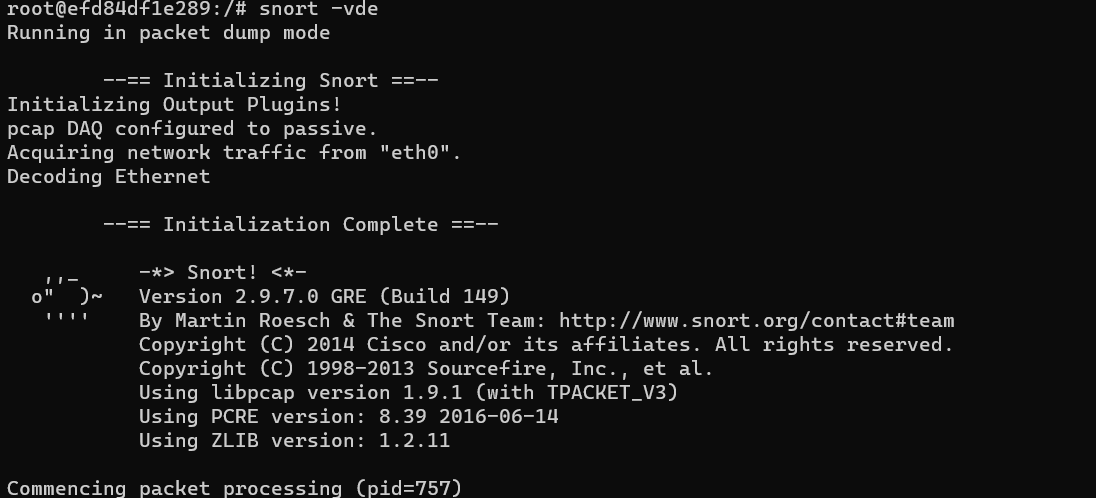
**TD 1 : Système de détection d’intrusions :**

**Partie 1 :**

1. Lorsqu’on lance snort avec les options -vde, on obtient la sortie suivante :



Dès qu’on commence à utiliser l’un des 2 sites web, snort commence à capturer les paquets et à les afficher :

Une image contenant texte

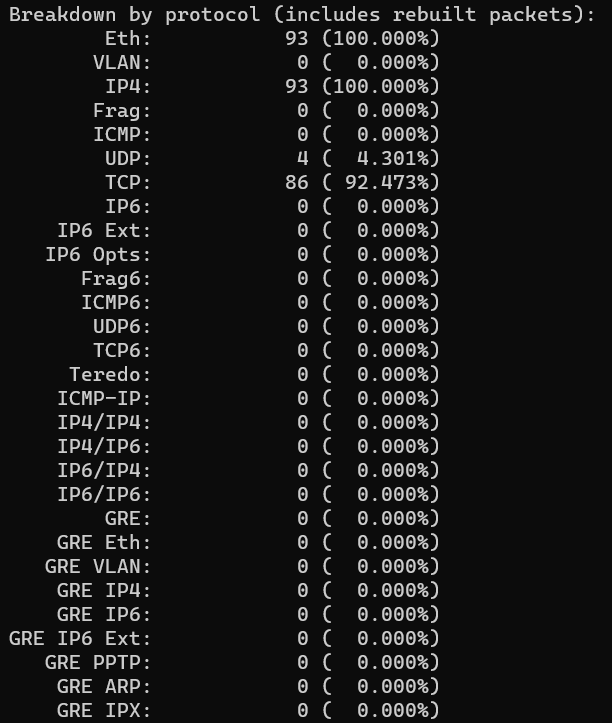
Description générée automatiquement

On observe que les paquets sont lisibles directement en clair.

1. Lorsqu’on arrête snort, il nous fournit des statistiques sur les paquets analysés. Ici il n’en a pas filtré un seul.

Une image contenant texte

Description générée automatiquement



1. Allons lire les options de la ligne de commande sur le manuel en ligne : <http://books.gigatux.nl/mirror/snortids/0596006616/snortids-CHP-3-SECT-3.html>.

**-v** : Option “verbose”, affiche tous les paquets sur la console.

Sans cette option : Le résultat est le même parce qu’on utilise snort en mode packet logging

**-d** : Affiche les charges utiles des paquets.

Sans cette option :

Une image contenant texte

Description générée automatiquement

**-e** : Affiche les information correspondants à la couche de liaison (adresse MAC).

Sans cette option :

Une image contenant texte

Description générée automatiquement

1. Lorsqu’on lance une image docker en mode port-forwarding comme indiqué dans le TD, la pile réseau n’est pas entièrement initialisée. Il est donc nécessaire de lancer l’image normalement. On obtient donc une image qui répond aux pings :

Une image contenant texte

Description générée automatiquement

(On note au passage que le délai de réponse est 50 fois plus élevé que lors d’un ping vers l’hôte local).

Lorsque j’ai lancé snort en mode sniffer sur le docker configuré sur WSL, il n’est pas parvenu à décoder correctement les paquets ICMP, et ne détectait rien du tout.

J’ai donc recommencé cette question sur un docker installé sur une machine virtuelle Ubuntu.

Depuis docker, j’ai lancé : **snort -l log -dve icmp**

On ping ensuite l’image docker:

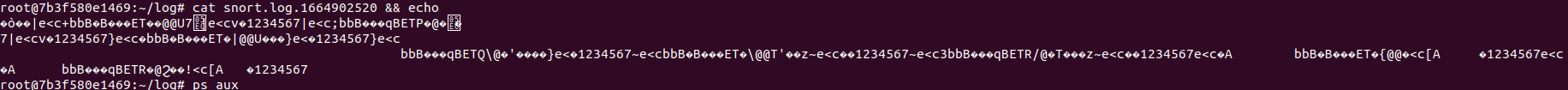
Une image contenant texte

Description générée automatiquement

(ça fonctionne cette fois-ci)

Evidemment, snort n’affiche rien à l’écran puisqu’il sauvegarde tout dans le fichier de log.

Et enfin on lit le fichier de sortie de snort, lui aussi en binaire :



On constate qu’il est illisible tel quel

Pour afficher le contenu du fichier : **snort -dvr log/snort.log.1664902520**

Il s’agit bien de paquets ICMP :

Une image contenant texte

Description générée automatiquement

1. Pour logger les paquets au format binaire (au format tcpdump) : **snort -b icmp**

Si on devait capturer beaucoup de paquets, la sauvegarde sous ce format permettrait de gagner en performances et d’avoir des fichiers plus petits.

1. Pour afficher le fichier binaire, on peut utiliser snort : **snort -r /var/log/snort/snort.log.1664903854**

On obtient le même résultat que précédemment :

Une image contenant texte

Description générée automatiquement

Mais on peut aussi procéder avec la commande *tcpdump*, qu’il faut au préalable installer :

**tcpdump -r /var/log/snort/snort.log.1664903854** (ajouter l’option *-X* pour avoir le contenu des paquets au format hexadécimal)

Une image contenant texte

Description générée automatiquement

**Partie 2 :**

1. Pour commencer, je fais en sorte qu’il soit possible de se connecter en SSH à mon docker. Pour cela, j’installe le paquet **openssh-server**. Je change le mot de passe **root** avec la commande **passwd**. J’autorise la connexion de l’utilisateur root dans le fichier **/etc/ssh/sshd\_config** (**PermitRootLogin yes**). Lorsque j’essaie de me connecter, je constate que docker ne veut pas qu’on écoute sur le port 22. Je change donc le port du serveur SSH à **1022** dans le même fichier de configuration. Je redémarre le serveur, et il est désormais possible de se connecter en SSH à mon docker :

Une image contenant texte

Description générée automatiquement

Commentons donc toutes les règles de détection snort sauf celles dans **/etc/snort/rules/local.rules**.

Ajoutons ensuite la règle **alert tcp any any -> any 1022 (msg: "Tentative de connexion SSH, bikkuri shita :O"; sid:460)**

On lance maintenant snort en mode IDS :

**snort -dev -l log -c /etc/snort/snort.conf -i eth0**

Puis on essaie de se connecter en SSH au docker :



(on interrompt la connexion pour ne pas surcharger de paquets).

Aucune alerte levée.

Je consulte une FAQ officielle : <https://www.snort.org/faq/i-m-not-receiving-alerts-in-snort>

Il s’agit peut-être d’un problème de « TCP offload », lorsque la carte réseau calcule elle-même le checksum des paquets pour décharger le CPU de cette tâche.

Lançons la commande correcte : **snort -dev -l log -c /etc/snort/snort.conf -i eth0**

Cela semble fonctionner

Une image contenant texte

Description générée automatiquement

On reçoit une alerte sur environ la moitié des paquets, ce qui est logique étant donné que l’autre moitié correspond aux réponses du serveur.

Le fichier de log s’est rempli :

Une image contenant texte

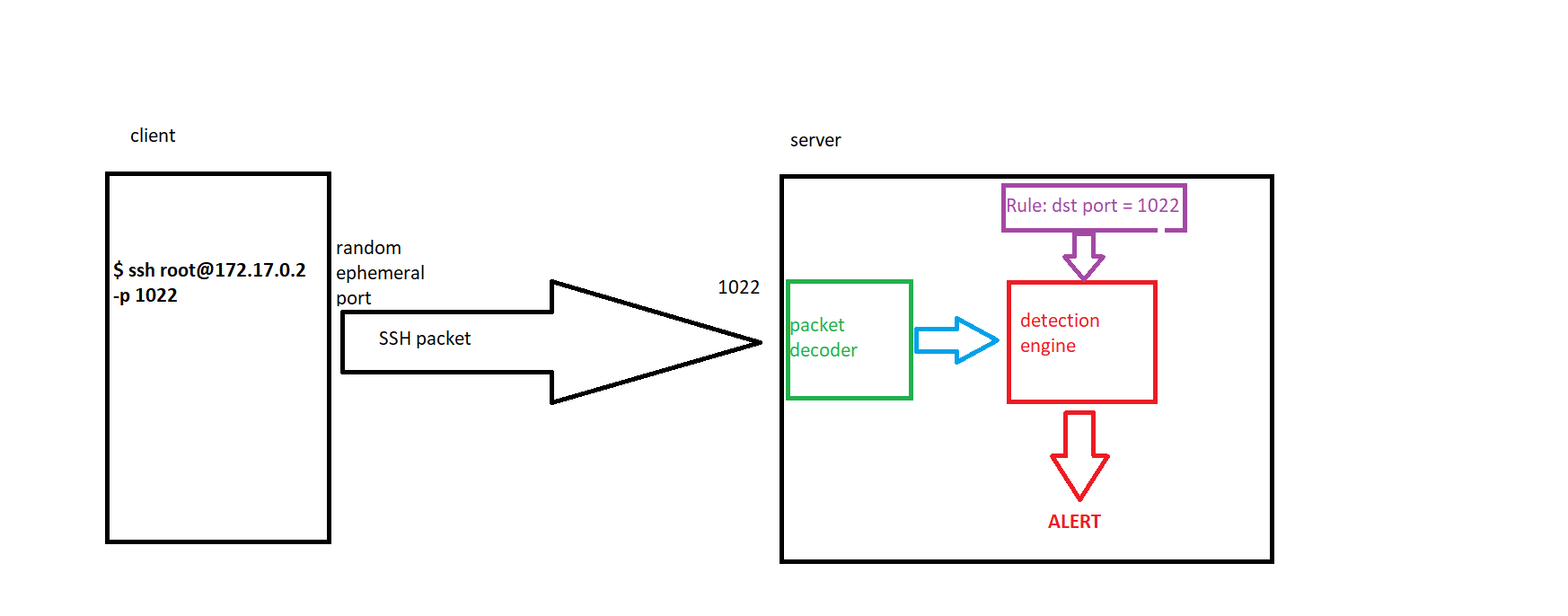
Description générée automatiquement

Mais impossible de le lire

Une image contenant texte

Description générée automatiquement

Je n’ai hélas pas trouvé d’explications.



1. Pour mener à bien le scan XMAS et sa détection, je me suis basé sur le tutoriel suivant :

Commençons par retirer la règle précédemment créée et ajoutons la règle suivante :

**alert tcp any any -> any 1022 (msg: "Tentative de scan XMAS, bikkuri shita :O"; flags: FPU; sid: 460; rev: 1)**

Cette règle va détecter tous les scans XMAS sur le port SSH 1022. Essayons d’abord de lancer une connexion SSH conventionnelle :

**ssh root@172.17.0.2 -p 1022**

Aucune alerte levée :



Essayons maintenant un scan XMAS sur ce même port :

**sudo nmap -sX -p1022 172.17.0.2**

Cette fois-ci, snort a bien détecté un scan XMAS sur le port 1022.



Essayons maintenant de lancer un scan avec Python scappy :

J’ai utilisé le script à l’adresse suivante : <https://github.com/PacktPublishing/Python-Penetration-Testing-Cookbook/blob/master/Chapter06/xmas-scanner.py>

Le port est bien détecté :



De plus, une alerte est bien levée par snort :



1. Le fichier acquisitions.md se récupère à l’adresse <http://localhost:3000/ftp/acquisitions.md>

On l’obtient avec un listing directory sur le dossier ftp.

Ajoutons maintenant la règle suivant à snort : **alert tcp any any -> any 3000 (msg: "Someone is trying to steal our acquisitions!"; content: "/ftp/acquisitions.md";sid: 460; rev:1)** après avoir au préalable retiré les autres.

Il faut aussi penser à aujouter l’option **-k none** à snort puisqu’on accède au site via un port-forwarding :

**snort -dev -l log -c /etc/snort/snort.conf -i eth0 -k none**

Essayons de récupérer un document quelconque sur notre site. Aucune alerte levée :



Essayons de récupérer notre fichier d’acquisitions :



En effet notre paquet contient bien la chaîne recherchée :

Une image contenant texte

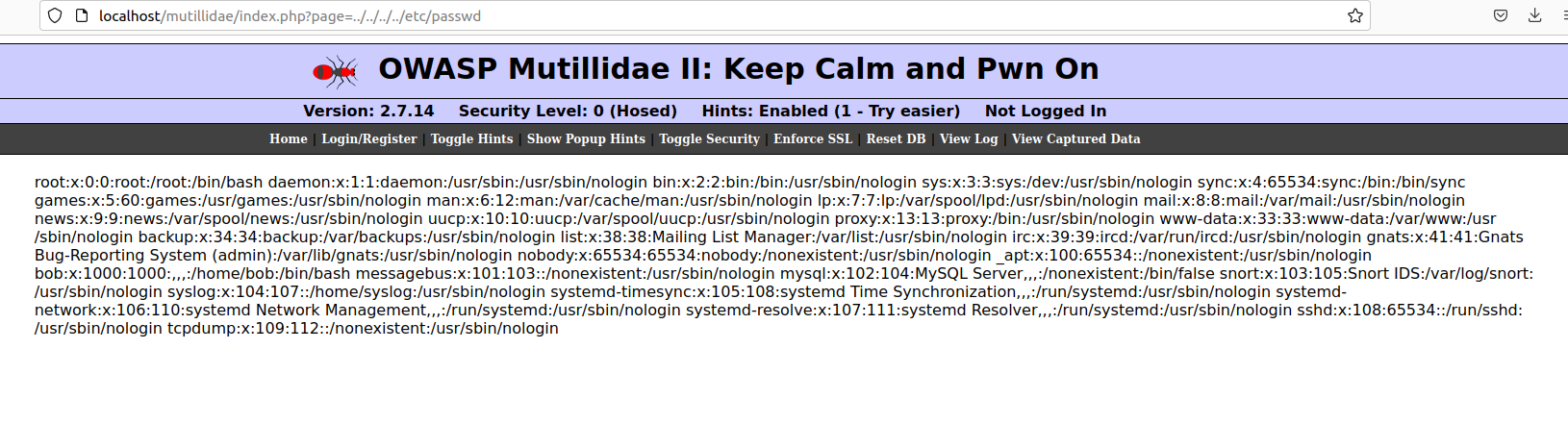
Description générée automatiquement

1. Recherchons maintenant des vulnérabilités sur le site <http://localhost/mutillidae/>

Testons au hasard une page du site : <http://localhost/mutillidae/index.php?page=text-file-viewer.php>

On remarque que le fichier PHP à afficher est lu comme un paramètre, nous pourrions donc essayer de lire un fichier système en modifiant un peu l’URL :

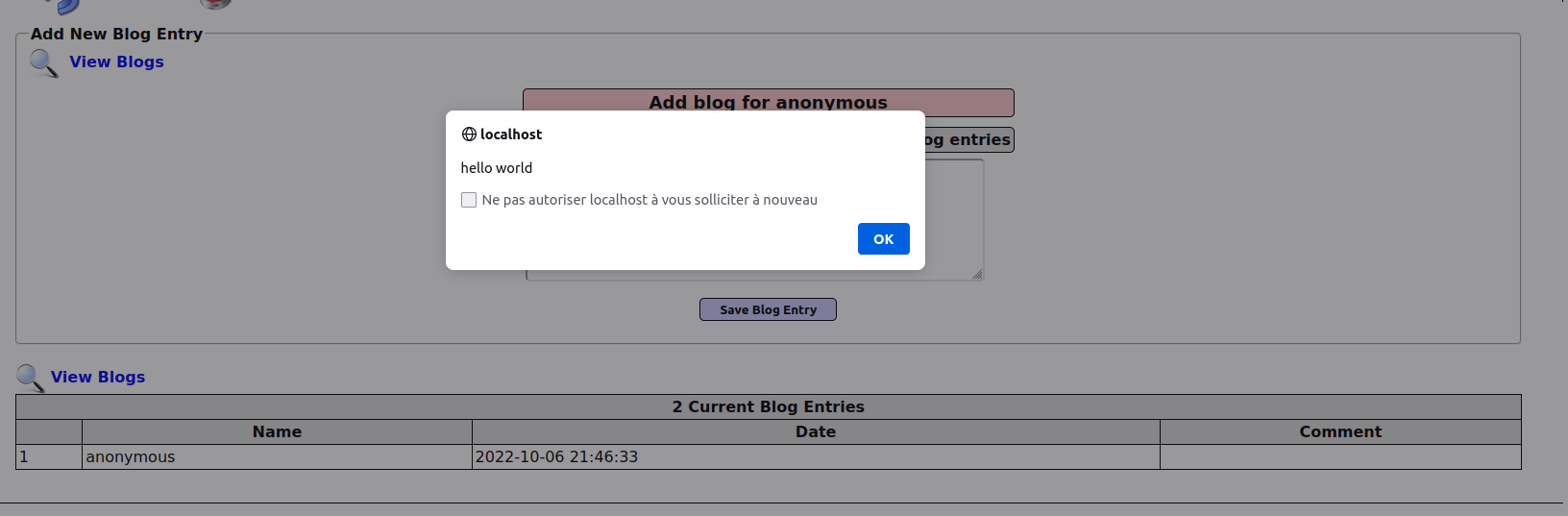
<http://localhost/mutillidae/index.php?page=../../../../etc/passwd>



Je n’ai pas eu à chercher bien loin pour trouver une deuxième faille de sécurité : la page <http://localhost/mutillidae/index.php?page=add-to-your-blog.php> nous propose en effet de remplir un formulaire. Nous pourrions essayer de vérifier s’il est bien protégé contre les injections XSS :



Il semblerait que cette page ne soit pas protégée correctement contre les failles XSS. Ces dernières permettent par exemple de voler les cookies de l’utilisateur ou bien de charger du contenu à son insu.



*Note : j’ai aussi essayé de procéder à une injection SQL avec l’outil sqlmap, sans succès malheureusement.*

Essayons maintenant de détecter l’exploitation de ces failles via snort.

Ajoutons la règle locale snort suivante : **alert tcp any any -> any 80 (msg: "Someone is trying to access forbidden path"; content: "/..";sid: 460; rev:1)**

Lançons snort: **alert tcp any any -> any 80 (msg: "Someone is trying to access forbidden path"; content: "/..";sid: 460; rev:1)**

Si nous essayons de consulter le site « normalement », aucune alerte n’est levée.



Par contre, si nous essayons de consulter <http://localhost/mutillidae/index.php?page=../../../../etc/passwd>

, notre attaque est bien détectée par snort :

Une image contenant texte, périphérique, jauge

Description générée automatiquement

Si nous cherchons à détecter les injections XSS, cela pourrait s’avérer beaucoup plus ardu. Contentons-nous donc simplement des injections de script. Il suffit pour cela de détecter l’envoi du pattern suivant dans une requête : **%3Cscript%3E(.\*)%3C%2Fscript%3E** (on note que les caractère spéciaux sont encodés au format URL).

On peut essayer avec les règles disponibles sur cette page : <https://github.com/OpenIxia/sample-cloud-ids/blob/master/sensor/snort/local.rules>

Il est cependant plus facile d’écrire la règle suivante, qui n’a pas généré de faux positif lors de mes tests : **alert tcp any any -> any 80 (msg:"XSS detected :'("; flow:established,to\_server; content: "%3Cscript"; classtype: Web-application-attack; sid:640; rev:1;)**

On se contente ici d’essayer de détecter le pattern **%3Cscript**, ce qui fonctionne également parfaitement.

Sur une navigation normale :



Sur l’injection testée plus haut :



1. Essayons de trouver l’adresse email de l’administrateur du site.

La page de login se trouve sur <http://localhost:3000/#/login>. Dans un premier temps, essayons d’obtenir cette adresse email avec une simple méthode d’OSINT (en cherchant par exemple sur la page de contacts), puis dans le code source.

Cela m’a permis de réussir un autre challenge par hasard, en modifiant le bouton d’envoi du feedback :

Une image contenant texte

Description générée automatiquement

Cela dit, je n’ai toujours pas trouvé l’email souhaité.

Après un certain temps de recherche, je me décide à tenter l’injection SQL. Je cherche d’abord une URL où un paramètre est injectable (<http://site.come/dir/page?par=inject_here>), sans succès. J’essaie donc sur le formulaire de connexion en entrant le login suivant : **test' OR id = 1;#** , puis un mot de passe aléatoire.

Si le formulaire est injectable, la requête effectivement exécutée devrait être quelque chose comme :

**SELECT \* FROM users WHERE login = ‘test’ OR id = 1;#’ AND password = pass;**

Ce qui en SQL signifie “sélectionner le premier utilisateur avec l’identifiant 1 », donc probablement l’administrateur.

Cela fonctionne :

Une image contenant texte

Description générée automatiquement

Je me suis bien emparé du compte administrateur, et je connais donc son adresse email :

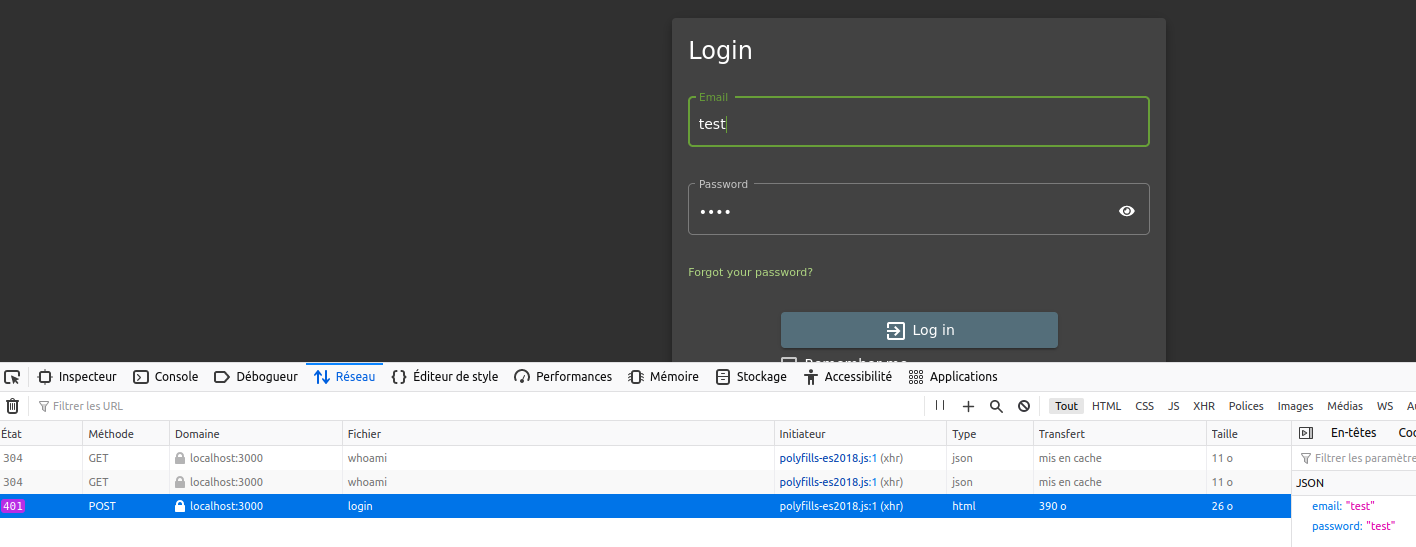
Une image contenant texte

Description générée automatiquement

J’ai techniquement pris le contrôle du compte administrateur, je pourrais donc m’arrêter ici. Ce n’est cependant pas ce qui est demandé dans le TD.

Je télécharge la liste des 1050 identifiants les plus communs sur <https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/best1050.txt> (nb : je suggère d’utiliser le dictionnaire **howsecureismypassword** ou **rockyou** pour une liste plus exhaustive).

Commençons par regarder le format de la requête d’authentification :



On observe que les paramètres d’authentification sont transmis sous la forme **email=email&password=password** à l’adresse [**http://localhost:3000/rest/user/login**](http://localhost:3000/rest/user/login). De plus, en cas d’echec, on reçoit un code de retour **HTTP 401**.

Ecrivons rapidement un petit script python pour tester tous les mots de passe de notre dictionnaire :

*import requests*

*url = 'http://localhost:3000/rest/user/login'*

*best\_1050\_file = open("best1050.txt")*

*for password in best\_1050\_file:*

*password = password.strip() # remove new line character*

*credentials = {'email': 'admin@juice-sh.op', 'password': password}*

*http\_response = requests.post(url, data = credentials)*

*if http\_response.status\_code != 401:*

*print("password found:", password)*

*break*

*best\_1050\_file.close()*





Essayons maintenant de détecter ce type d’attaque avec snort. La difficulté est de distinguer une attaque de bruteforce d’un utilisateur de bonne foi qui se trompe de mot de passe. Il faudrait donc déclencher l’alerte qu’en cas d’un certain nombre d’erreurs dans une courte fenêtre de temps.

Pour cela, snort nous propose le paramètre de **threshold** : <https://www.snort.org/faq/readme-thresholding>

Nous obtenons donc la règle suivante :

**alert tcp any 3000 -> any any (msg:"Bruteforce attack detected ! :'("; content: "HTTP/1.1 401 Unauthorized"; threshold: type threshold, track by\_src, count 10, seconds 60; sid:640; rev:1;)**

Une alerte sera déclenchée si le serveur renvoie un code 401 plus de 10 fois au même client dans une fenêtre de 60 secondes.

Un utilisateur de bonne foi se trompe de mot de passe 3 fois, aucune alerte :



Je lance le script de bruteforce, alerte :

Une image contenant texte

Description générée automatiquement

1. Après avoir installé hping3 à l’aide de la commande apt, je lance la commande suivante :

**sudo hping3 -S --flood -V -p 3000 172.17.0.2**

On remarque que les droits root sont nécessaires sous Linux pour envoyer beaucoup de paquets réseau en même temps. Au passage j’inscris directement l’adresse IP de l’image docker et non pas l’hôte local, même si cela fonctionnerait puisque nous avons précédemment configuré docker pour faire du port forwarding. Cela afin d’éviter une perte de performances et du calcul CPU inutiles.

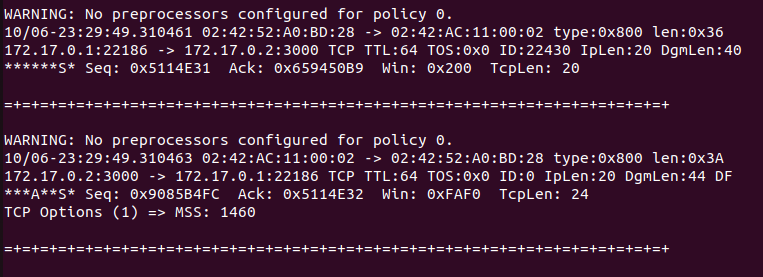
Je lance donc la commande et **l’interrompt après environ 10 secondes** :

Une image contenant texte

Description générée automatiquement

Nous envoyons donc en moyenne environ **150 000 paquets / seconde**.

Sans surprise, snort relève la détection de nombreux paquets sur le port 3000 :



Notons au passage que le site reste toujours accessible via le navigateur, l’attaque DOS a donc une portée assez limitée.

Maintenant, notre objectif ne va plus seulement consister en la détection d’une attaque, mais aussi dans le blocage de celle-ci. Il faut donc configurer snort pour bloquer des paquets : <http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node29.html>.

Commençons par écrire une règle en nous inspirant du **threshold** de la question précédente pour détecter une attaque DOS et lever une alerte :

**alert tcp any 3000 -> any any (msg:"DOS attack detected ! :'("; threshold: type threshold, track by\_src, count 1000, seconds 10; sid:640; rev:1;)**

Cette fois-ci, on cherche à détecter 1000 requêtes sans pattern particulier dans une fenêtre de 10 secondes, provenant de la même adresse IP. Le nombre de requêtes autorisées est en effet un peu plus élevé pour ne pas déclencher de faux positif si l’utilisateur télécharge un gros fichier par exemple. Testons notre règle :

Navigation « honnête » :



Attaque DOS :



Essayons de remplacer la règle d’**alert** par un **drop** :

**drop tcp any 3000 -> any any (msg:"DOS attack detected ! :'("; threshold: type threshold, track by\_src, count 1000, seconds 10; sid:640; rev:1;)**

snort indique qu’il a réussi à éliminer environ **50% des paquets de l’attaque DOS**

Une image contenant texte

Description générée automatiquement

Sur une période de temps de 10s, hping n’a transmis que 860 000 paquets, soit 86 000 paquets / seconde.

Une image contenant texte

Description générée automatiquement

Nous avons réussi à réduire l’attaque de moitié, mais nous pourrions essayer de faire mieux.

J’ai essayé d’améliorer mon taux de mitigation en suivant les règles proposées sur ou <https://www.researchgate.net/publication/335803817_Mitigation_of_DoS_and_Port_Scan_Attacks_Using_Snort> ou bien <https://www.researchgate.net/publication/338660054_DETECTING_DDoS_ATTACK_USING_Snort>, mais sans grand succès hélas.

Afin d’expérimenter l’attaque **DDOS** (Distributed DOS, attaque DOS depuis plusieurs machines), j’ai lancé hping3 depuis plusieurs terminaux, puisque je n’ai pas pour le moment un botnet avec plusieurs milliers de machines à ma disposition. Le résultat sur la console snort est très similaire.

Une façon envisageable d’échapper à la détection DOS avec une attaque DDOS serait donc de limiter l’envoi de requête par machine individuelle. Nous aurions donc beaucoup de machines envoyant chacune des requêtes à une fréquence acceptable pour l’analyseur de paquets. L’ensemble des requêtes envoyées serait par contre suffisant pour perturber le fonctionnement de notre serveur.

Le gros inconvénient de cette méthode réside cependant dans le fait de disposer d’un nombre conséquent de machines à disposition.