Table des matières

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.

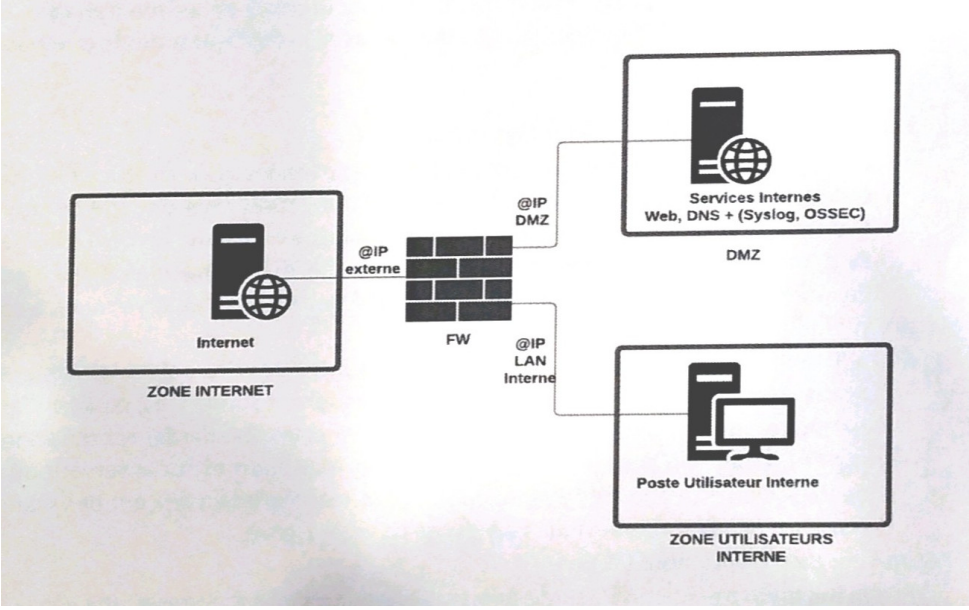

Fig. 1 : Schéma initial donné par le sujet

2 Technologie de virtualisation

La toute première réflexion que nous avons 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 :

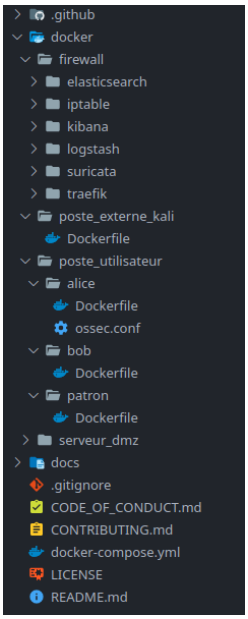

Fig. 3 : Structure de fichiers du projet

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 a copier dans le conteneur. Une telle configuration ressemble au fichier suivant :

```
1 FROM atomicorp/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 allions 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 avions 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 réfléchis à la manière d'organiser ce réseau de machines. Le plus simple était de commencer par la machine externe au système. On suppose quelle 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.

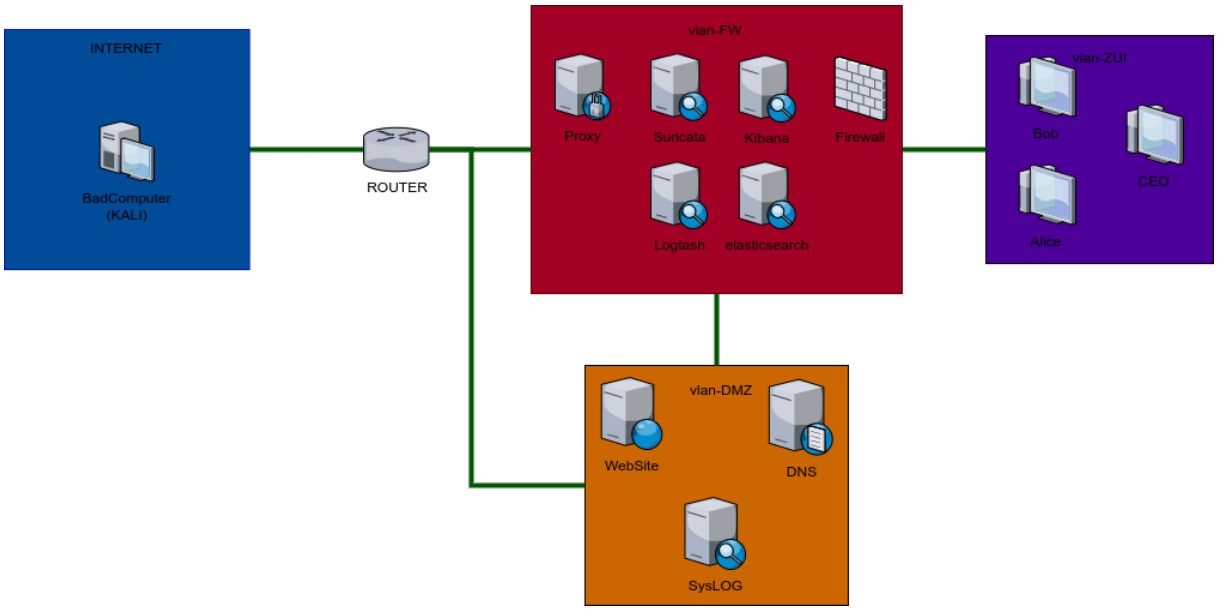

FIG. 4 : Schéma du réseau complet que nous proposons

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
           automatically)
4      ## === ZUI ===
5      poste-utilisateur-CEO:
6          ## image from which the conatiner should be built, equals to
               FROM in Dockerfile
7          build: ./docker/poste_utilisateur/ceo/
8          tty: true
9          stdin_open: true
10         ## restart on crash
11         restart: always
12         # ports to expose
13         ports:
14             # host:container
15             - "2221:22"
```

```
        networks:
            # the virtual lan used for this service
            vlan-zui:
                ipv4_address: 10.10.3.2
    ## === INTERNET ===
    kali:
        build: ./docker/poste_externe_kali/
        tty: true
        stdin_open: true
        restart: always
        ports:
            - "2222:22"
        networks:
            # another sub-network
            internet:
                ipv4_address: 10.10.0.2
```

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

```
sudo systemctl start docker.service
docker-compose build
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
6e7e16cdb753   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 :

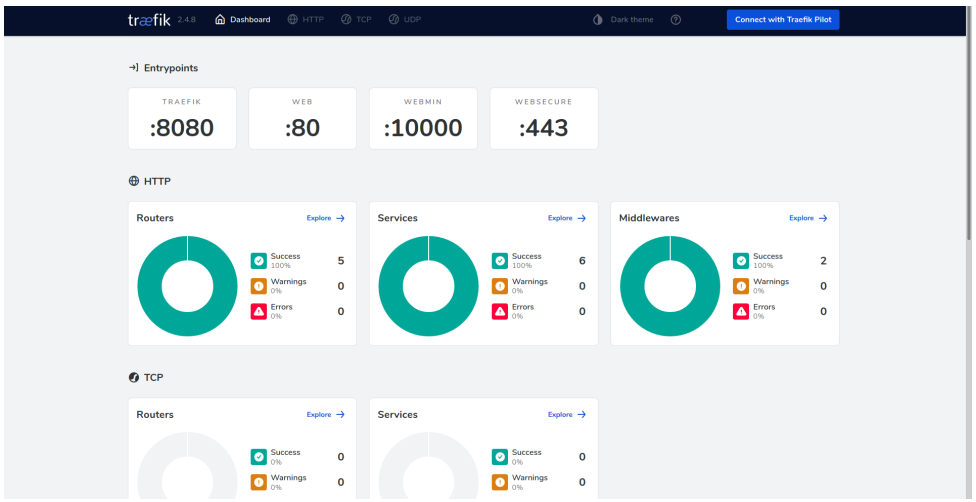

FIG. 6 : Le dashboard de traefik

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

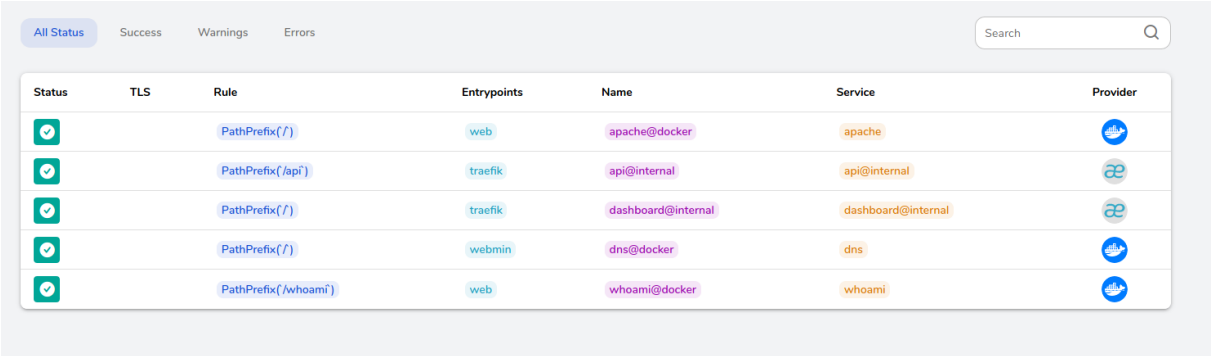

FIG. 7 : Configuration des routes dans Traefik

Nous avons par la même occasion créer un serveur web hébergeant une page toute simple :

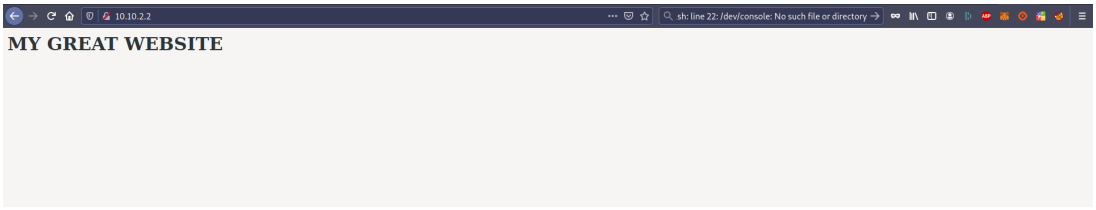

FIG. 8 : Page d'accueil du site web du projet

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.

Nous allons définir les règles de filtrage pour configurer le firewall. Étant sous un système UNIX nous allons utiliser `iptable` pour définir les règles de filtrage de base. Voici à quoi ressemble le fichier de configuration :

```
1  *filter
2
3  # INIT
4  :INPUT DROP [0:0]
5  :FORWARD ACCEPT [0:0]
6  :OUTPUT ACCEPT [44:6020]
7
8  # Basic input rules
9  -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
10 -A INPUT -p icmp -j ACCEPT
11 -A INPUT -i lo -m comment --comment "Loopback interface" -j ACCEPT
12
13 # Specific rules WEB
14 -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
15 -A INPUT -p tcp -m tcp --dport 443 -j ACCEPT
16
17 # Default REJECT LINE
18 -A INPUT -j LOG --log-prefix DROPPED_INGRESS-
19
20 COMMIT
```

Par défaut on a décidé de bloquer tout le trafic entrant. On fonctionne alors sur le principe de la whitelist, c'est à dire que seul un ensemble de ports explicités seront autorisés à entrer. Pour l'instant sur le trafic sortant on autorise tous les ports entre 44 et 6020.

Tout d'abord on autorise tous les ordinateurs. C'est à dire que l'on autorise tous les ordinateurs à communiquer avec le réseau. On accepte le protocole ICMP donc les pings venant de l'extérieur. Enfin on active le "Loopback interface" sur le firewall.

Ensuite nous avons autorisé le trafic web en ouvrant les ports 80 et 443 correspondant aux protocoles HTTP et HTTPS afin de pouvoir accéder au serveur web.

Pour terminer on autorise les logs a transiter sur le réseau. La dernière ligne permet d'enregistrer les règles sur le firewall.

6 Collecte logs FW et Serveur

Le sujet nous proposait d'utiliser le protocole syslog-ng afin de transférer les logs dans le système et enfin de pouvoir les collecter dans un serveur spécifique avant de les traiter dans une interface utilisateur dédié.

Nous avons décidé d'utiliser un stack de services différent qui nous attirait plus. Nous sommes donc parti sur Logstash un service permettant de collecter les logs issus d'une multitude de programme et de les "traduire" dans un format générique qui sera compris par les autres services d'analyse tels que Elasticsearch ou Kibana.

Il suffira que **chaque capteur** dans le système, que ce soit au niveau firewall, niveau réseau ou niveau machine, puisse **envoyer ses logs** au service Logstash. Ce dernier se chargera de la traçabilité de l'information et du traitement de cette dernière.

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, envoie des alertes sur serveur de collecte, tester depuis zone internet avec scan nmap

8 Mécanisme de défense hôte avec OSSEC

OSSEC est un paquet **à installer sur chaque machine que nous souhaitons monitorer**. Cependant en raison du manque de temps et des tâches restantes nous avons décidé de partir sur une image docker incluant déjà OSSEC. Ainsi nous avons une machine linux tournant avec OSSEC. En conservant la configuration par défaut nous pouvons déjà monitorer quelques évènements et les envoyer à Logstash pour finir sur le serveur de collecte.

Ce qui faut retenir de cette partie est que **OSSEC permet de faire du monitoring en tant que capteur au niveau d'une machine**. Il s'agit alors d'une sonde hôte. Parmi les évènements que l'on peut monitorer on retrouve la modification de contenu dans certains dossiers tels que `/etc/` ou encore la modification de droits, la création d'utilisateurs, le changement de configuration des groupes, etc… **Une des forces de OSSEC est sa capacité à détecter les rootkits et à vérifier l'intégrité du système.**

9 Affichage des logs avec Kibana

Pour l'affichage des logs et leur recherche en **filtrant les logs** qui ne sont pas intéressant nous avons la possibilité d'utiliser elasticsearch ou Kibana. Nous avons décidé de rester sur Kibana qui était une solution plus légère. Avec Logstash en tant que middleware, le résultat sera le même.

Ainsi avec une **configuration de base** (à comprendre par défaut) nous pouvons déjà récupérer quelques logs de test et afficher le nombre d'évènement reçus sur le dashboard :

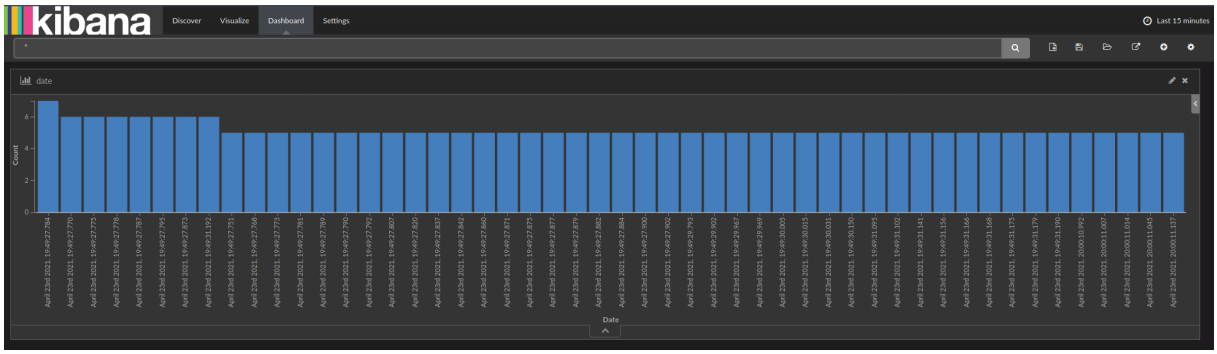

FIG. 9 : Les logs s'affichent par ordre chronologique sur le dashboard Kibana

10 Attaque avec Metasploit

En utilisant une image Kali linux avec le framework Metasploit, plusieurs outils sont disponibles afin de tester les capteurs que nous avons installé.

Pour tester le firewall nous pouvons utiliser l'outil `nmap` avec par exemple la commande suivante qui permet de scanner l'ensemble des ports ouverts entre 1 et 10 000 en cherchant à identifier les protocoles ouverts ainsi que leurs vulnérabilités :

```
1 nmap -sV --script=vulners -v 10.0.0.XX
```

Ainsi du côté de la Kali linux sur internet nous ne trouvons que les ports 80 et 443 d'accessibles sans aucune vulnérabilité (par le script vulners) :

```
1 Starting Nmap 7.80 ( https://nmap.org ) at 2020-05-07 17:28 CEST
2 NSE: Loaded 46 scripts for scanning.
3 NSE: Script Pre-scanning.
4 Initiating NSE at 17:28
5 Completed NSE at 17:28, 0.00s elapsed
6 Initiating NSE at 17:28
7 Completed NSE at 17:28, 0.00s elapsed
8 Initiating Ping Scan at 17:28
```

```
Scanning 10.0.0.XX [4 ports]
Completed Ping Scan at 17:28, 0.19s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 17:28
Completed Parallel DNS resolution of 1 host. at 17:28, 2.17s elapsed
Initiating SYN Stealth Scan at 17:28
Scanning 10.0.0.XX [1000 ports]
Discovered open port 80/tcp on 10.0.0.XX
Discovered open port 443/tcp on 10.0.0.XX
Completed SYN Stealth Scan at 17:28, 5.58s elapsed (1000 total ports)
Initiating Service scan at 17:28
Scanning 16 services on x.x.x.x
Service scan Timing: About 56.25% done; ETC: 17:30 (0:00:44 remaining)
Completed Service scan at 17:30, 84.04s elapsed (16 services on 1 host)
NSE: Script scanning x.x.x.x.
Initiating NSE at 17:30
Completed NSE at 17:30, 6.91s elapsed
Initiating NSE at 17:30
Completed NSE at 17:30, 1.55s elapsed
Nmap scan report for x.x.x.x
Host is up (0.19s latency).
Not shown: 984 closed ports
PORT      STATE SERVICE            VERSION
80/tcp    open  http               Apache httpd 2.x.xx (OpenSSL/1.0.2d
   PHP/5.6.20)
Service detection performed. Please report any incorrect results at
   https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 100.83 seconds
           Raw packets sent: 1022 (44.944KB) | Rcvd: 1001 (40.108KB)
```

Pour tester le fonctionnement de suricata en tant que sonde réseau, on peut essayer d'ouvrir un reverse shell depuis l'une des machines vers la Kali Linux. Ainsi ce trafic devrait être récupéré et loggé par suricata. Avec une bonne configuration on pourrait même "drop" (c'est à dire arrêter) la requête pour éviter les fuites de données.

Enfin pour tester le fonctionnement de OSSEC on pourrait déployer un payload sur la machine où est installée OSSEC et essayer de faire de la mitigation en insérant notre reverse shell (ou session meterpreter) dans un autre processus, ou en tentant une élévation de privilèges.

Dans ces deux derniers cas, suricata et OSSEC devraient générer une alerte ou un log qui sera envoyé au serveur de collecte. Dans ce cas **Logstash récupère ce log** qui est propre à chaque logiciel **pour le convertir en un log générique** qui sera ensuite "interprété" par Kibana pour afficher les informations pertinentes à l'OSSI ou au technicien surveillant le système.

> **Note** : pour simplifier l'attaque sur ce système et n'ayant pas nécessairement les connaissances nécessaire pour dérouler un exploit au complet, nous avons ajouté une machine tournant sur Metasploitable, une version d'ubuntu possédant volontairement des failles afin de guider les

> débutants sur l'utilisation de Metasploit

11 Conclusion

Certaines des parties que nous avons décrites peuvent paraître peu fournies. Nous avions pour sujet des technologies que nous ne connaissions pas et que nous ne maîtrisions 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 évènements 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 suspicieux, au serveur de collecte, par le moyen d'un TAP par exemple
  - **détection sur la machine** (HIDS) : ici on cherche à détecter les comportements suspicieux 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'évènements) le système ne sera pas capable de tous les gérer. Le SIEM va alors être **noyé sous les évènements**. C'est pourquoi une bonne configuration est obligatoire, pour ne filtrer que les évènements 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.