

Table des matières

1	Introduction	2
2	Technologie de virtualisation	2
3	Création des sous-réseaux	4
4	Mise en place des machines factices	6
5	Création du Firewall	7
5.1	Définition des règles de filtrage	8
5.2	Durcissement configuration	8
6	Utilisation de Syslog-ng pour les logs FW et Serveur	8
7	Mise en place de la défense réseau avec Suricata	8
8	Mécanisme de défense hôte avec OSSEC	8
9	Affichage des logs avec Kibana	8
10	Attaque avec metasploit	8
11	Conclusion	8

1 Introduction

L'objectif de ce projet est de concevoir une simulation dans laquelle nous allons tenter de déclencher une intrusion puis de la détecter en remontant l'alerte.

Pour arriver à cet objectif nous allons donc devoir :

- créer un ensemble de machines virtuelles connectées entre elles
- créer un firewall permettant de filtrer les requêtes
- créer un serveur en DMZ qui sera notre "point sensible"
- créer un attaquant, une machine virtuelle externe, qui tentera de pénétrer le réseau

Ce rapport décrit notre procédure et les évolutions du projet au fur et à mesure de nos avancées ainsi que de nos recherches.

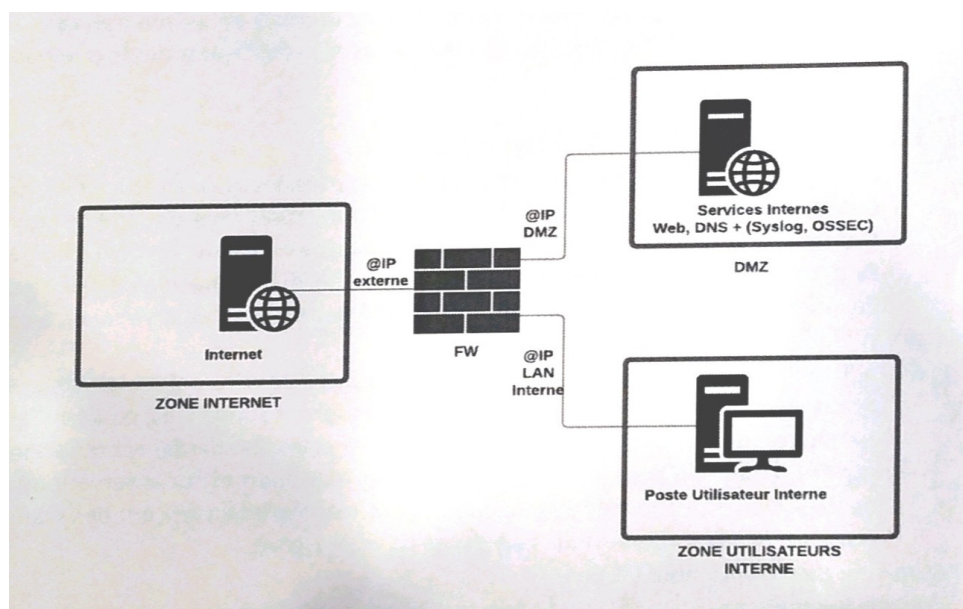


FIG. 1 : Schéma initial donné par le sujet

2 Technologie de virtualisation

La toute première réflexion que nous avons eue était le choix de la technologie permettant de faire tourner nos multiples machines virtuelles. Le sujet nous proposant un minimum de 4 machines, il nous paraissait difficile de toutes les faire tourner sous **VirtualBox** ou **VMware** en raison des limites de matériel à notre disposition, en effet nous travaillons majoritairement sur PC portable et nous sommes limités par la quantité de RAM et de puissance de calcul.

Un autre type de virtualisation existe cependant : la “**containérisation**”. Le principe ? Au lieu d’émuler une machine complète, on n’**émule que la partie applicative**, ainsi le noyau de la machine est utilisé pour les appels courants cependant les machines sont isolées du système et les unes des autres dans des conteneurs. Ainsi nous pouvons faire tourner un plus grand nombre de ces machines virtuelles avec le même matériel.

La technologie que nous avons décidé d’utiliser est Docker.



FIG. 2 : Logo du groupe Docker

L’outil docker-compose va nous permettre de **décrire les réseaux** ainsi que les interactions entre chaque image composant ce projet.

Ainsi nous allons **décrire chaque machine**, les configurer, créer des réseaux virtuels et attribuer chaque machine à ces réseaux, mettre en place un pare-feu et ainsi de suite.

Nous avons donc la structure de fichier suivante :

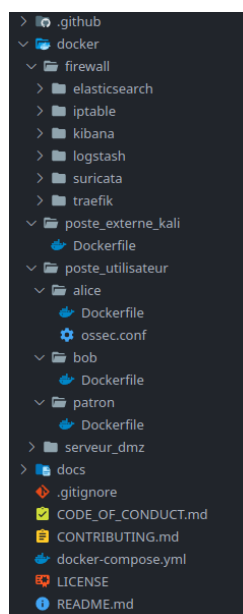


FIG. 3 : Structure de fichiers du projet

Chaque dossier dans le dossier `docker` représente une machine sur le réseau. Dans chacun de ces

dossier on va retrouver un fichier `Dockerfile` contenant la configuration de base de la machine avec par exemple les fichiers à copier dans le conteneur. Une telle configuration ressemble au fichier suivant :

```
1 FROM atomiccorp/ossec-docker
2
3 # copy config
4 COPY ossec.conf /var/ossec/etc/ossec.conf
5
6 # restart ossec
7 RUN ["/var/ossec/bin/syscheck_update", "-a"]
8 RUN ["/var/ossec/bin/syscheck_update", "-l"]
9 RUN ["/var/ossec/bin/ossec-control", "restart"]
```

On se base sur une image préexistante sur Docker Hub, puis on peut y copier des fichiers, exécuter des commandes et ainsi de suite. Ces commandes seront exécutées lors du build de l'image du conteneur. Nous travaillerons alors avec cette image fraîche pour le reste du projet.

3 Création des sous-réseaux

Nous venons de voir comment nous allons créer chaque machine. Maintenant nous devons nous poser la question de **comment organiser** ces machines en réseau afin de pouvoir commencer à configurer ces dernières et à simuler les intrusions.

Pour commencer avec la configuration de docker-compose, nous avons mis **toutes les machines sur le même réseau**, en exposant les ports nécessaires. Cela fonctionnait mais était bien loin de ce que l'on peut retrouver en entreprise ou simplement dans un réseau réel.

Nous avons découvert que docker-compose permet de créer **un lien DNS** direct vers chaque machine du réseau afin de pouvoir y accéder simplement par son nom. Imaginons maintenant que l'image docker de mon pare-feu s'appelle `firewall`, pour y accéder depuis n'importe quelle machine présente sur le réseau, nous n'avons pas besoin de connaître son identité, il suffira d'y accéder par : `http://firewall/` ce qui est pratique.

Nous avons donc réfléchi à la manière d'organiser ce réseau de machines. Le plus simple était de commencer par la machine externe au système. On suppose qu'elle se trouve sur internet donc elle est mise dans son propre sous-réseau que l'on va appeler `Internet`.

Ensuite avant d'accéder à l'ensemble des machines sur le réseau, les requêtes devront passer par un router, c'est donc le nœud suivant. Ensuite nous avons un sous-réseau pour le pare-feu comprenant différents services tels que Logstash, Suricata, Kibana ou encore Elasticsearch. Dans notre simulation, chacun de ces services tournera sur sa propre machine. Il s'agira du sous-réseau `vlan-FW`.

On continue avec l'espace DMZ contenant le site web, le serveur DNS et syslog pour remonter les logs par Logstash. Ce sous-réseau sera nommé `vlan-DMZ`.

Enfin nous avons l'espace utilisateur avec les postes utilisateur des employés de l'entreprise : Alice, Bob et le CEO. Ce sous-réseau portera le nom de `vlan-ZUI`.

Voici le schéma récapitulatif de ce que nous allons essayer d'implémenter pour ce projet. On distingue bien les différents sous-réseaux présents.

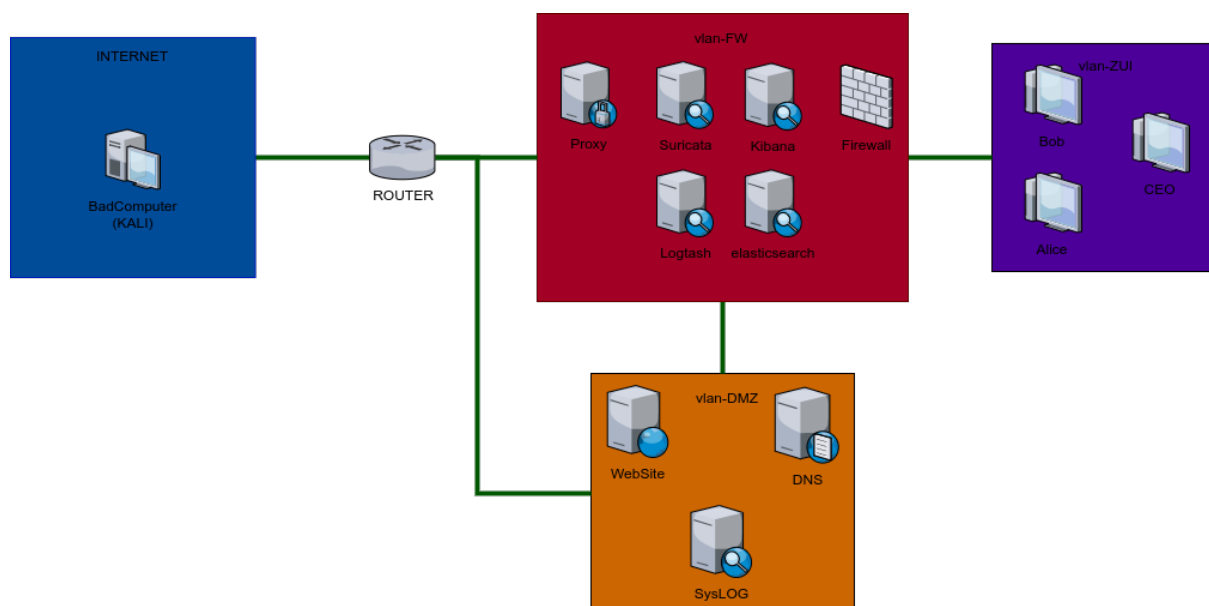


FIG. 4 : Schéma du réseau complet que nous proposons

En utilisant docker-compose cette configuration se faire de façon relativement simple :

```
1 version: "3.1"
2 services:
3     ## service name, NOT the name of the container (that gets assigned
4     ## automatically)
5     ## === ZUI ===
6     poste-utilisateur-CEO:
7         ## image from which the container should be built, equals to
8         ## FROM in Dockerfile
9         build: ./docker/poste_utilisateur/ceo/
10        tty: true
11        stdin_open: true
12        ## restart on crash
13        restart: always
14        # ports to expose
15        ports:
16            # host:container
17            - "2221:22"
```

```

16     networks:
17         # the virtual lan used for this service
18         vlan-zui:
19             ipv4_address: 10.10.3.2
20     ## === INTERNET ===
21     kali:
22         build: ./docker/poste_externe_kali/
23         tty: true
24         stdin_open: true
25         restart: always
26         ports:
27             - "2222:22"
28         networks:
29             # another sub-network
30             internet:
31                 ipv4_address: 10.10.0.2

```

Maintenant que les sous-réseaux sont créés, il suffit d'aller dans le répertoire principal du projet et de taper les commandes suivantes :

```

1 sudo systemctl start docker.service
2 docker-compose build
3 docker-compose up

```

En tapant ensuite la commande `docker ps` on peut voir l'ensemble des conteneurs qui tournent sur la machine. Avec `docker stats` on peut suivre la consommation des ressources de l'ensemble des machines du réseau :

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
072baf68cbb0	tpi-network-simulation-traefik_1	0.00%	17.31MiB / 15.32GiB	0.11%	29.5kB / 36kB	233kB / 0B	13
f9fd5d831ee4	tpi-network-simulation-kibana_1	0.28%	68.91MiB / 15.32GiB	0.44%	598kB / 231kB	0B / 0B	11
7bf04ee53a53	tpi-network-simulation-logstash_1	4.09%	826MiB / 15.32GiB	5.27%	1.8MB / 52.5MB	0B / 8.34MB	43
067a95c0f382	tpi-network-simulation-elasticsearch_1	0.47%	1.228GiB / 15.32GiB	8.01%	52.8MB / 2.4MB	0B / 180MB	136
2c512a24c293	tpi-network-simulation-whoami_1	0.00%	2.633MiB / 15.32GiB	0.02%	672B / 0B	0B / 0B	5
ce72518d3999	tpi-network-simulation-suricata_1	3.15%	68.76MiB / 15.32GiB	0.44%	112B / 504B	0B / 0B	14
f1e8b6952213	tpi-network-simulation-poste-utilisateur-patron_1	0.09%	45.64MiB / 15.32GiB	0.29%	0B / 3.68kB	0B / 442kB	56
68be2e8d0ceb	tpi-network-simulation-post-utilisateur-bob_1	0.00%	832KiB / 15.32GiB	0.01%	4.21kB / 0B	0B / 0B	1
6e7e16c0b753	tpi-network-simulation-kali_1	0.02%	291.4MiB / 15.32GiB	1.86%	36.5kB / 2.61kB	0B / 47.1MB	11
7a3683380947	tpi-network-simulation-apache_1	0.00%	16.11MiB / 15.32GiB	0.10%	672B / 0B	0B / 4.1kB	82
d4b20ba06dbf	tpi-network-simulation-syslog_1	0.00%	2.766MiB / 15.32GiB	0.02%	672B / 0B	0B / 20.5kB	2
f18e91c76261	tpi-network-simulation-post-utilisateur-alice_1	0.00%	804KiB / 15.32GiB	0.01%	4.21kB / 0B	0B / 0B	1
b48446f16081	tpi-network-simulation-dns_1	0.00%	68.32MiB / 15.32GiB	0.44%	0B / 504B	0B / 77.8kB	27

FIG. 5 : Statistiques de performance de l'ensemble des conteneurs présents dans le projet

Ce qui est bien moindre en comparaison avec des machines virtuelles sous VirtualBox ou VMware.

4 Mise en place des machines factices

Nous avons utilisé Traefik comme router qui nous permet notamment d'avoir une belle interface graphique :

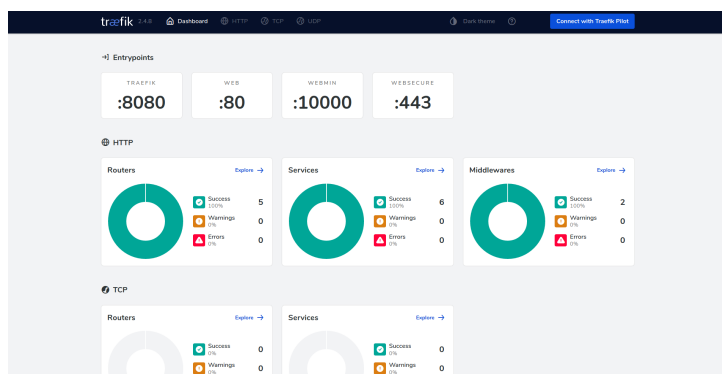


FIG. 6 : Le dashboard de traefik

Ainsi que de configurer les routes accessibles depuis l'extérieur :

<div> All Status Success Warnings Errors </div> <div>Search</div>						
Status	TLS	Rule	Entrypoints	Name	Service	Provider
✓		PathPrefix(/)	web	apache@docker	apache	🔗
✓		PathPrefix(/api)	traefik	api@internal	api@internal	🔗
✓		PathPrefix(/)	traefik	dashboard@internal	dashboard@internal	🔗
✓		PathPrefix(/)	webmin	dns@docker	dns	🔗
✓		PathPrefix(/whoami)	web	whoami@docker	whoami	🔗

FIG. 7 : Configuration des routes dans Traefik

5 Création du Firewall

Le sujet nous proposait de nous tourner vers pfSense, une solution de pare-feu open source. Cependant, nous avons essayé de lancer un conteneur Docker tournant avec une image de pfSense et ce sans succès. Nous avons alors cherché d'autres produits permettant d'atteindre les mêmes objectifs, c'est à dire d'avoir un pare-feu open-source et gratuit à notre disposition pour filtrer les requêtes et remonter tous les évènements à un serveur de collecte.

5.1 Définition des règles de filtrage

5.2 Durcissement configuration

6 Utilisation de Syslog-ng pour les logs FW et Serveur

-> logs centralisés via le protocole syslog, serveur de collecte sous syslog-ng

7 Mise en place de la défense réseau avec Suricata

-> Activer la fonction IDS, préciser les interfaces surveillées, quels sont les avantages inconvénients, utilisation jeu de règle standard, envoi des alertes sur serveur de collecte, tester depuis zone internet avec scan nmap

8 Mécanisme de défense hôte avec OSSEC

-> configuration des fonctions principales : vérification d'intégrité, détection de rootkit, collecte de logs...

9 Affichage des logs avec Kibana

10 Attaque avec metasploit

-> attaquer de l'extérieur et de l'intérieur, voir comment le système réagit, quels sont les alertes remontées

11 Conclusion

Certaines des parties que nous avons décrites peuvent paraître peu fournies. Nous avons pour sujet des technologies que nous ne connaissons pas et que nous ne maîtrisons pas. L'abandon des VM pour passer sur des conteneurs s'est effectué tard dans l'avancée du projet, nous forçant à tout recommencer depuis zéro afin d'obtenir un système "viable".

Pour revenir au sujet, il est clair que ce qui était attendu est une sorte de **SIEM** allégé en fonctionnalités. Le tout permettant de monitorer des événements à différentes échelles. Nous cherchions donc à relever :

- **les intrusions externes** : par le Firewall qui remonte les tentatives d'intrusion externe, c'est la façon la plus courante de logger les tentatives d'intrusion
- **les intrusions internes** : si une intrusion s'est fait en interne (non capturée par le Firewall ou bien accès physique en interne) nous avons 2 moyens de détecter cette intrusion
 - **détection par le réseau** (NIDS) : c'est le rôle d'une sonde réseau, de faire remonter les trafics suspects, au serveur de collecte, par le moyen d'un TAP par exemple
 - **détection sur la machine** (HIDS) : ici on cherche à détecter les comportements suspects sur une machine comme par exemple une élévation de privilèges, la création d'un nouveau compte, un changement de configuration ou de droits etc...

Ce système de **détection à 3 niveaux** permet en théorie de capturer l'ensemble des attaques possibles aujourd'hui en supposant qu'elles utilisent des vecteurs d'attaque connus. Les nouveaux virus arrivants sur le marché cherchent à contourner ces moyens de détection en changeant leur signature ou leur comportement de manière à ne pas être détecté.

Un soucis que l'on va avoir avec cette approche est que si nous faisons **remonter trop d'informations** (i.e. trop d'événements) le système ne sera pas capable de tous les gérer. Le SIEM va alors être **noyé sous les événements**. C'est pourquoi une bonne configuration est obligatoire, pour ne filtrer que les événements qui nous intéressent et ainsi ne remonter que les informations utiles.

Une autre solution pourrait être d'avoir un **serveur de collecte par sous-réseau** de manière à agréger les logs de son sous-réseau, de les filtrer avant de l'envoyer au serveur de collecte central.

Pour finir, nous avons tous appris beaucoup avec ce sujet, même si le temps nous manquait pour aller plus loin dans les démarches. Nous pensons que la détection d'intrusion est une stratégie qui est aujourd'hui nécessaire dans tout système contenant des données sensibles.