PR07 : SIO SISR – Déploiement d'une solution de détection d'intrusion

Sommaires

Contexte :

Une entreprise nous a sollicités afin de mettre en place une solution de détection d'intrusion pour son infrastructure.

Objectifs :

Déployer un IDS pour détecter en temps réel les attaques de type DDoS, brute force et flooding, afin d'assurer la sécurité et la disponibilité des ressources critiques.

Les logs seront centralisés et analysés via le SIEM Graylog pour une meilleure visibilité et réactivité face aux incidents.

Cahier des charges :

Mise en place d'un Système de Détection d'Intrusion (IDS) :

- Procéder à l'installation et à la configuration d'une solution IDS adaptée à l'environnement (Windows ou Linux), en détaillant chaque étape de déploiement.
- S'assurer que l'IDS est opérationnel et capable de surveiller l'ensemble des flux réseau critiques de l'infrastructure.

Détection des Attaques et Gestion des Signatures :

- Configurer l'IDS pour détecter spécifiquement les attaques de type DDoS, brute force et flooding.

Centralisation des Logs avec Graylog :

- Installer et configurer la solution SIEM Graylog pour centraliser les journaux d'événements générés par l'IDS.
- Vérifier la bonne remontée des logs dans Graylog et mettre en place des tableaux de bord pour la visualisation en temps réel des alertes de sécurité.

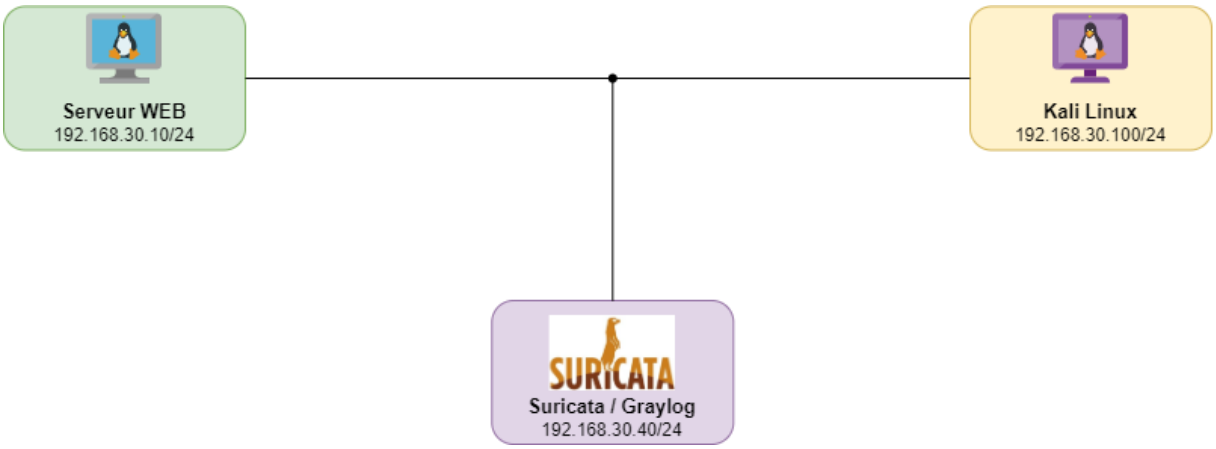
Solution :

Afin de répondre au cahier des charges, j'ai mis en place une VM sous Debian 11 hébergeant à la fois le serveur Suricata (pour la détection d'intrusion) et Graylog (pour la centralisation et l'analyse des logs).

Afin de tester l'efficacité du système, j'ai simulé diverses attaques depuis une VM Kali Linux, en utilisant Hping3 pour les attaques de type DDoS et flooding, ainsi que Hydra pour les attaques par brute force.
J'ai capturé et analysé le trafic réseau avec Wireshark pour observer les flux malveillants et valider la détection des paquets.

Enfin, j'ai configuré les règles de détection dans Suricata une par une, puis vérifié la bonne remontée des alertes dans Graylog, assurant ainsi la traçabilité et la réactivité face aux incidents de sécurité.

Schéma ASI :



Prérequis :

Une machine virtuelle Debian 11 sera déployée pour héberger à la fois Suricata (IDS) et Graylog (SIEM).
Le serveur web existant sera conservé comme cible pour les tests d'attaque.

Les machines nécessaires au projet seront mises en place :

- Une VM Kali Linux pour générer les attaques (DDoS, brute force, flooding)
- Une VM Debian 11 pour l'IDS et le SIEM
- Le serveur web, déjà opérationnel

Les interfaces réseau de toutes les machines seront configurées pour fonctionner sur le même réseau local.

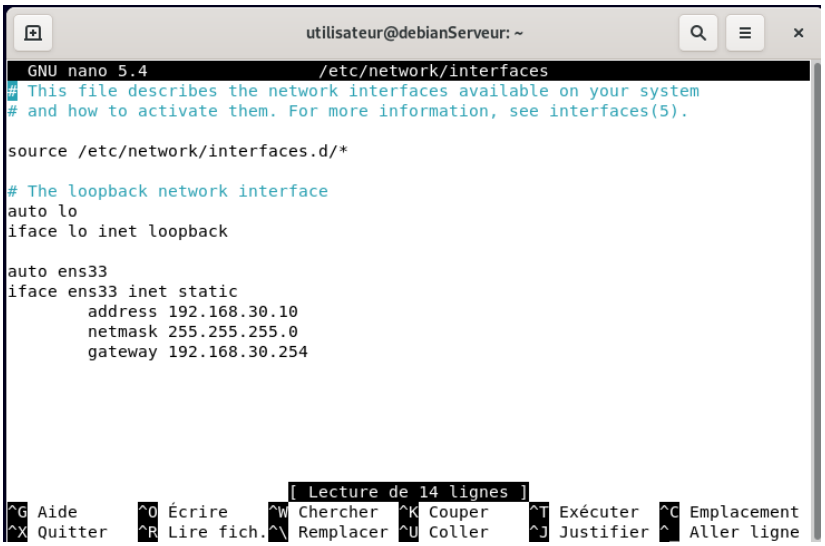
Configuration des interfaces réseaux :

On va d'abord aller dans le fichier /etc/network/interfaces afin de paramétrer notre carte réseaux avec la commande sudo nano /etc/network/interfaces.

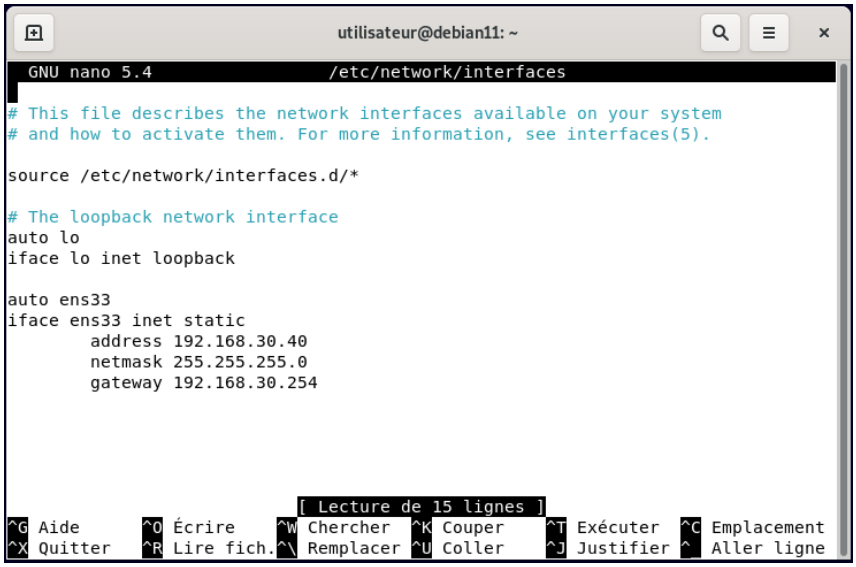
On redémarrera les interfaces avec la commande sudo systemctl restart networking.service pour que les changements prennent effet.

Il ne faut pas oublier d'activer l'accès par pont.

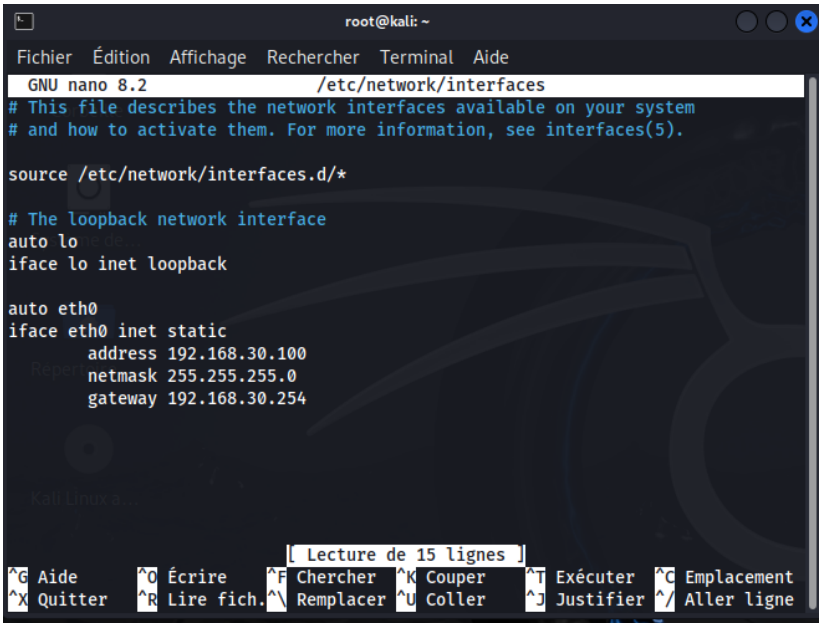
Pour notre serveur Web on lui définit comme adresse IP : 192.168.30.10



Notre serveur Suricata a pour adresse IP : 192.168.30.40



Notre VM Kali a pour adresse IP : 192.168.30.100



Installation et Configuration de l'IDS Suricata :

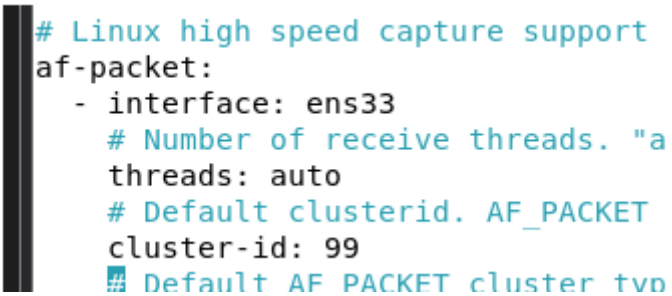
Installation du Suricata :

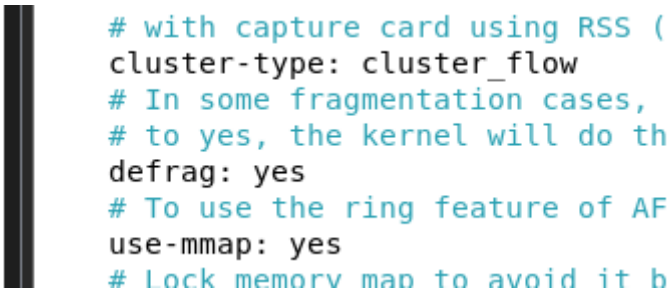
Nous allons installer Suricata et les outils nécessaires. Pour commencer, nous exécutons la commande suivante :

- sudo apt update && sudo apt install -y suricata jq

Ensuite, nous allons modifier le fichier de configuration principal :

- sudo nano /etc/suricata/suricata.yaml





```
# with capture card using RSS (
cluster-type: cluster_flow
# In some fragmentation cases,
# to yes, the kernel will do th
defrag: yes
# To use the ring feature of AF
use-mmap: yes
# Lock memory map to avoid it b
```

Configuration :

- Interface : ens33 - Surveille cette interface réseau.
- Cluster-id : 99 - Identifiant unique pour le traitement multithread.
- Cluster-type : cluster_flow - Analyse des paquets d'une même connexion.
- Defrag: yes - Reconstruit les paquets IP fragmentés.
- Use-mmap : yes - Active le mode rapide pour la lecture des paquets.
- Threads : auto - Choisit automatiquement le nombre de threads selon le processeur.

Création du fichier pour nos règles :

- sudo nano /etc/suricata/rules/custom.rules

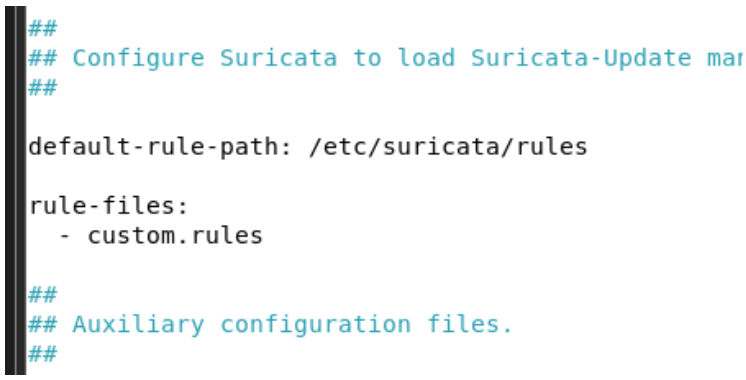
Nous allons modifier le fichier suricata.yaml pour inclure notre fichier de règles :

- sudo nano /etc/suricata/suricata.yaml

Cherchez la section des règles et ajoutez :

rule-files:

- custom.rules



```
##
## Configure Suricata to load Suricata-Update mar
##

default-rule-path: /etc/suricata/rules

rule-files:
  - custom.rules

##
## Auxiliary configuration files.
##
```

On va ensuite redémarre le service pour appliquer les changements :

- sudo systemctl restart suricata

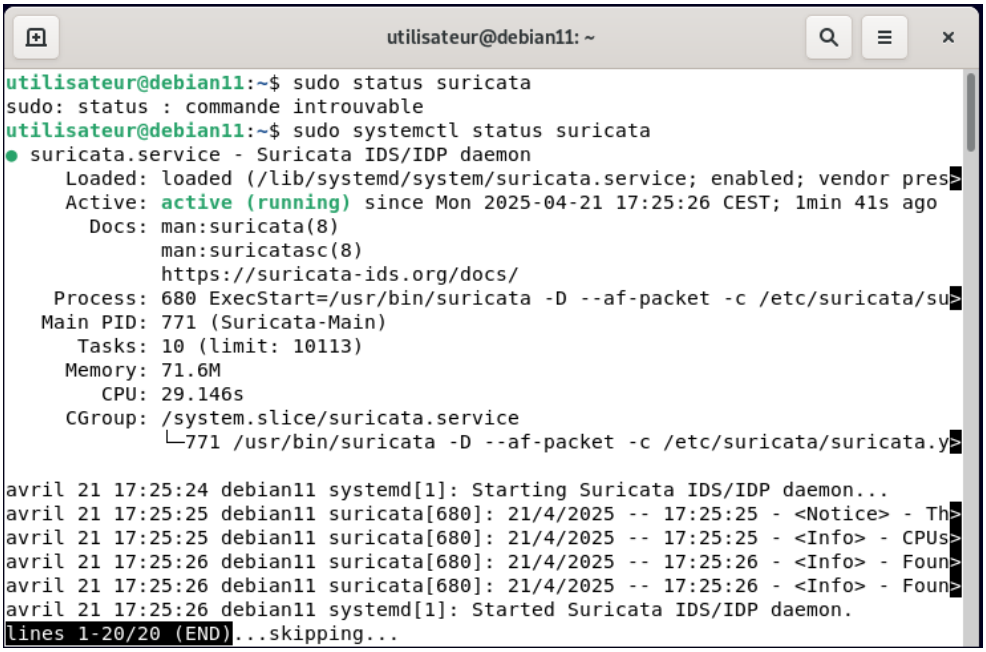
On vérifie notre configuration avec la commande :

- sudo suricata -T -c /etc/suricata/suricata.yaml -v

```
root@debian11:/etc/suricata/rules# suricata -T -c /etc/suricata/suricata.yaml -v
24/2/2025 -- 12:30:24 - <Info> - Running suricata under test mode
24/2/2025 -- 12:30:24 - <Notice> - This is Suricata version 6.0.1 RELEASE running in SYSTEM mode
24/2/2025 -- 12:30:24 - <Info> - CPUs/cores online: 4
24/2/2025 -- 12:30:24 - <Info> - fast output device (regular) initialized: fast.log
24/2/2025 -- 12:30:24 - <Info> - eve-log output device (regular) initialized: eve.json
24/2/2025 -- 12:30:24 - <Info> - stats output device (regular) initialized: stats.log
24/2/2025 -- 12:30:24 - <Info> - 1 rule files processed. 5 rules successfully loaded, 0 rules failed
24/2/2025 -- 12:30:24 - <Info> - Threshold config parsed: 0 rule(s) found
24/2/2025 -- 12:30:24 - <Info> - 5 signatures processed. 0 are IP-only rules, 4 are inspecting packet payload, 0 inspect application layer, 0 are decoder event only
24/2/2025 -- 12:30:24 - <Notice> - Configuration provided was successfully loaded. Exiting.
24/2/2025 -- 12:30:24 - <Info> - cleaning up signature grouping structure... complete
root@debian11:/etc/suricata/rules# nano custom.rules
root@debian11:/etc/suricata/rules#
```

Puis on vérifier que suricata a démarré correctement :

- sudo systemctl status suricata



Vérification du fonctionnement de suricata :

Afin de tester la configuration de Suricata nous allons créer une règle pour le ping (ICMP)

On édite notre fichier de règles :

- sudo nano /etc/suricata/rules/custom.rules

Puis on édite notre règle de détection du Ping vers le serveur Web :

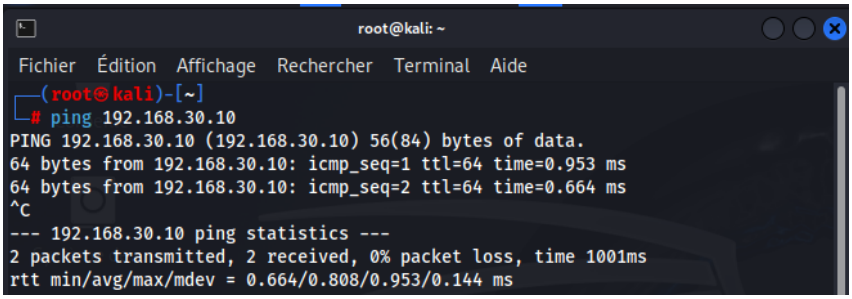
- alert icmp any any -> 192.168.30.10 any (msg:"Ping vers le serveur web"; itype:8; sid:1000001;)

```
alert icmp any any -> 192.168.30.10 any (msg:"Ping vers le serveur web"; itype:8; sid:1000001;)
```

Redémarrage de suricata pour prendre en compte les modifications :

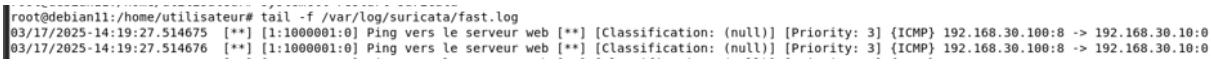
- sudo systemctl restart suricata

Puis on vérifie la détection via un test ICMP depuis la VM Kali :



On consulte ensuite les alertes en temps réel :

- tail -f /var/log/suricata/fast.log



Suricata détecte bien notre Ping vers le serveur WEB depuis notre VM Kali.

On va donc pouvoir passer à la configuration de l'ensemble de nos règles.

Création des règles pour les différentes attaques :

Lancement d'attaques DDoS, brute force et flooding via Kali Linux, Capture et analyse du trafic avec Wireshark :

Pour établir nos règles concernant les différentes attaques, nous utiliserons Wireshark afin d'analyser les trames et éditer nos règles en conséquence.

A l'aide de Kali est des logiciels déjà présent (Hping 3 et Hydra) nous allons réaliser des attaques sur le serveur WEB :

Attaques ICMP Flood :

```
┌──(root㉿kali)-[/home/utilisateur]
└─# hping3 -1 --flood -V 192.168.30.10

using eth0, addr: 192.168.30.100, MTU: 1500
HPING 192.168.30.10 (eth0 192.168.30.10): icmp mode set, 28 headers + 0 data
bytes
hping in flood mode, no replies will be shown


^C
--- 192.168.30.10 hping statistic ---
1043406 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

- hping3 -1 --flood -v 192.168.30.10 :

- -1 : Envoie des paquets ICMP.
- --flood : Active l'envoi rapide et continu de paquets, sans attendre de réponse.
- -v : Mode verbeux, permet d'afficher les paquets envoyés.
- 192.168.30.10 : Adresse IP de la cible (le serveur web).

Objectif : Simuler une attaque ICMP flood visant à surcharger le serveur.

Fichier Wireshark :

No.	Time	Source	Destination	Protocol	Length	Info
2637	4.164609901	192.168.30.100	192.168.30.10	ICMP	60	Echo (ping) request id=0x232f, seq=0/0, ttl=64 (reply in 2699)
2638	4.164609985	192.168.30.100	192.168.30.10	ICMP	60	Echo (ping) request id=0x232f, seq=256/1, ttl=64 (reply in 2700)
2639	4.164610041	192.168.30.100	192.168.30.10	ICMP	60	Echo (ping) request id=0x232f, seq=512/2, ttl=64 (reply in 2701)
2640	4.164610098	192.168.30.100	192.168.30.10	ICMP	60	Echo (ping) request id=0x232f, seq=768/3, ttl=64 (reply in 2702)

Attaques TCP SYN Flood :

```
┌──(root㉿kali)-[/home/utilisateur]
└─# hping3 -S --flood -p 80 192.168.30.10
HPING 192.168.30.10 (eth0 192.168.30.10): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 192.168.30.10 hping statistic ---
761920 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

- hping3 -S --flood -p 80 192.168.30.10 :

- -S : Envoi des paquets TCP avec le flag SYN (demande de connexion).
- --flood : Envoi massif et rapide de paquets.
- -p 80 : Cible le port 80, utilisé par le service HTTP.
- 192.168.30.10 : Adresse IP de la cible (le serveur web).

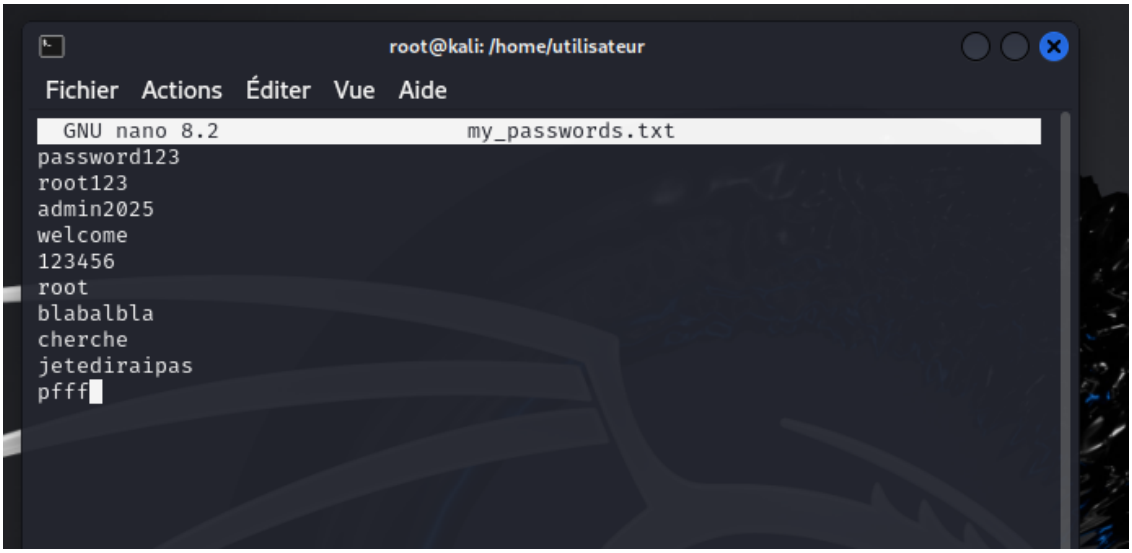
Objectif : Simuler une attaque SYN flood sur le port web pour provoquer une surcharge du service.

Fichier Wireshark :

37	4.509759755	192.168.30.100	192.168.30.10	TCP	60 1345 → 80 [SYN] Seq=0 Win=512 Len=0
38	4.509812303	192.168.30.10	192.168.30.100	TCP	58 80 → 1345 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
39	4.509925020	192.168.30.100	192.168.30.10	TCP	60 1346 → 80 [SYN] Seq=0 Win=512 Len=0
40	4.509930957	192.168.30.10	192.168.30.100	TCP	58 80 → 1346 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

Attaques Brute Force :

Afin de réduire le temps de recherche nous avons créé un fichier contenant une série de mot de passe à tester :



Nous procédons ensuite à l'exécution des attaques :

- hydra -l root -P /home/utilisateur/my_passwords.tkt ssh://192.168.30.10

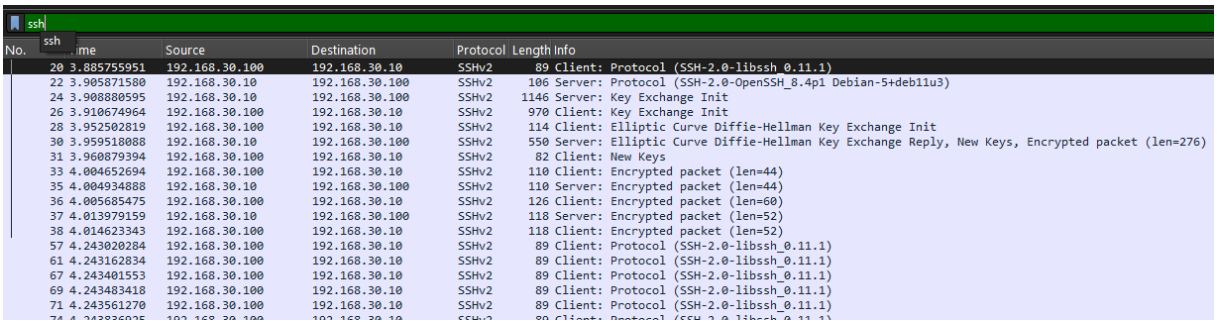
• -l root : Nom d'utilisateur ciblé (ici, "root").

• -P /home/utilisateur/my_passwords.tkt : Fichier contenant une liste de mots de passe à tester.

• ssh://192.168.30.10 : Protocole et adresse IP de la cible (port SSH par défaut : 22).

Objectif : Tester la robustesse du service SSH face aux tentatives d'accès non autorisées via brute force.

Fichier Wireshark pour le SSH :

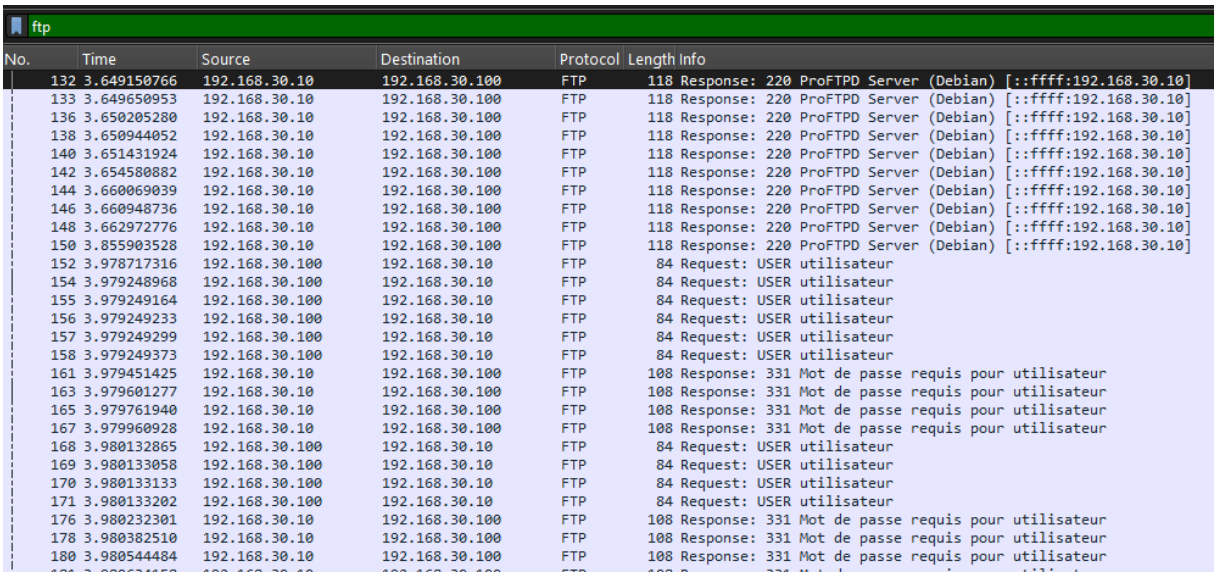


- hydra -l root -P /home/utilisateur/my_passwords.tkt ftp://192.168.30.10

- -l root : Nom d'utilisateur ciblé.
- -P /home/utilisateur/my_passwords.tkt : Liste des mots de passe à essayer.
- ftp://192.168.30.10 : Protocole FTP ciblé sur le serveur (port par défaut : 21).

Objectif : Simuler une attaque brute force sur le service FTP afin d'évaluer sa capacité à résister aux tentatives de connexion multiples.

Fichier Wireshark pour le FTP :



Création des règles suricata en fonction des logs de Wireshark :

1) alert icmp any any -> any any (msg:"Possible ICMP Flood attack"; itype:8; threshold:type both, track by_dst, count 50, seconds 10; classtype:attempted-dos; sid:1000005; rev:1;)

Explication :

- alert icmp any any -> any any :
    - icmp : Cette règle surveille les paquets ICMP utilisés pour les pings.
    - any any -> any any : Cela signifie que la règle s'applique à tout trafic ICMP, peu importe l'adresse source ou de destination.
- msg:"Possible ICMP Flood attack" :
    - Ce message est généré lorsqu'un paquet est détecté, il indique qu'un potentiel flood ICMP (attaque par inondation de pings) a été observé.
- itype:8 :
    - L'ICMP de type 8 est une requête Echo (ping). Cela signifie que la règle surveille les pings entrants.
- threshold:type both, track by_dst, count 50, seconds 10 :
    - type both : Cette option indique que la règle s'applique à la fois aux paquets entrants et sortants.
    - track by_dst : Suivi du nombre de paquets par destination (pour détecter des attaques visant une cible spécifique).
    - count 50, seconds 10 : Si 50 paquets ICMP sont envoyés à la même destination en moins de 10 secondes, une alerte sera générée.
- classtype:attempted-dos :
    - Cette règle est classée comme une tentative d'attaque DoS (Denial of Service), qui vise à rendre un service indisponible (ici, par saturation de requêtes ping).
- sid:1000005 :
    - C'est l'ID unique de la règle, utilisée pour identifier cette alerte dans les journaux.
- rev:1 :
    - Révision de la règle, indiquant la version de celle-ci.

2) alert tcp any any -> any 80 (msg:"Possible Syn Flood detected"; flags:S; threshold:type both, track by_dst, count 30, seconds 10; classtype:attempted-dos; sid:1000002; rev:1;)

Explication :

- alert tcp any any -> any 80 :
    - Cette règle surveille le trafic TCP sur le port 80 (utilisé pour HTTP, donc pour les sites web).

  - any any indique que cette règle est appliquée à tout paquet TCP, peu importe les adresses source ou de destination, tant que la destination est le port 80.

- msg:"Possible Syn Flood detected" :
  - Ce message alerte qu'une attaque SYN Flood a été détectée. Il s'agit d'une attaque où un attaquant envoie un grand nombre de paquets SYN pour saturer la capacité du serveur à établir des connexions.
- flags:S :
  - Filtre pour les paquets SYN. Ce sont des paquets utilisés pour initier une connexion TCP.
- threshold:type both, track by_dst, count 30, seconds 10 :
  - type both : Applique la règle aux paquets entrants et sortants.
  - track by_dst : Suivi du nombre de paquets par destination (en l'occurrence, le serveur web).
  - count 30, seconds 10 : Si 30 paquets SYN sont envoyés à la même destination (port 80) en moins de 10 secondes, une alerte sera générée.
- classtype:attempted-dos :
  - Cela indique que la règle se rapporte à une attaque DoS.
- sid:1000002 :
  - ID unique de la règle.
- rev:1 :
  - Révision de la règle.

3) alert tcp any any -> any 22 (msg:"Potential SSH Brute Force"; flow:to_server; threshold:type both, track by_src, count 10, seconds 60; classtype:attempted-recon; sid:1000003; rev:1;)

Explication :

- alert tcp any any -> any 22 :
  - La règle cible le trafic TCP destiné au port 22, qui est le port par défaut pour le service SSH.
- msg:"Potential SSH Brute Force" :
  - Ce message alerte sur une tentative de brute force SSH, où l'attaquant essaie de deviner le mot de passe du compte root ou d'autres comptes.
- flow:to_server :
  - Cette option surveille uniquement les paquets envoyés au serveur (c'est-à-dire vers le port 22 du serveur SSH).

- threshold:type both, track by_src, count 10, seconds 60 :
    - track by_src : Suivi du nombre de tentatives par source (pour détecter plusieurs tentatives d'un même attaquant).
    - count 10, seconds 60 : Si 10 paquets sont envoyés en moins de 60 secondes à la même destination (serveur SSH), une alerte est générée.
- classtype:attempted-recon :
    - Cela indique une tentative de reconnaissance (l'attaquant essaie de pénétrer dans le système).
- sid:1000003 :
    - ID unique de la règle.
- rev:1 :
    - Révision de la règle.

4) alert tcp any any -> any 21 (msg:"Potential FTP Brute Force"; flow:to_server; threshold:type both, track by_src, count 10, seconds 60; classtype:attempted-recon; sid:1000004; rev:1;)

Explication :

- alert tcp any any -> any 21 :
    - Cette règle surveille le trafic TCP destiné au port 21, utilisé pour le service FTP.
- msg:"Potential FTP Brute Force" :
    - Le message alerte sur une tentative de brute force pour accéder au service FTP.
- flow:to_server :
    - Suivi des paquets envoyés au serveur FTP (vers le port 21).
- threshold:type both, track by_src, count 10, seconds 60 :
    - track by_src : Suivi du nombre de tentatives par source (cela permet de détecter un attaquant qui fait plusieurs tentatives depuis une seule adresse).
    - count 10, seconds 60 : Si 10 tentatives de connexion échouées sont envoyées au serveur FTP en 60 secondes, une alerte est générée.
- classtype:attempted-recon :
    - Indique que la règle se réfère à une tentative de reconnaissance (l'attaquant cherche à identifier un mot de passe pour accéder au service FTP).
- sid:1000004 :
    - ID unique de la règle.
- rev:1 :

- Révision de la règle.

Vérification des règles mises en place sur Suricata :

- Tests de détection : Nous allons refaire les mêmes attaques (DDoS, brute force et flooding) que celles réalisées lors de l'écoute du réseau avec Wireshark pour valider le bon fonctionnement des règles.

Vérification de la remontée des alertes en temps réel :

- tail -f /var/log/suricata/fast.log

```
root@debian11:/home/utilisateur# tail -f /var/log/suricata/fast.log
03/17/2025-14:19:27.514685  [**] [1:1000001:0] Ping vers le serveur web [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.30.100:8 -> 192.168.30.10:0
03/17/2025-14:19:27.514685  [**] [1:1000001:0] Ping vers le serveur web [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.30.100:8 -> 192.168.30.10:0
03/17/2025-14:19:27.514686  [**] [1:1000001:0] Ping vers le serveur web [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.30.100:8 -> 192.168.30.10:0
03/17/2025-14:19:27.514686  [**] [1:1000001:0] Ping vers le serveur web [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.30.100:8 -> 192.168.30.10:0
03/17/2025-14:19:27.514687  [**] [1:1000001:0] Ping vers le serveur web [**] [Classification: (null)] [Priority: 3] {ICMP} 192.168.30.100:8 -> 192.168.30.10:0
03/17/2025-14:29:17.014195  [**] [1:1000005:1] Possible ICMP Flood attack [**] [Classification: Attempted Denial of Service] [Priority: 2] {ICMP} 192.168.30.100:8 -> 192.168.30.10:0
03/17/2025-14:29:27.015419  [**] [1:1000005:1] Possible ICMP Flood attack [**] [Classification: Attempted Denial of Service] [Priority: 2] {ICMP} 192.168.30.100:8 -> 192.168.30.10:0
03/17/2025-14:51:24.796870  [**] [1:1000002:1] Possible Syn Flood detected [**] [Classification: Attempted Denial of Service] [Priority: 2] {TCP} 192.168.30.100:1769 -> 192.168.30.10:80
03/17/2025-14:51:34.796267  [**] [1:1000002:1] Possible Syn Flood detected [**] [Classification: Attempted Denial of Service] [Priority: 2] {TCP} 192.168.30.100:30338 -> 192.168.30.10:80
03/17/2025-15:29:57.318530  [**] [1:1000003:1] Potential SSH Brute Force [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.30.100:41548 -> 192.168.30.10:22
03/17/2025-15:32:17.669943  [**] [1:1000004:1] Potential FTP Brute Force [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 192.168.30.100:59954 -> 192.168.30.10:21
```

Suricata détecte bien nos différentes attaques vers le serveur WEB depuis notre VM Kali.

Installation et Configuration de Graylog (SIEM) :

Installation de Graylog :

Nous allons installer Graylog 6.1, ainsi que ses composants nécessaires (MongoDB 6 / OpenSearch et OpenJDK 17) sur le même serveur que Suricata :

Mise a jour des paquets présents dans les dépôts :

- sudo apt-get update

Puis installer les outils nécessaires pour la suite de l'installation :

- sudo apt-get install curl lsb-release ca-certificates gnupg2 pwgen

Passons maintenant à l'installation de MongoDB :

Installation de la clé GPG correspondant au dépôt MongoDB :

- curl -fsSL https://www.mongodb.org/static/pgp/server-6.0.asc | sudo gpg -o /usr/share/keyrings/mongodb-server-6.0.gpg –dearmor

Ajout du dépôt de MongoDB :

- echo "deb [signed-by=/usr/share/keyrings/mongodb-server-6.0.gpg] http://repo.mongodb.org/apt/debian bullseye/mongodb-org/6.0 main" | sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list

Installation de la dépendance libssl1.1 :

- wget http://archive.ubuntu.com/ubuntu/pool/main/o/openssl/libssl1.1_1.1.1f-1ubuntu2.23_amd64.deb
- sudo dpkg -i libssl1.1_1.1.1f-1ubuntu2.23_amd64.deb

Nous pouvons maintenant lancer l'installation :

- sudo apt-get install -y mongodb-org

Nous allons ensuite relancer le service MongoDB et activez son démarrage automatique au lancement du serveur Debian.

- sudo systemctl daemon-reload
- sudo systemctl enable mongod.service
- sudo systemctl restart mongod.service
- sudo systemctl --type=service --state=active | grep mongod

Nous allons pouvoir passer à l'installation de OpenSearch :

Nous allons commencer par ajouter la clé de signature pour les paquets OpenSearch :

- curl -o- https://artifacts.opensearch.org/publickeys/opensearch.pgp | sudo gpg --dearmor --batch --yes -o /usr/share/keyrings/opensearch-keyring

Puis nous allons ajouter le dépôt OpenSearch :

- echo "deb [signed-by=/usr/share/keyrings/opensearch-keyring] https://artifacts.opensearch.org/releases/bundle/opensearch/2.x/apt stable main" | sudo tee /etc/apt/sources.list.d/opensearch-2.x.list

Mise à jour des paquets présents dans les dépôts :

- sudo apt-get update

On va pouvoir installer OpenSearch en prenant soin de lui définir un compte Admin avec un mot de passe d'au moins 8 caractères :

- sudo env OPENSEARCH_INITIAL_ADMIN_PASSWORD=Thomas2025! apt-get install opensearch

On va maintenant procéder à la configuration minimale pour faire fonctionner le service :

- sudo nano /etc/opensearch/opensearch.yml

Et configurer les options suivantes :

- cluster.name: graylog : ce paramètre nomme le cluster OpenSearch avec le nom "graylog".
- node.name: ${HOSTNAME} : le nom du nœud est défini dynamiquement pour correspondre à celui de la machine Linux locale. Même si nous n'avons qu'un seul nœud, il est important de le nommer correctement.
- path.data: /var/lib/opensearch : ce chemin spécifie où OpenSearch stocke ses données sur la machine locale, en l'occurrence dans "/var/lib/opensearch".
- path.logs: /var/log/opensearch : ce chemin définit où les fichiers journaux d'OpenSearch sont stockés, ici dans "/var/log/opensearch".
- discovery.type: single-node : ce paramètre configure OpenSearch pour fonctionner avec un nœud unique, d'où le choix de l'option "single-node".
- network.host: 127.0.0.1 : cette configuration fait qu'OpenSearch écoute uniquement sur son interface de boucle locale, ce qui est suffisant puisqu'il est sur le même serveur que Graylog.
- action.auto_create_index: false : en désactivant la création automatique d'index, OpenSearch ne créera pas d'index automatiquement lorsqu'un document est envoyé sans index existant.
- plugins.security.disabled: true : cette option désactive le plugin de sécurité d'OpenSearch, ce qui signifie qu'il n'y aura ni authentification, ni gestion des accès, ni chiffrement des communications. Ce paramétrage permet de gagner du temps pour effectuer la mise en place de Graylog, mais il est à éviter en production

Nous allons devoir configurer Java Virtual Machine utilisé par OpenSearch afin d'ajuster la quantité de mémoire que peut utiliser ce service.

Nous allons éditez le fichier de configuration suivant :

- sudo nano /etc/opensearch/jvm.options

Afin d'allouer assez de ressource (ici 4Go) nous allons modifier ces deux lignes :

- Xms1g par –Xms4g
- Xms1g par –Xms4g

Nous devons vérifier la configuration du paramètre "max_map_count" au niveau du noyau Linux. Il définit la limite des zones de mémoire mappées par processus, afin de répondre aux besoins de notre application.

On utilise la commande :

- cat /proc/sys/vm/max_map_count

Afin de vérifier d'avoir la valeur 262144 dans vm.max_map_count=262144 ce qui est bien notre cas.

On va pouvoir ensuite activer le démarrage automatique de Opensearch et lancer le service :

- sudo systemctl daemon-reload
- sudo systemctl enable opensearch
- sudo systemctl restart opensearch

Nous allons enfin pouvoir procéder à l'installation de Graylog :

On va télécharger puis installer Graylog avec les commandes suivantes :

- wget https://packages.graylog2.org/repo/packages/graylog-6.1-repository_latest.deb
- sudo dpkg -i graylog-6.1-repository_latest.deb
- sudo apt-get update
- sudo apt-get install graylog-server

Quand c'est fait, nous devons apporter des modifications à la configuration de Graylog avant de chercher à le lancer.

Commençons par configurer ces deux options :

- password_secret : ce paramètre sert à définir une clé utilisée par Graylog pour sécuriser le stockage des mots de passe utilisateurs (dans l'esprit d'une clé de salage). Cette clé doit être unique et aléatoire.
- root_password_sha2 : ce paramètre correspond au mot de passe de l'administrateur par défaut dans Graylog. Il est stocké sous forme d'un hash SHA-256.

Nous allons commencer par générer une clé de 96 caractères pour le paramètre password_secret :

- pwgen -N 1 -s 96

Mot de passe généré par la commande :

wVSGYwOmwBIDmtQvGzSuBevWoXe0MWpNWCzhorBfvMMhia2zIjHguTbfl4uXZJdHOA0EEb1sXJTZKINhIIBm3V57vwfQV59

On va copier la valeur retournée, puis ouvrez le fichier de configuration de Graylog :

- sudo nano /etc/graylog/server/server.conf

On va coller la clé au niveau du paramètre password_secret

Ensuite, nous devons définir le mot de passe du compte "admin" créé par défaut.
Dans le fichier de configuration, c'est le hash du mot de passe qui doit être stocké, ce qui implique de le calculer. L'exemple ci-dessous permet d'obtenir le hash du mot de passe "PuitsDeLogs@" :

- echo -n "PuitsDeLogs@" | shasum -a 256

Mot de passe généré par la commande :

6b297230efaa2905c9a746fb33a628f4d7aba4fa9d5c1b3daa6846c68e602d71

On va copier la valeur obtenue en sortie.

Ouvrir de nouveau le fichier de configuration de Graylog :

- sudo nano /etc/graylog/server/server.conf

Et on va coller la valeur au niveau de l'option root_password_sha2

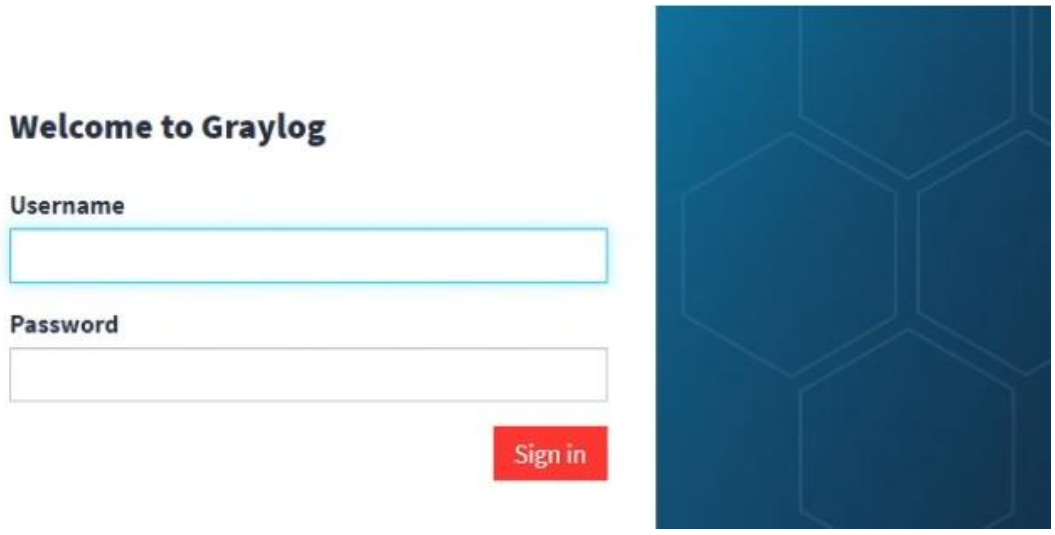
On profite d'être dans le fichier de configuration pour configurer l'option nommée "http_bind_address". On indique "0.0.0.0:9000" pour que l'interface web de Graylog soit accessible sur le port 9000, via n'importe quelle adresse IP du serveur.

Puis, on va configurer l'option "elasticsearch_hosts" avec la valeur "http://127.0.0.1:9200" pour déclarer notre instance locale OpenSearch.

On va maintenant utiliser cette commande pour activer Graylog et qu'il démarre automatiquement au prochain démarrage :

- sudo systemctl enable --now graylog-server

Nous pouvons maintenant nous connecter à Graylog via notre navigateur !



Remonté des log suricata sur graylog via Filebeat :

Étant donné que le lab est déployé sur des machines virtuelles avec des ressources limitées, nous avons choisi d'utiliser Filebeat pour la remontée des logs, car il est plus léger et mieux adapté à un environnement contraint que l'utilisation directe d'Elasticsearch

Utilisation de filebeat

FileBeat est une alternative légère à Logstash et peut envoyer directement les logs à Graylog sans passer par Elasticsearch.

1. Vérifier l'entrée FileBeat dans Graylog

Graylog doit être prêt à recevoir les logs de FileBeat via une entrée (input).

Aller dans Graylog (via l'interface web) :

1. Aller dans System → Inputs
2. Regarder qu'un Beats Input (FileBeat) est configuré et en écoute
3. Noter le port d'écoute (exemple : 5044)

Si l'entrée n'existe pas, nous allons la crée :

- Type : Beats
- Port : 5044 (ou un autre disponible)
- Bind Address : 0.0.0.0
- Lance l'entrée

2. Configurer FileBeat pour envoyer vers Graylog

Dans FileBeat, nous allons éditer le fichier de configuration :

- sudo nano /etc/filebeat/filebeat.yml

Modifie la sortie pour Graylog :

- output.logstash:

  hosts: ["127.0.0.1:5044"]

On va redémarrer FileBeat :

- sudo systemctl restart filebeat

Vérifier que FileBeat envoie bien les logs :

- sudo filebeat test output

On doit voir "connection successful".

3. Vérifier les logs JSON (ex: EVE.log de Suricata)

Si les logs sont en JSON (ex: eve.json de Suricata) comme dans notre configuration :

```
2 packets transmitted, 2 received, 0% packet loss, time 1015ms
rtt min/avg/max/mdev = 0.703/0.737/0.772/0.034 ms
root@debian11:/home/utilisateur# cat /etc/filebeat/filebeat.yml | grep -A 10 "filebeat.inputs"
filebeat.inputs:
  - type: log
    paths:
      - /var/log/suricata/eve.json
    json.keys_under_root: true
    json.add_error_key: true

# Each - is an input. Most options can be set at the input level, so
# you can use different inputs for various configurations.
# Below are the input-specific configurations.

root@debian11:/home/utilisateur#
```

Il faut activer l'analyse JSON dans FileBeat :

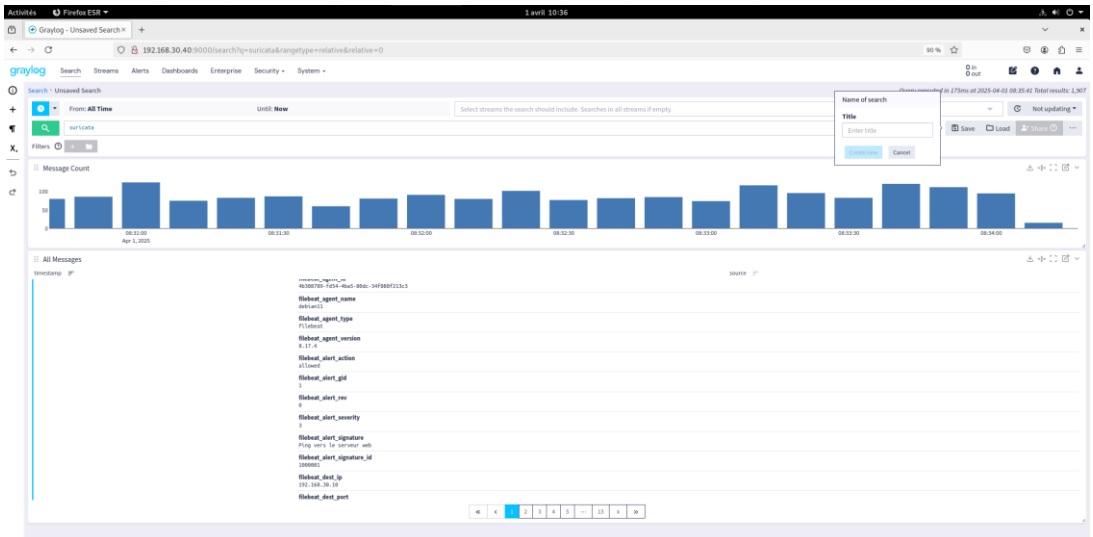
Ajouter cette config dans /etc/filebeat/filebeat.yml :

```
filebeat.inputs:
 - type: filestream
   id: suricata-logs
   paths:
    - /var/log/suricata/eve.json
   json.keys_under_root: true
   json.add_error_key: true
```

Puis redémarrer FileBeat :

```
sudo systemctl restart filebeat
```

Nous voyons actuellement un grand nombre de logs :



On va donc en désactiver une partie !

1. Désactiver les logs de monitoring de FileBeat

Actuellement, FileBeat envoie ses propres logs internes en plus des logs Suricata.

Pour désactiver ça :

1. Éditer le fichier de configuration :

- sudo nano /etc/filebeat/filebeat.yml

Ajoute ces lignes (ou modifie si elles existent) :

- logging.level: error
- monitoring.enabled: false

Redémarre FileBeat :

sudo systemctl restart filebeat

Test de la remontée des logs

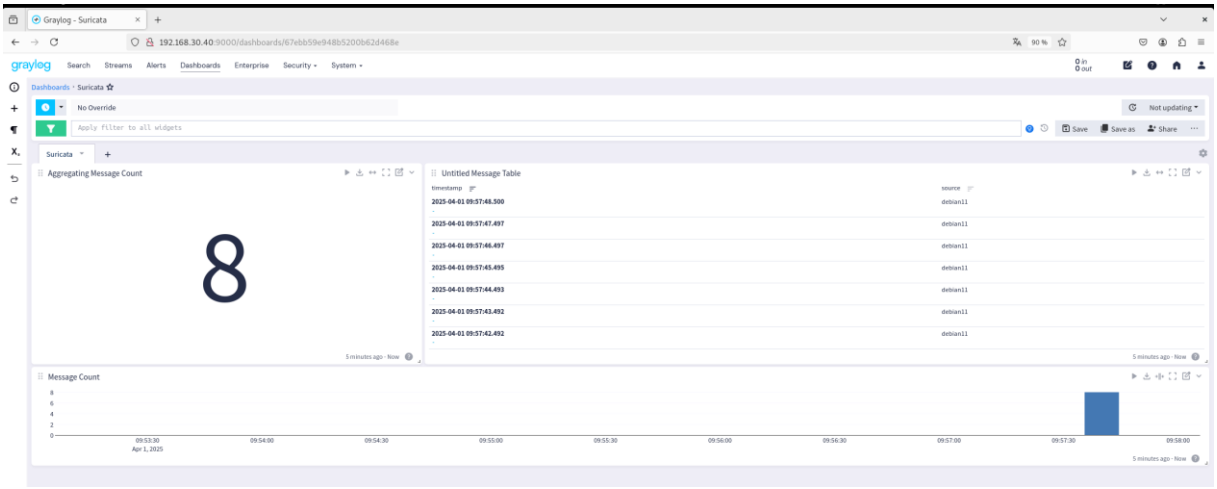
Afin de tester la remontée des logs, nous allons effectuer un ping vers notre serveur Web :

On peut voir que nous avons bien la remontée de notre Ping.

Configuration du Dashboard pour la visualisation des alertes :

Nous pouvons maintenant configurer notre tableau de bord afin de faire remonter les informations que l'on souhaite.

Ici, les pings vers le serveur web :



Nous effectuerons la même démarche pour l'ensemble de nos alertes !

Conclusion :

La mise en place d'un IDS avec Suricata et la centralisation des logs via Graylog permet de détecter en temps réel les attaques DDoS, brute force et flooding. Après avoir simulé des attaques et vérifié la remontée des alertes, j'ai confirmé l'efficacité du système pour assurer la sécurité et la disponibilité des ressources critiques.

Le tableau de bord Graylog a été configuré pour offrir une visualisation claire et en temps réel des alertes de sécurité, permettant ainsi une gestion rapide et efficace des incidents. Ce système offre une surveillance optimale et une réactivité rapide face aux menaces.