresse est indiquée dans le paramètre de commande de lancement du programme.

Le programme injecte les flows via l'envoi de requêtes http « PUT » sur le port 8181 du contrôleur. Le body de la requête contient les commandes de programmation de flows au format « .json ».

Les flows à envoyer sont passés en paramètre de la commande, ces paramètres correspondent à l'id des fichiers flows au format « json » dont le contenu sera envoyé dans le header de la requête. Ces fichiers sont situés dans la bibliothèque des flows dans le dossier « flow ». L'utilité des différents flows est décrite dans la suite de ce document.

Voici la nomenclature de la commande à effectuer :

```
sudo python3 ./initswitchb.py <id_fichier_flow1> <id_fichier_flow2> <id_fichier_flow>
```

<id_fichier_flow> correspond à l'id du fichier `fl<id>.json` dans le répertoire flow.

Voici un exemple de commande `sudo` à exécuter depuis le répertoire du script :

```
sudo python3 ./initswitchb.py 100 200 300 400 500 101 202 303 404 505
```



2.2.5\ DROP d'une IP depuis Snort vers ODL

Le script « ftpthreat.sh » est le script qui gère le parsing des logs de Snort, il prend son dernier fichier de log, puis ses 30 dernières lignes.

Ensuite, il récupère l'IP à bannir s'il y'a plus de 3 tentatives de connexion ratées et envoie la requête vers l'API d'ODL qui effectuera un drop et bannira l'IP pour 1 minute.

Voici la commande sudo à exécuter depuis le répertoire du script :

```
sudo ./ftpthreat.sh
```

L'envoi de la commande drop s'appuie sur le programme python « putflow.py ». Il fonctionne selon le même principe que le programme initswitchb.py décrit plus haut. L'adresse IP du contrôleur et le fichier flow à envoyer est passé en paramètre.

Voici la nomenclature de la commande à effectuer :

```
sudo python3 ./putflow.py <ip controleur> <nom complet du fichier flow>
```

Voici un exemple de commande sudo à exécuter depuis le répertoire du script :

```
sudo python3 ./putflow.py 172.17.0.2 /home/student/SDN/miniprojetfile/flow/f11402.json
```

2.3\ Snort

```
alert tcp $HOME_NET 21 -> $EXTERNAL_NET any (msg:"FTP Brute-Force attempt";  
flow:from_server,established; content:"530 "; sid:1000001; rev:10;)
```

C'est la règle de Snort permettant la détection d'une erreur de connexion FTP (erreur 530). Cette règle sera livrée dans un fichier texte « snortruleftp.txt » qui servira au script bash pour l'écrire automatiquement dans la configuration de Snort dans le container.



2.4\ ODL

2.4.1\ Flows de commutation des paquets IP

```
{ "flow-node-inventory:flow": [{"id": "101", "priority": 10, "table_id": 0, "cookie_mask": 0, "hard-timeout": 0, "match": {"ipv4-destination": "192.168.16.1/32", "ethernet-match": {"ethernet-type": {"type": 2048}}}, "cookie": 101, "flags": "", "instructions": {"instruction": [{"order": 0, "apply-actions": {"action": [{"order": 0, "output-action": {"max-length": 65535, "output-node-connector": "1"}}, {"order": 1, "output-action": {"max-length": 65535, "output-node-connector": "3"}} ]}], "idle-timeout": 0}] }
```

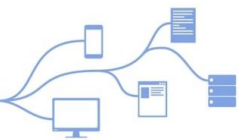
```
{ "flow-node-inventory:flow": [{"id": "202", "priority": 10, "table_id": 0, "cookie_mask": 0, "hard-timeout": 0, "match": {"ipv4-destination": "192.168.16.2/32", "ethernet-match": {"ethernet-type": {"type": 2048}}}, "cookie": 202, "flags": "", "instructions": {"instruction": [{"order": 0, "apply-actions": {"action": [{"order": 0, "output-action": {"max-length": 65535, "output-node-connector": "2"}}, {"order": 1, "output-action": {"max-length": 65535, "output-node-connector": "3"}} ]}], "idle-timeout": 0}] }
```

```
{ "flow-node-inventory:flow": [{"id": "303", "priority": 10, "table_id": 0, "cookie_mask": 0, "hard-timeout": 0, "match": {"ipv4-destination": "192.168.16.3/32", "ethernet-match": {"ethernet-type": {"type": 2048}}}, "cookie": 303, "flags": "", "instructions": {"instruction": [{"order": 0, "apply-actions": {"action": [{"order": 0, "output-action": {"max-length": 65535, "output-node-connector": "3"}}, {"order": 1, "output-action": {"max-length": 65535, "output-node-connector": "3"}} ]}], "idle-timeout": 0}] }
```

```
{ "flow-node-inventory:flow": [{"id": "404", "priority": 10, "table_id": 0, "cookie_mask": 0, "hard-timeout": 0, "match": {"ipv4-destination": "192.168.16.4/32", "ethernet-match": {"ethernet-type": {"type": 2048}}}, "cookie": 404, "flags": "", "instructions": {"instruction": [{"order": 0, "apply-actions": {"action": [{"order": 0, "output-action": {"max-length": 65535, "output-node-connector": "4"}}, {"order": 1, "output-action": {"max-length": 65535, "output-node-connector": "3"}} ]}], "idle-timeout": 0}] }
```

```
{ "flow-node-inventory:flow": [{"id": "505", "priority": 10, "table_id": 0, "cookie_mask": 0, "hard-timeout": 0, "match": {"ipv4-destination": "192.168.16.5/32", "ethernet-match": {"ethernet-type": {"type": 2048}}}, "cookie": 505, "flags": "", "instructions": {"instruction": [{"order": 0, "apply-actions": {"action": [{"order": 0, "output-action": {"max-length": 65535, "output-node-connector": "5"}} ]}], "idle-timeout": 0}] }
```

Ces flows permettent de programmer le switch de façon à commuter l'adresse IP contenue dans une trame Ethernet type IPv4 vers où est connecté la machine de destination. Chaque trame est dupliquée vers la sonde SNORT afin qu'elle puisse analyser le trafic. Ces flows ont une priorité supérieure aux flows ARP, mais une priorité inférieure aux flow DROP envoyé par la sonde SNORT.



2.4.2\ Flows de commutation des paquets ARP

```
{"flow-node-inventory:flow":[{"id": "100", "priority": 2, "table_id": 0, "cookie_mask": 0, "hard-timeout": 0, "match": {"in-port": "1"}, "cookie": 100, "flags": "", "instructions": {"instruction": [{"order": 0, "apply-actions": {"action": [{"order": 2, "output-action": {"max-length": 65535, "output-node-connector": "2"}}, {"order": 1, "output-action": {"max-length": 65535, "output-node-connector": "4"}}, {"order": 0, "output-action": {"max-length": 65535, "output-node-connector": "3"}}]}]}], "idle-timeout": 0}]}
```

```
{"flow-node-inventory:flow":[{"id": "200", "priority": 2, "table_id": 0, "cookie_mask": 0, "hard-timeout": 0, "match": {"in-port": "2"}, "cookie": 200, "flags": "", "instructions": {"instruction": [{"order": 0, "apply-actions": {"action": [{"order": 2, "output-action": {"max-length": 65535, "output-node-connector": "1"}}, {"order": 1, "output-action": {"max-length": 65535, "output-node-connector": "4"}}, {"order": 0, "output-action": {"max-length": 65535, "output-node-connector": "3"}}]}]}], "idle-timeout": 0}]}
```

```
{"flow-node-inventory:flow":[{"id": "300", "priority": 2, "table_id": 0, "cookie_mask": 0, "hard-timeout": 0, "match": {"in-port": "3"}, "cookie": 300, "flags": "", "instructions": {"instruction": [{"order": 0, "apply-actions": {"action": [{"order": 2, "output-action": {"max-length": 65535, "output-node-connector": "1"}}, {"order": 1, "output-action": {"max-length": 65535, "output-node-connector": "4"}}, {"order": 3, "output-action": {"max-length": 65535, "output-node-connector": "5"}}, {"order": 0, "output-action": {"max-length": 65535, "output-node-connector": "2"}}]}]}], "idle-timeout": 0}]}
```

```
{"flow-node-inventory:flow":[{"id": "400", "priority": 2, "table_id": 0, "cookie_mask": 0, "hard-timeout": 0, "match": {"in-port": "4"}, "cookie": 400, "flags": "", "instructions": {"instruction": [{"order": 0, "apply-actions": {"action": [{"order": 2, "output-action": {"max-length": 65535, "output-node-connector": "1"}}, {"order": 1, "output-action": {"max-length": 65535, "output-node-connector": "3"}}, {"order": 0, "output-action": {"max-length": 65535, "output-node-connector": "2"}}]}]}], "idle-timeout": 0}]}
```

```
{"flow-node-inventory:flow":[{"id": "500", "priority": 2, "table_id": 0, "cookie_mask": 0, "hard-timeout": 0, "match": {"in-port": "5"}, "cookie": 500, "flags": "", "instructions": {"instruction": [{"order": 0, "apply-actions": {"action": [{"order": 0, "output-action": {"max-length": 65535, "output-node-connector": "3"}}]}]}], "idle-timeout": 0}]}
```

L'utilisation de ces flows permettent la programmation du switch de telle manière que l'ensemble du trafic entrant soit envoyé vers tous les ports du switch. En effet, cette méthode permet aux différents applicatifs d'initialiser leurs tables ARP. Lorsque le trafic IP est initialisé, ces flows ne « match » plus car ils ont une priorité inférieure.



2.4.3\ Flow de DROP d'une IP

```
{"flow-node-inventory:flow": [{"id": "1402", "priority": 20, "table_id": 0, "cookie_mask": 0, "hard-timeout": 180, "match": {"ipv4-source": "192.168.16.2/32", "ethernet-match": {"ethernet-type": {"type": 2048}}}, "cookie": 1402, "flags": "", "instructions": {"instruction": [{"order": 0, "apply-actions": {"action": [{"order": 0, "drop-action": {}} ]}}}], "idle-timeout": 60}]}
```

Ce flow, au format JSON, envoie une commande de « DROP » au switch de façon qu'une trame Ethernet/IP qui possède comme adresse source l'adresse de l'attaque soit filtrée. Le flux a une durée de vie de 60 secondes après le dernier « match » et au plus, 3 minutes après sa création.

3\ Mise en production

Dans un premier temps, dans un premier terminal, on exécute la commande :

```
Sudo ./createmini projet.sh
```

Cette commande va lancer le script createmini projet.sh qui va créer le switch OpenVSwitch ainsi que les 5 conteneurs qui vont héberger nos applicatifs.

```
-e creation du switch ovs miniprojet
-e creation des 5 containers herbergeant les différents serveurs en mode detach
9f3277bb3a487d528c030dc9e0bac4238e7e9ec59ad16fd5bc2a9a2102847e2b
93a38d337b7eb62ed3c591cff44fb239ab65d0f41d331b3dd68cf61634d04af8
accb7d20ed91c0b254e8dbde25a4e3e4d3bae04e4af0c9e323876d8917b96bb
e9b1574f82443f0f6a7a1194d226eb0ab794da8ea9222855f35a51ba7f600503
8c0df367215189138058a29e759254a14e1f733e44c5516bbfd7890dalf891
CONTAINER ID      IMAGE               COMMAND             CREATED           STATUS              PORTS              NAMES
8c0df3672151      amouat/network-utls "bash"             Less than a second ago Up Less than a second 8080               web_server
e9b1574f8244      amouat/network-utls "bash"             1 second ago      Up Less than a second 8080               ids
accb7d20ed91      amouat/network-utls "bash"             2 seconds ago      Up 1 second         8080               postea
93a38d337b7e      amouat/network-utls "bash"             3 seconds ago      Up 2 seconds        8080               posteb
9f3277bb3a48      opendaylight/odl    "bash"             3 seconds ago      Up 3 seconds        8080               sdn_ctl
```

Figure 4 - Création de l'OVS et des conteneurs

L'exécution du script va mettre à jour les conteneurs et installer les paquets FTP et Nano. Le conteneur de Snort aura python 3 d'installé et le module « request » afin de pouvoir exécuter nos scripts python

Pour donner suite à cela, le script va mettre certains scripts dans l'IDS et ajouter la règle de DROP en cas d'erreur 530 dans Snort.

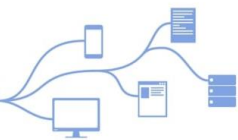
```
Positionnement des fichiers neccessaires et modification des regles
```

Figure 5 - Echo positionnement des scripts et règle DROP de Snort

Ensuite, le script va créer l'interface eth1 à chaque conteneur et les connecter aux interfaces du switch OpenVSwitch.

```
création d'un interface eth1 sur les containers + attachement de l'interface au port du switch
affectation des adresses ip sur le réseau 192.168.16.X/24 et adresses mac
```

Figure 6 – Echo création des interfaces et interfaçage à l'OVS



Par la même occasion, les interfaces eth0 sont désactivées sur tous les containers pour pas qu'ils n'aient accès à internet et permettre par la même occasion de les isoler complètement.

```
déconnexion de l'interface par défaut eth0 du bridge docker des containers
```

Figure 7 - Echo déconnexion des interfaces eth0

Ensuite, le script va démarrer notre contrôleur ODL Karaf :

```
demarrage du controleur ODL : karaf
```

Figure 8 - Echo démarrage de karaf

Ensuite, le plugin RESTCONF est installé :

```
demarrage du plugins openflow du controleur odl-restconf
```

Figure 9 - Echo installation plugin RESTCONF

Ensuite, le script active le protocole Openflow sur le switch et le connecte au contrôleur.

```
configuration openflow du switch
tempo de 10 secondes
verification du switch
bc314b9-a4a4-4ac6-bfc7-84a99ddfbad
  Bridge miniprojet
    Controller "tcp:172.17.0.2"
      is_connected: true
    Port "b2bb99b252d54_1"
      Interface "b2bb99b252d54_1"
    Port "07be86aa3e674_1"
      Interface "07be86aa3e674_1"
    Port "6e0f72812a264_1"
      Interface "6e0f72812a264_1"
    Port "3e4479f02b734_1"
      Interface "3e4479f02b734_1"
    Port miniprojet
      Interface miniprojet
        type: internal
    Port "ce0126cdd57f4_1"
      Interface "ce0126cdd57f4_1"
```

Figure 10 – résultat de la configuration Openflow du switch

Pour finir, le script configure les flows de base à l'aide d'un script, on aura donc les flows de base pour les paquets IP et ARP.



```
-e lancement de la configuration openflow du switch
-e lancement du script python initswitch.py dans le repertoire courant
-e ce script prend en argument les id des fichier flow json situé dans le repertoire
-e ../flow ces fichiers sont de la forme fl<id flow>.json
-e tempo de 10 secondes
identification de l'id openflow du switch et de l'id des ports '
'
<Response [200]> ok le controleur repond
id topologie: flow:1
id openflow du switch : 86684814587979
port du switch id 5
port du switch id 4
port du switch id LOCAL
port du switch id 1
port du switch id 3
port du switch id 2
lancement configuration initiale du switch :86684814587979
lancement du fichier flow:../flow/fl100.json
id flow:100
id cookie:0x 0x64
reponse controleur <Response [201]>

lancement du fichier flow:../flow/fl200.json
id flow:200
id cookie:0x 0xc8
reponse controleur <Response [201]>

lancement du fichier flow:../flow/fl300.json
id flow:300
id cookie:0x 0x12c
reponse controleur <Response [201]>

lancement du fichier flow:../flow/fl400.json
```

Figure 11 – Traces du programme python initialisant le contrôleur ODL (flows de commutation IP et ARP)

```
-e dumps des flow dans le switch
cookie=0x65, duration=9.895s, table=0, n_packets=0, n_bytes=0, priority=10,ip,nw_dst=192.168.16.1 actions=output:"f7797dddabcc4_l",output:"46ae7188e6394_l"
cookie=0xc8, duration=9.895s, table=0, n_packets=0, n_bytes=0, priority=10,ip,nw_dst=192.168.16.2 actions=output:"20c788479eb34_l",output:"46ae7188e6394_l"
cookie=0x12f, duration=9.895s, table=0, n_packets=0, n_bytes=0, priority=10,ip,nw_dst=192.168.16.3 actions=output:"46ae7188e6394_l",output:"46ae7188e6394_l"
cookie=0x194, duration=9.895s, table=0, n_packets=0, n_bytes=0, priority=10,ip,nw_dst=192.168.16.4 actions=output:"a849d5e3a4014_l",output:"46ae7188e6394_l"
cookie=0x1f9, duration=9.895s, table=0, n_packets=0, n_bytes=0, priority=10,ip,nw_dst=192.168.16.5 actions=output:"d14a52cb58c14_l"
cookie=0xc64, duration=9.895s, table=0, n_packets=0, n_bytes=0, priority=2,in_port="f7797dddabcc4_l" actions=output:"46ae7188e6394_l",output:"a849d5e3a4014_l",output:"f7797dddabcc4_l"
cookie=0xc8, duration=9.895s, table=0, n_packets=0, n_bytes=0, priority=2,in_port="20c788479eb34_l" actions=output:"46ae7188e6394_l",output:"a849d5e3a4014_l",output:"f7797dddabcc4_l"
cookie=0x12c, duration=9.895s, table=0, n_packets=0, n_bytes=0, priority=2,in_port="46ae7188e6394_l" actions=output:"20c788479eb34_l",output:"a849d5e3a4014_l",output:"f7797dddabcc4_l",output:"c14_l"
cookie=0x190, duration=9.895s, table=0, n_packets=0, n_bytes=0, priority=2,in_port="a849d5e3a4014_l" actions=output:"20c788479eb34_l",output:"46ae7188e6394_l",output:"f7797dddabcc4_l"
cookie=0x1f4, duration=9.895s, table=0, n_packets=0, n_bytes=0, priority=2,in_port="d14a52cb58c14_l" actions=output:"46ae7188e6394_l"
```

Figure 12 – Dump des flows du switch en fin de configuration

L'exécution du script createminiprojet.sh est terminée, nous allons maintenant mettre Snort en mode écoute à l'aide des commandes suivantes :

Dans un premier temps, on se place dans le dossier ftpthreat :

```
Cd /ftptthreat/
```

Puis on exécute Snort en mode écoute :

```
Sudo snort -A console -q -c /etc/snort/etc/snort.conf -i eth1 -k none
```



Une fois Snort lancé, on va ouvrir deux nouveaux terminaux qui seront les postes A et B afin de tester l'architecture. Il suffit d'exécuter ces deux commandes dans deux terminaux différents, elles nous permettent de nous connecter aux postes A et B situés dans les conteneurs Docker :

```
Sudo docker exec -it posteA bash
```

Et :

```
Sudo docker exec -it posteB bash
```

Sur le poste A, qui sera notre poste attaquant, on va volontairement faire des erreurs d'authentification en FTP afin de Snort détecte une tentative d'intrusion :

```
root@3ead2326c290:/# ftp 192.168.16.4
Connected to 192.168.16.4.
220 (vsFTPd 3.0.3)
Name (192.168.16.4:root): df
331 Please specify the password.
Password:
530 Login incorrect.
Login failed.
ftp> user
(username) s
331 Please specify the password.
Password:
530 Login incorrect.
Login failed.
ftp> user
(username) s
331 Please specify the password.
Password: _
```

Figure 11 - Erreurs de connexion volontaires

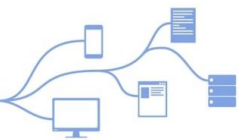
Dans le terminal où Snort est lancé, on peut observer les tentatives de connexion échouées du poste A qui est en 192.168.16.2

```
01/14-01:53:14.557027  [**] [1:1000001:10] FTP Brute-Force attempt [**] [Priority: 0] {TCP} 192.168.16.4:21 -> 192.16
8.16.2:34480
01/14-01:53:14.557187  [**] [1:1000001:10] FTP Brute-Force attempt [**] [Priority: 0] {TCP} 192.168.16.4:21 -> 192.16
8.16.2:34480
01/14-01:53:19.039061  [**] [1:1000001:10] FTP Brute-Force attempt [**] [Priority: 0] {TCP} 192.168.16.4:21 -> 192.16
8.16.2:34480
01/14-01:53:22.205476  [**] [1:1000001:10] FTP Brute-Force attempt [**] [Priority: 0] {TCP} 192.168.16.4:21 -> 192.16
8.16.2:34480
01/14-01:53:26.644770  [**] [1:1000001:10] FTP Brute-Force attempt [**] [Priority: 0] {TCP} 192.168.16.4:21 -> 192.16
8.16.2:34482
01/14-01:53:26.644869  [**] [1:1000001:10] FTP Brute-Force attempt [**] [Priority: 0] {TCP} 192.168.16.4:21 -> 192.16
8.16.2:34482
```

Si on sort du mode écoute de Snort, on peut exécuter le script ftpthreat.sh à l'aide de la commande suivante :

```
./ftpthreat.sh
```

Ce script va envoyer une requête vers l'API du contrôleur afin de bloquer la connexion du poste A.



On peut alors voir sur le poste A qu'on a un timeout, on a été banni du réseau :

```
Login failed.  
ftp> exit  
221 Goodbye.  
root@3ead2326c290:/# ftp 192.168.16.4  
ftp: connect: Connection timed out
```

Figure 12 - Bannissement du poste A

Avec l'aide de notre terminal connecté au poste B, on va pouvoir vérifier que l'accès au serveur web est toujours possible. Cette manipulation nous permet de confirmer que seul le poste A a été banni et que la continuité de service pour les autres postes est maintenue.

A l'aide de la commande :

```
ftp 192.168.16.4
```

On tente de se connecter au serveur web qui possède l'adresse IP 192.168.16.4 via FTP.

```
root@cd1d022b3f42:/# ftp 192.168.16.4  
Connected to 192.168.16.4.  
220 (vsFTPd 3.0.3)  
Name (192.168.16.4:root): user  
331 Please specify the password.  
Password:  
230 Login successful.  
Remote system type is UNIX.  
Using binary mode to transfer files.  
ftp>
```

Figure 13 - Connexion réussie en FTP au serveur web

On peut observer que la connexion est opérationnelle, en conclusion, seul le poste A à bien été banni du réseau.

Cette conclusion peut être confirmée en allant simplement regarder dans la table de flows du projet, on devrait normalement y retrouver une nouvelle règle demandant le bannissement de l'IP 192.168.16.2.

On exécute donc la commande suivante :

```
Sudo ovs-ofctl dump-flows miniprojet
```

On peut bien observer notre nouvelle règle qui banni l'IP 192.168.16.2, il s'agit de l'IP du poste A :

```
cookie=0x57a, duration=28.669s, table=0, n packets=8, n bytes=592, idle timeout=60, hard timeout=180, priority=20, ip,nw_dst=192.168.16.2 actions=drop
```

Figure 14 - Bannissement du poste A dans la table de flows

En conclusion, nous pouvons observer que tout fonctionne bien, le poste A a bien été banni et le poste B possède toujours l'accès au serveur.



4\ Conclusion

Pour conclure, comme vu dans le chapitre précédent, le projet est arrivé à son terme et est fonctionnel. La vidéo de présentation nous montre un projet fonctionnant, on peut bien observer une attaque détectée par un IDS déclenchant une commande vers le contrôleur, afin de mettre à jour les tables de flux des commutateurs du réseau et bloquer l'attaque.

Pour ce faire, nous avons utilisé les technologies Docker, ODL, Snort et OpenVSwitch, nous avons développé des scripts en bash et en python afin de permettre à chaque applicatif d'exécuter parfaitement les actions attendues.

Docker, avec l'aide des dockerfiles développés nous a permis d'isoler chaque applicatif dans des conteneurs, Snort, quant à lui, nous a permis d'analyser le trafic et de lever une alarme lors de la détection de l'erreur 530 d'authentification FTP.

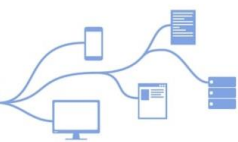
OpenVSwitch a permis la connectivité de tous nos applicatifs, ce dernier étant piloté par ODL, nous avons possibilité de modifier l'interconnectivité des équipements par rapport aux alarmes annoncées par Snort.

Sur un réseau d'entreprise, l'utilisation de ces technologies permettrait l'amélioration de la sécurité réseau. En effet, en reprenant la même base et en couplant ce projet aux autres sécurités réseaux plus classiques, à savoir les pare-feux, les VLANs ou encore l'utilisation de DMZ, on pourrait obtenir un réseau fortement sécurisé qui s'adapterait en fonction des attaques.

Grâce à Snort, on aurait la possibilité d'observer une multitude de paramètres et de lever une alarme au contrôleur ODL qui s'occuperait de modifier la topologie réseau en fonction des différentes attaques.

On obtiendrait alors un réseau non seulement fortement sécurisé, mais surtout, évolutif, qui s'adapterait et se modifierait constamment en fonction des attaquants tout en garantissant une continuité de services aux équipements ayant un comportement normal.

En conclusion, on obtiendrait une topologie n'étant pas fixe, mais complètement changeante, ce qui rendrait la tâche difficile aux attaquants.



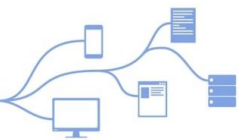
Annexes

cleanminiprojet.sh

```
# suppression de la configuration miniprojet
docker stop $(docker ps -a -q)
docker rm $(docker ps -a -q)
ovs-vsctl del-br miniprojet
```

getnode.py

```
import requests
import sys
print (" identification de l'id openflow du switch et de l'id des ports '\n'")
odl_username = 'admin'
odl_password = 'admin'
odl_url = 'http://172.17.0.2:8181/restconf/operational/network-topology:network-topology'
try:
    response = requests.get(odl_url, auth=(odl_username, odl_password))
except Exception as e:
    print("erreur!", e.__class__, "dans l'URL")
    quit()
if str(response)=="<Response [200]>":
    print ( str(response), " ok le controleur repond")
else :
    print ( str(response), " ko inutile d'aller plus loin")
    quit()
nodes= response.json()['network-topology']['topology']
#print(nodes)
try:
    for topology in nodes :
        print("id topologie:",topology['topology-id'])
        for sw in topology['node']:
            sub_id_node= sw['node-id'][9:len(sw['node-id'])]
            print("id openflow du switch :",sub_id_node)
            for port in sw['termination-point']:
                port_id=port['tp-id'][port['tp-id'].rfind(":")+1:len(port['tp-id'])]
                print(" port  du switch id ",port_id)
except Exception as e:
    print("impossible de recuperer la topologie",'\\n',"le controleur n'est probablement pas correctement connecté au swith",e.__class__)
```

createmini projet.sh

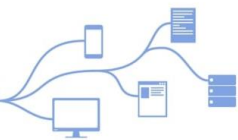
```
# executer ce shell en mode sudo
# installer openvswitch et ovs-docker
echo -e "\e[7m nettoyage de l'environnement docker et OVS \e[0m"
sudo ./cleanmini projet.sh
sleep 2
echo -e "\e[7m creation du switch ovs mini projet\e[0m"
sudo ovs-vsctl add-br mini projet
sleep 3
echo -e "\e[7m creation des 5 containers herbergeant les différents serveurs en mode detach\e[0m"
docker run -itd --name sdn_ctl opendaylight/odl bash
docker run -itd --name posteb amouat/network-utils bash
docker run -itd --name postea amouat/network-utils bash
docker run -itd --name ids linton/docker-snort bash
docker run -itd --name web_server -p 21:21 -e FTP_USER=user -e FTP_PASSWORD=user panubo/vsftpd
docker ps
sleep 5
echo -e "\e[7m Installation de packages \e[0m"
docker exec ids /bin/sh -c "apt update;apt install -y nano ftp python3 python3-pip net-tools;pip3 install requests --upgrade;mkdir /ftptthreat"
docker exec web_server /bin/sh -c "apt update;apt install -y nano ftp net-tools"
docker exec postea /bin/sh -c "apt update;apt install -y nano ftp net-tools"
docker exec posteb /bin/sh -c "apt update;apt install -y nano ftp net-tools"
sleep 5
echo -e "\e[7m Positionnement des fichiers nécessaires et modification des règles \e[0m"
docker cp putflow.py ids:/ftptthreat/putflow.py
docker cp /home/user/Projet/mini projetfile/flow/fl1402.json ids:/ftptthreat/fl1402.json
docker cp ftptthreat.sh ids:/ftptthreat/ftptthreat.sh
docker cp snortruleftp.txt ids:/ftptthreat/snortruleftp.txt
docker exec ids /bin/sh -c "chmod 777 -R /ftptthreat"
docker exec ids /bin/sh -c "cat /ftptthreat/snortruleftp.txt >> /etc/snort/rules/local.rules"
sleep 5
echo -e "\e[7m création d'une interface eth1 sur les containers + attachement de l'interface au port du switch \e[0m"
echo -e "\e[7m affectation des adresses ip sur le réseau 192.168.16.X/24 et adresses mac\e[0m"
echo -e "\e[7m NB : la configuration réseau du contrôleur n'est pas modifiée \e[0m"
sleep 5
echo -e "\e[7m le postea sera attaché au port openflow=1 IP 192.168.16.1 (les numéros sont attribués par ordre de création)\e[0m"
ovs-docker add-port mini projet eth1 postea --ipaddress=192.168.16.1/24 --macaddress="10:00:00:00:00:01"
echo -e "\e[7m le posteb sera attaché au port openflow=2 IP 192.168.16.2\e[0m"
ovs-docker add-port mini projet eth1 posteb --ipaddress=192.168.16.2/24 --macaddress="10:00:00:00:00:02"
echo -e "\e[7m l'IDS sera attaché au port openflow=3 IP 192.168.16.3\e[0m"
ovs-docker add-port mini projet eth1 ids --ipaddress=192.168.16.3/24 --macaddress="10:00:00:00:00:03"
echo -e "\e[7m le Web serveur sera attaché au port openflow=4 IP 192.168.16.4 \e[0m"
```



```
ovs-docker add-port miniprojet eth1 web_server --ipaddress=192.168.16.4/24 -
-macaddress="10:00:00:00:00:04"
echo -e "\e[7m le controleur sera attaché au port openflow=5 IP 192.168.16.
5 \e[0m"
ovs-docker add-port miniprojet eth1 sdn_ctl --ipaddress=192.168.16.5/24 --m
acaddress="10:00:00:00:00:05"

sleep 5
echo -e "\e[7m déconnexion de l'interface par défaut eth0 du bridge docker d
es containers\e[0m"
echo -e " \e[7m le container sdn_ctl qui instancie le controleur reste sur 1
a meme stack ip que le switch pour l'Init, il est accessible à l'adresse 1
72.17.0.2\e[0m,"
echo -e " \e[7m le controleur est accessible à l'adresse 192.168.16.5 depui
s L'IDS \e[0m,"

docker network disconnect bridge postea
docker network disconnect bridge posteb
docker network disconnect bridge ids
docker network disconnect bridge web_server
sleep 5
echo -e "\e[7m démarrage du controleur ODL : karaf \e[0m"
docker exec sdn_ctl /opt/opendaylight/bin/start clean
echo -e "\e[7m démarrage du plugins openflow du controleur odl-openflowpl
ugin-flow-services-rest\e[0m "
sleep 10
docker exec sdn_ctl /opt/opendaylight/bin/client feature:install odl-open
flowplugin-flow-services-rest
echo -e "\e[7m démarrage du plugins openflow du controleur odl-restconf\e
[0m "
sleep 10
docker exec sdn_ctl /opt/opendaylight/bin/client feature:install odl-restc
onf
echo -e "\e[7m configuration openflow du switch\e[0m"
sleep 15
sudo ovs-vsctl set bridge miniprojet protocols=OpenFlow14,OpenFlow13
sudo ovs-vsctl set-controller miniprojet tcp:172.17.0.2
echo -e " \e[7m verification du switch\e[0m"
sleep 5
sudo ovs-vsctl show
echo -e "\e[7mlancement de la configuration openflow du swich\e[0m"
echo -e "\e[7mlancement du script python initswich.py dans le repertoire c
ourant\e[0m"
echo -e "\e[7mce script prend en argument les id des fichier flow json situ
é dans le repertoire\e[0m"
echo -e "\e[7m../flow ces fichiers sont de la forme fl<id_flow>.json\e[0m"
sleep 5
sudo python3 ./initswitchb.py 100 200 300 400 500 101 202 303 404 505
echo -e " \e[7m dumps des flow dans le switch\e[0m"
sleep 5
sudo ovs-ofctl dump-flows miniprojet
echo -e "\e[7m resultat : les paquets ip sont routés vers le container cor
respondant à l'adresse ip destinataire et vers IDS \e[0m"
echo -e "\e[7m resultat : les paquet ARP sont forwardes sur tous les ports d
u switch\e[0m"
echo -e "\e[7mFIN DE LA CONFIGURATION INITIALE DU POC MINIPROJET\e[0m"
echo -e " \e[7m pour se connecter sur un container < docker attach contain
er> ctrl p ctrl q pour sortir sans le tuer\e[0m"
```

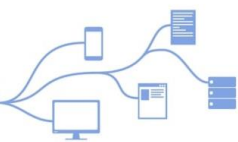


initswitchb.py

```
import requests
import sys
import json
def putflow(id_node,id_flow,fic_flow):
    print("id cookie:0x",hex(int(id_flow)))

#ouverture et lecture du fichier flow
    try:
        fic_flow = open(fic_flow,"r")
        flow = fic_flow.read()
        fic_flow.close()
    except Exception as e:
        print("problème à l'ouverture ou a la lecture du fichier flow", e.__class__)
        quit()
    convflow=json.loads(flow)
    odl_username = 'admin'
    odl_password = 'admin'
    odl_url = "http://172.17.0.2:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:{}/table/0/flow/{}".format(id_node,id_flow)
    try:
        response = requests.put(odl_url, auth=(odl_username, odl_password),json=convflow)
    except Exception as e:
        print("erreur!", e.__class__, "dans l'URL")
        quit()
    print("reponse controleur",response,'\n')

def idnode():
    print (" identification de l'id openflow du switch et de l'id des ports '\n'")
    odl_username = 'admin'
    odl_password = 'admin'
    odl_url = 'http://172.17.0.2:8181/restconf/operational/network-topology:network-topology'
    try:
        response = requests.get(odl_url, auth=(odl_username, odl_password))
    except Exception as e:
        print("erreur!", e.__class__, "dans l'URL")
        quit()
    if str(response)=="<Response [200]>":
        print ( str(response), " ok le controleur repond")
    else :
        print ( str(response), " ko inutile d'aller plus loin")
        quit()
    nodes= response.json()['network-topology']['topology']
    #print(nodes)
    try:
        for topology in nodes :
            print("id topologie:",topology['topology-id'])
            for sw in topology['node']:
                sub_id_node= sw['node-id'][9:len(sw['node-id'])]
                print("id openflow du switch :",sub_id_node)
                for port in sw['termination-point']:
                    port_id=port['tp-id'][port['tp-id'].rfind(":")+1:len(port['tp-id'])]
                    print(" port du switch id ",port_id)
```



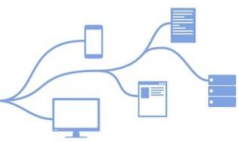
```
except Exception as e:
    print("impossible de recuperer la topologie",'\\n',"le contro
leur n'est probablement pas correctement connecté au switch",e.__class__)

    return(sub_id_node)
#programme principal
# recuperation de l'id openflow du switch
id_nodes=idnode()
print("lancement configuration initiale du switch :{}".format(id_nodes))
#contruction du nom de fichier fl *.json, * etant id du fichier passé en ar
gument
#l'id du flow openflow a passer dans l'URL RESTCONF = l'id du fichier flow
#le numero de cookie = numero de l'id du flow openflow
# les fichier json qui contiennent les flow doivent etre coherents avec ces
règles

for id_fic_flow in sys.argv:
    if id_fic_flow.isalnum():
        fic_flow="../../flow/""fl"+id_fic_flow+".json"
        id_flow=id_fic_flow
        print("lancement du fichier flow:{} ".format(fic_flow))
        print("id flow:{} ".format(id_flow))
        putflow(id_nodes,id_flow,fic_flow)
```

ftptthreat.sh

```
#!/bin/bash
cd /var/log/snort
logfile=$(ls -lrt | tail -n1)
tcpdump -n -tttt -r $logfile > /ftptthreat/log.txt
cat /ftptthreat/log.txt | tail -30 > /ftptthreat/log30.txt
iptoban=$(grep -o "[0-9]\\+\\. [0-9]\\+\\. [0-9]\\+\\. [0-9]\\+" /ftptthreat/log30.txt
| sort | uniq | tail -2 | grep -v "192.168.16.4")
nbtry=$(grep -o "[0-9]\\+\\. [0-9]\\+\\. [0-9]\\+\\. [0-9]\\+" /ftptthreat/log30.txt |
sort | grep -v "192.168.16.4" | wc -l)
if [ "$nbtry" -gt 3 ]
then
    echo $iptoban " sera ban pour 1min"
    python3 /ftptthreat/putflow.py 192.168.16.5 /ftptthreat/fl1402.json 1
402
else
    echo "aucune ip à bannir"
fi
```



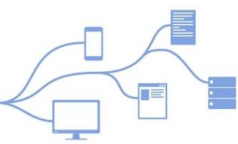
putflow.py

```
import requests
import sys
import json
def putflow(ip,id_node,id_flow,fic_flow):
    print("id cookie:0x",hex(int(id_flow)))

#ouverture et lecture du fichier flow
    try:
        fic_flow = open(fic_flow,"r")
        flow = fic_flow.read()
        fic_flow.close()
    except Exception as e:
        print("problème à l'ouverture ou a la lecture du fichier flow", e.__class__)
        quit()
    convflow=json.loads(flow)
    odl_username = 'admin'
    odl_password = 'admin'
    odl_url = "http://{ }:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:{ }/table/0/flow/{ }".format(ip,id_node,id_flow)
    print("URL du flow=",odl_url)
    try:
        response = requests.put(odl_url, auth=(odl_username, odl_password),json=convflow)
    except Exception as e:
        print("erreur!", e.__class__, "dans l'URL")
        quit()
    print("reponse controleur",response,'\n')

def idnode(ip):

    print (" identification de l'id openflow du switch et de l'id des ports '\n'")
    odl_username = 'admin'
    odl_password = 'admin'
    odl_url = "http://{ }:8181/restconf/operational/network-topology:network-topology".format(ip)
    print("URL topology",odl_url)
    try:
        response = requests.get(odl_url, auth=(odl_username, odl_password))
    except Exception as e:
        print("erreur!", e.__class__, "dans l'URL")
        quit()
    if str(response)=="<Response [200]>":
        print ( str(response), " ok le controleur repond")
    else :
        print ( str(response), " ko inutile d'aller plus loin")
        quit()
    nodes= response.json() ['network-topology'] ['topology']
    #print(nodes)
    try:
        for topology in nodes :
            print("id topologie:",topology['topology-id'])
            for sw in topology['node']:
                sub_id_node= sw['node-id'][9:len(sw['node-id'])]
                print("id openflow du switch :",sub_id_node)
                for port in sw['termination-point']:
```



```
port_id=port['tp-id'][port['tp-id'].r
find(":"+1:len(port['tp-id']))
print(" port  du switch id ",port_id)
except Exception as e:
    print("impossible de recuperer la topologie",'\n',"le contro
leur n'est probablement pas correctement connecté au swith",e.__class__)

    return(sub_id_node)
#programme principal
# recuperation de l'id openflow du switch
ip=sys.argv[1]
fic_flow=sys.argv[2]
id_flow=sys.argv[3]
id_nodes=idnode(ip)
print("lancement configuration  du switch :{}".format(id_nodes))
#contruction du nom de fichier fl *.json, * etant id du fichier passé en ar
gument
#l'id du flow openflow a passer dans l'URL RESTCONF = l'id du fichier flow
#le numero de cookie = numero de l'id du flow openflow
# les fichier json qui contiennent les flow doivent etre coherents avec ces
règles
print("lancement du fichier flow:{}".format(fic_flow))
print("id flow:{}".format(id_flow))
putflow(ip,id_nodes,id_flow,fic_flow)
```

Vidéo de démonstration

<https://www.youtube.com/watch?v=B52Z-F6vLvA>

GitHub du projet

<https://github.com/max7703/simulation-attack-ids-sdn-odl>