

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	8
6	Collecte logs FW et Serveur	9
7	Mise en place de la défense réseau avec Suricata	9
8	Mécanisme de défense hôte avec OSSEC	10
9	Affichage des logs avec Kibana	10
10	Attaque avec Metasploit	11
11	Conclusion	13

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.

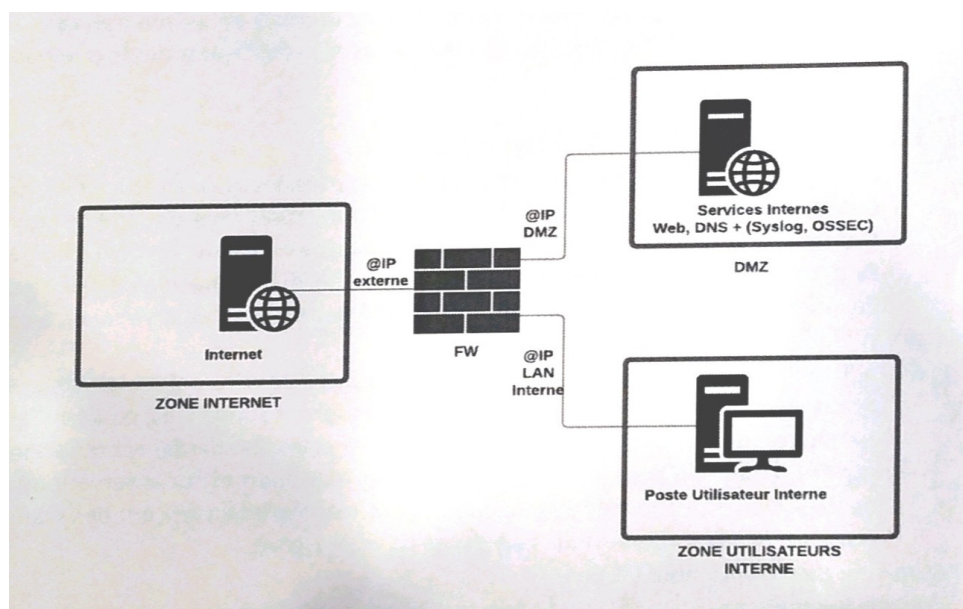


FIG. 1 : Schéma initial donné par le sujet

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 :

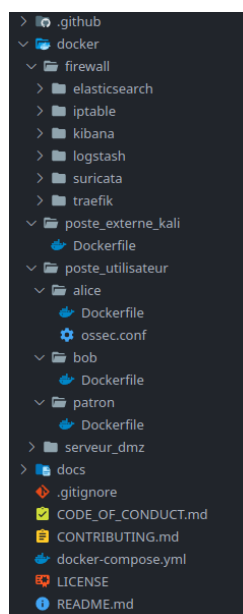


FIG. 3 : Structure de fichiers du projet

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.

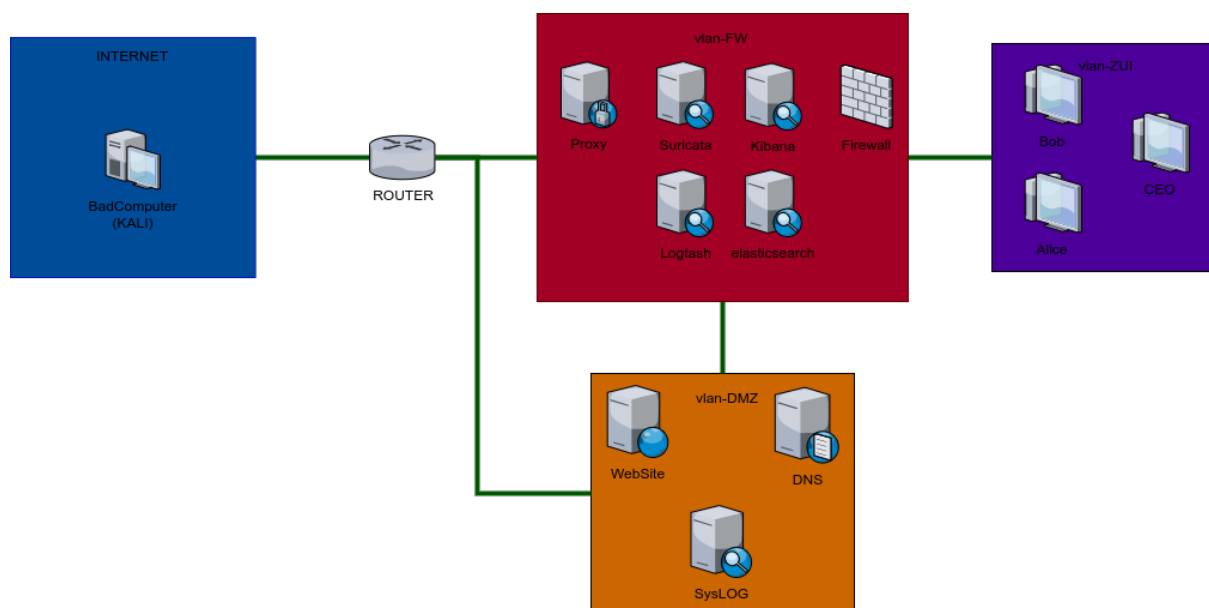


FIG. 4 : Schéma du réseau complet que nous proposons

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 :

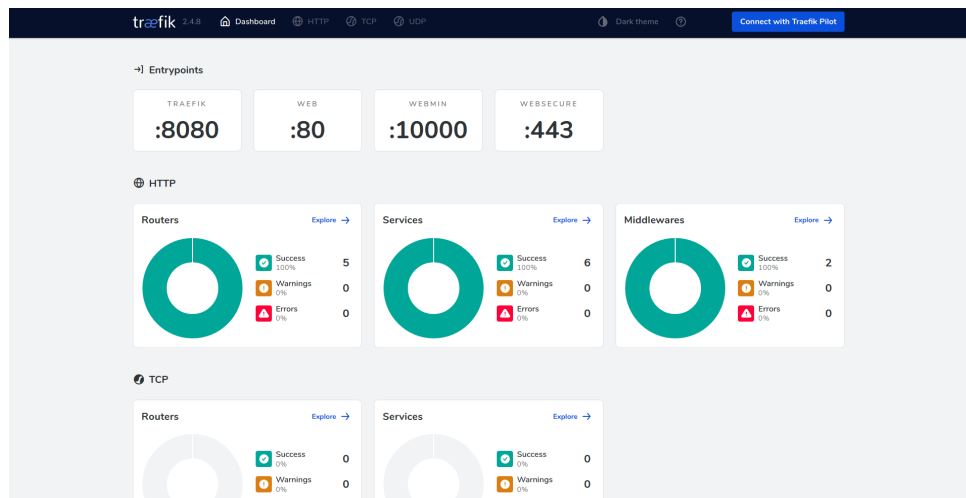
CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
072baf68cbb0	tpi-network-simulation_traefik_1	0.00%	17.31MiB / 15.32GiB	0.11%	29.5kB / 36kB	233kB / 0B	13
f9fd5d831ee4	tpi-network-simulation_kibana_1	0.28%	68.91MiB / 15.32GiB	0.44%	598kB / 231kB	0B / 0B	11
7bf04ee53a53	tpi-network-simulation_logstash_1	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 :

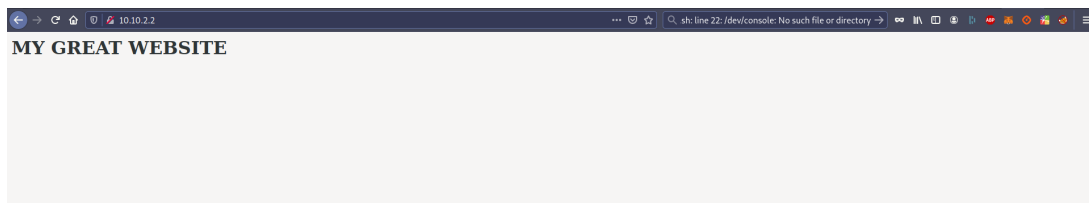
**FIG. 6 :** Le dashboard de traefik

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

All Status						
Success Warnings Errors						
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

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 événements à un serveur de collecte.

Nous allons définir les règles de filtrage pour configurer le firewall. Étant sous un système UNIX nous allons utiliser `iptables` 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 à 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

Suricata est tout d'abord un programme permettant de générer des logs en analysant un trafic réseau. Cependant on peut lui activer la fonction d'IDS (Intrusion Detection System) il devient alors un outil couteau suisse très puissant dans la détection et la défense d'un système informatique.

Un peu à la manière des autres logiciels et services que l'on a vu tout au long de ce projet, suricata peut s'installer sur n'importe quelle machine GNU/Linux et se configure facilement à l'aide de fichiers de configuration.

Pour être efficace, suricata doit observer tout le trafic entrant. On a donc deux possibilités : utiliser suricata comme passerelle, filtrant les paquets ou bien sur un TAP clonant tout le trafic.

L'avantage de l'utiliser en tant que passerelle est que l'on peut modifier ou arrêter les paquets indésirables, l'inconvénient est que cela peut ralentir le débit si la machine sur laquelle suricata tourne n'est pas assez performante. L'avantages avantages et inconvénients de l'utilisation de suricata sur un TAP sont complémentaires à la passerelle.

Les règles que l'on peut configurer pour suricata ressemblent aux suivantes :

```
1 alert http any any -> any any (msg:"Protocole HTTP détecté"; sid:1;)
2 alert tcp any any -> any any (msg:"Protocole TCP détecté"; sid:2;)
3 alert udp any any -> any any (msg:"Protocole UDP détecté"; sid:3;)
```

L'une des grandes forces de suricata est de pouvoir non seulement détecter les requêtes suspectes à l'aide des jeux de règles, mais on peut aussi tout simplement les bloquer.

Pour ce projet nous avons décidé d'utiliser un jeu de règles standard car cela correspond le plus à ce qu'un utilisateur moyen installera sur son réseau pour le protéger. Nous avons choisi d'utiliser le jeu de règles d'Emerging Threats disponible ici. Ces règles permettent de détecter la plupart des trafics liés à une activité suspecte, comme un lien vers un reverse shell par exemple.

On peut tester les fonctionnalités de suricata en mettant une alerte sur l'ensemble des ports tcp, ainsi si quelqu'un essaie d'envoyer une requête au routeur, pour voir si un port est ouvert par exemple, on pourra générer une alerte. Supposons que nous savons que le port 223 n'est jamais ouvert, on peut détecter un scan intrusif avec une requête ressemblant à :

```
1 alert tcp any any -> ROUTER_IP 223 (msg:"Tentative de scan détectée";  
  sid:4;)
```

De la même manière on va pouvoir créer tout un ensemble de règles personnalisées correspondant aux besoins de l'entreprise.

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 événements 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 événements 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 intéressants 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'événement reçus sur le dashboard :

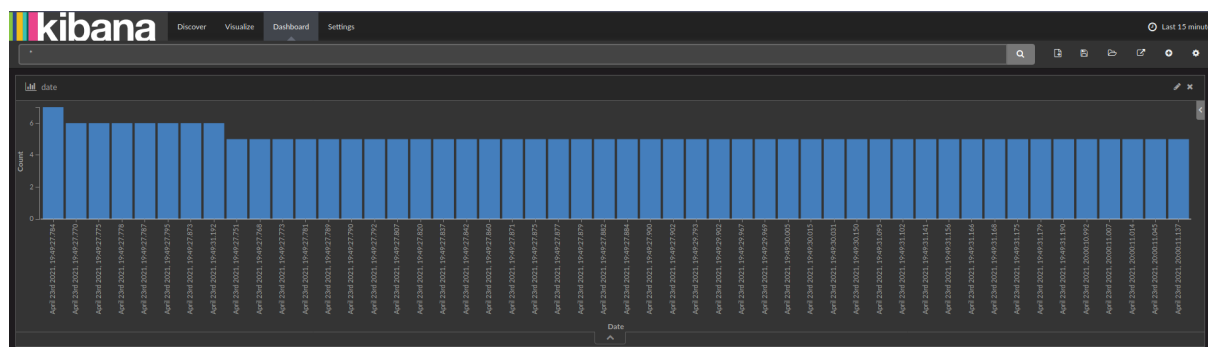


Fig. 9 : Les logs s'affichent par ordre chronologique sur le dashboard Kibana

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 PORT      STATE SERVICE  VERSION
2 80/tcp    open  http     Apache httpd 2.4.46 ((Unix))
3 |_http-csrf: Couldn't find any CSRF vulnerabilities.
4 |_http-dombased-xss: Couldn't find any DOM based XSS.
5 |_http-server-header: Apache/2.4.46 (Unix)
6 |_http-stored-xss: Couldn't find any stored XSS vulnerabilities.
7 |_http-trace: TRACE is enabled
8 443/tcp   open  ssl/http Golang net/http server (Go-IPFS json-rpc or
   InfluxDB API)
9 |_http-csrf: Couldn't find any CSRF vulnerabilities.
10 |_http-dombased-xss: Couldn't find any DOM based XSS.
11 | http-slowloris-check:
12 |   VULNERABLE:
13 |     Slowloris DOS attack
14 |       State: LIKELY VULNERABLE
15 |       IDs: CVE:CVE-2007-6750
16 |       Slowloris tries to keep many connections to the target web
   server open and hold
```

```
17 |         them open as long as possible. It accomplishes this by opening
    |         connections to
18 |         the target web server and sending a partial request. By doing
    |         so, it starves
19 |         the http server's resources causing Denial Of Service.
20 |
21 |         Disclosure date: 2009-09-17
22 |         References:
23 |         https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-6750
24 |         http://ha.ckers.org/slowloris/
25 | _http-stored-xss: Couldn't find any stored XSS vulnerabilities.
26 | _sslv2-drown:
27 |
28 | NSE: Script Post-scanning.
29 | Initiating NSE at 18:35
30 | Completed NSE at 18:35, 0.00s elapsed
31 | Initiating NSE at 18:35
32 | Completed NSE at 18:35, 0.00s elapsed
33 | Read data files from: /usr/bin/./share/nmap
34 | Service detection performed. Please report any incorrect results at
    |     https://nmap.org/submit/ .
35 | Nmap done: 1 IP address (1 host up) scanned in 595.52 seconds
```

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 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.

L'ensemble des fichiers utilisés pour ce projet est disponible en open source sur github sur le repository suivant : <https://github.com/NightlySide/TPI-Network-Simulation>.