**Techniques d'attaque - DDOS**

Tristan BILOT, Nora DELFAU, Enzar SALEMI, Madushan THAMBITHURAI

EPITA

10 juin 2021

1. **Notes**

**Paquet RST** : Un paquet TCP Reset (RST) est utilisé par un expéditeur TCP pour indiquer qu'il n'acceptera ni ne recevra plus de données. Les périphériques de gestion de réseau hors chemin peuvent générer et injecter des paquets de réinitialisation TCP afin de mettre fin aux connexions indésirables.

**Règle iptable :**

Pour attaquer le serveur cible (192.168.56.102), on peut insérer les règles iptables suivantes dans les VM attaquantes respectives :

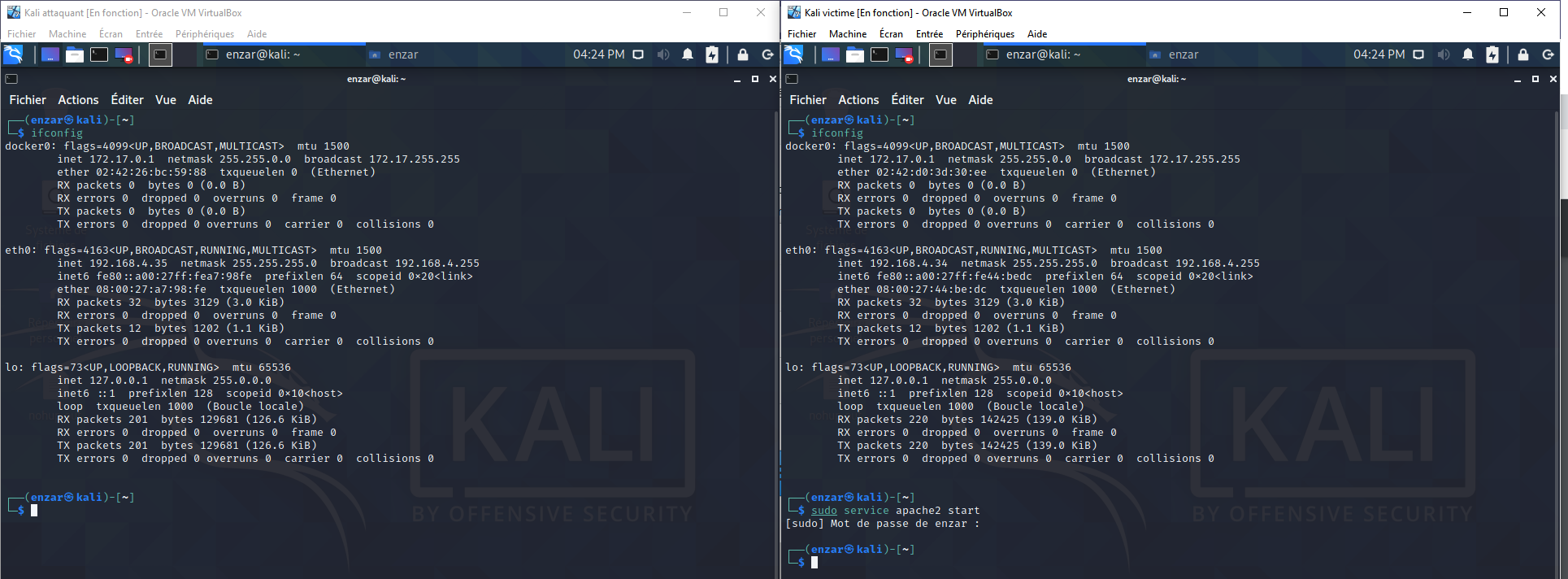
*iptables –A OUTPUT –p tcp –s 192.168.56.101 --tcp-flags RST RST –j DROP*

*iptables –A OUTPUT –p tcp –s 192.168.56.103 --tcp-flags RST RST –j DROP*

**Remarque** : Cette règle SUPPRIMERA les paquets de la chaîne de SORTIE qui ont le drapeau RST défini. Les règles iptables ne s'appliqueront pas aux paquets générés par Scapy, qui crée le paquet entier dans son espace. Cependant, les paquets malformés/manipulés fabriqués par Scapy seront vus par le noyau, qui enverra des réponses RST (réinitialisations) à la cible, puisque ce dernier (le noyau de l'attaquant) n'a pas initié cette communication TCP. Pour éviter cela, nous devons utiliser les règles iptables ci-dessus, afin que les RST du noyau n'atteignent pas la cible - sinon, le tampon SYN de la cible ne sera pas plein et l'attaque DDoS échouera.

1. **Déni de service par syn flooding sur un serveur web**

Nous avons une machine attaquant (à gauche) et une machine victime (à droite). Ces deux machines serviront durant tout le déroulement du TP.



1. **Créer le paquet p et envoyer le au serveur avec la commande sendp**

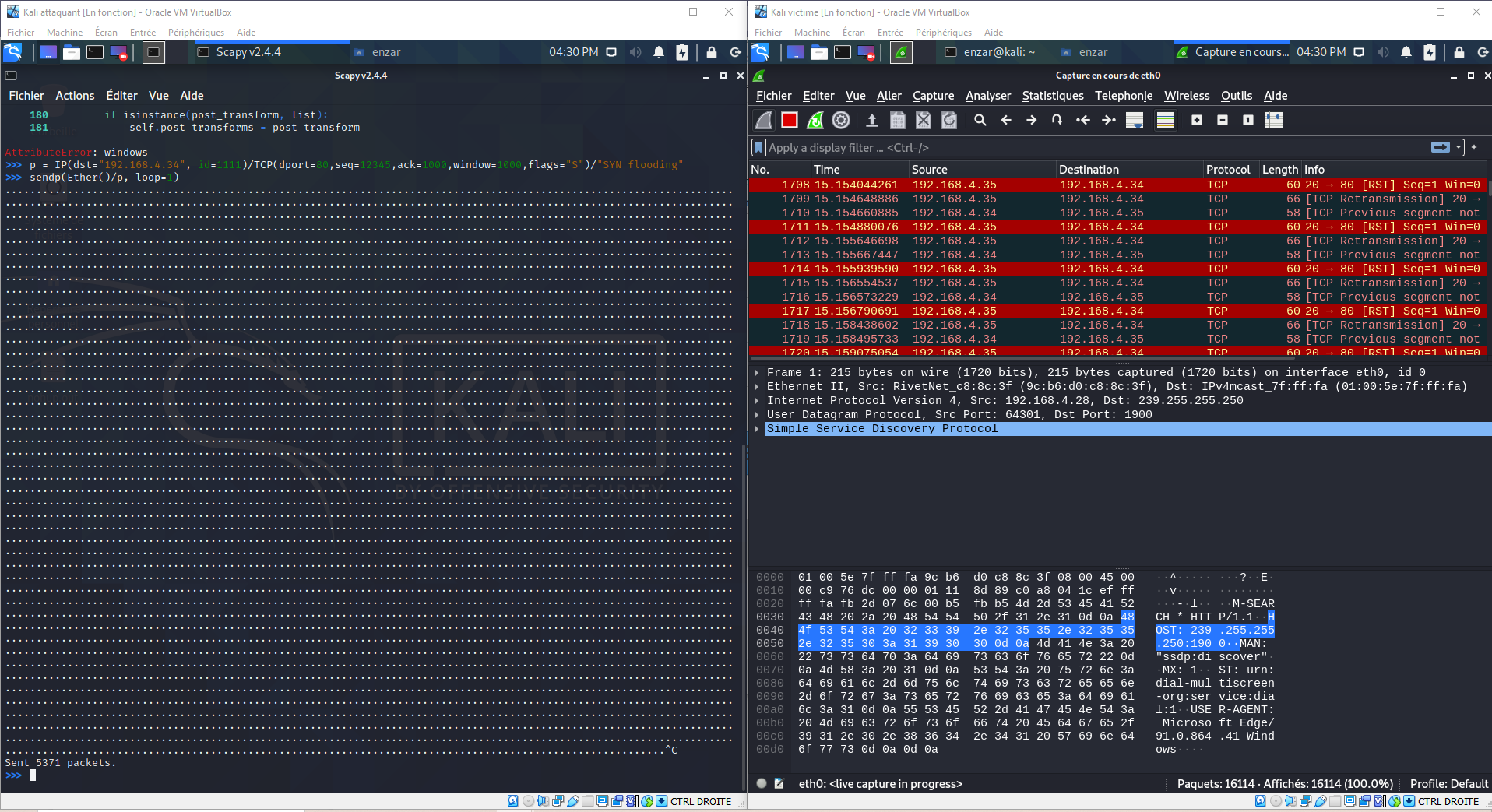
p=IP(dst=”137.194.35.X”,id=1111)/TCP(,dport=80,seq=12345,ack=1000,window=1000,flags="S")/"SYN flooding"

sendp(p, loop=1)

1. **Lancer l’attaque avec la commande *sendp(Ether()/p, loop=1)***

Sur la capture Wireshark de la machine cible, on retrouve les requêtes TCP de l’attaquant, les réponses de la machine cible et des paquets labellisées RST retournées par la machine de l’attaquant automatiquement.

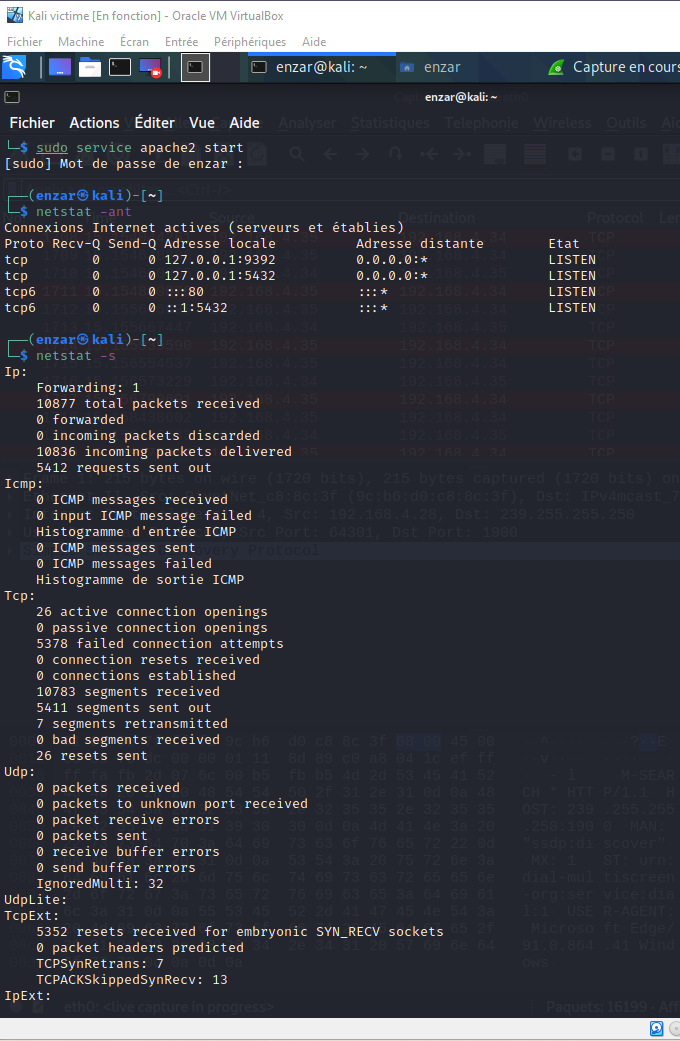
Un paquet TCP RST est utilisé dans le cas où l’envoyeur (ici l’attaquant) ne souhaite plus recevoir ni envoyer de données. Le paquet RST met donc fin à la demande de connexion.



1. **Vérifier le résultat de l’attaque sur la machine de votre collègue avec la commande (netstat –ant, netstat -s)**

D’après la commande “*netstat -ant*”, le port 80 (comme tous les autres ports) de la machine cible sont toujours en état d’écoute.

En affichant les statistiques avec la commande “*netstat -s*”, on remarque que parmi les 5412 requêtes reçues, seules 26 connexions ouvertes sont encore actives et toutes les autres sont en échec à cause des requêtes RST.



1. **Est-ce que l’attaque a réussi à ouvrir des centaines de connexions sur le serveur victime ? sinon, pourquoi ?**

D’après nos observations, l’attaque n’a pas réussi. Le serveur est toujours disponible et l’ensemble des requêtes ont été soit approuvées soit refusées. Les requêtes n’ont donc pas pu s’accumuler dans la mémoire du serveur pour le rendre instable.

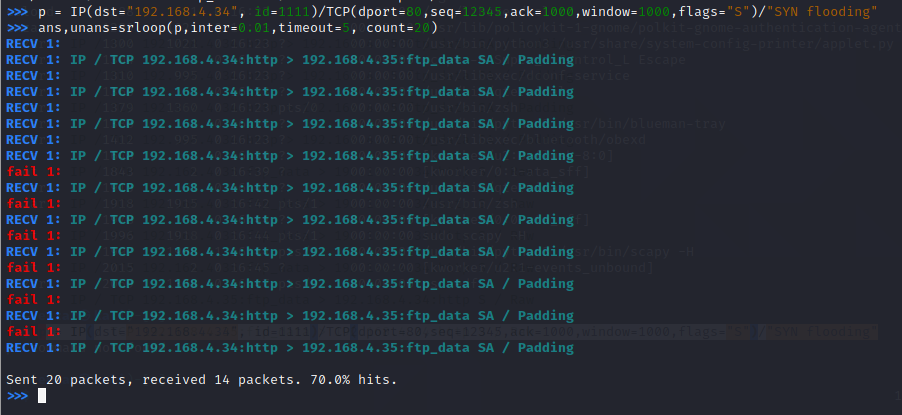
Le problème vient des paquets RST qui ont interrompu toutes les demandes de connexions et qui ont permis à la cible de vider sa mémoire des requêtes TCP.

1. **Les paquets générés par Scapy passeront par le noyau, qui va envoyer des réponses RST (les réinitialisations) à la cible, car le noyau de la machine attaquante n'avait pas initié les sessions TCP. Pour éviter cette situation, ajouter une règle iptables pour empêcher votre machine d’envoyer les RSTs. Sinon, l'attaque va échouer**
   1. *iptables –A OUTPUT –p tcp –s 137.194.X.Y --tcp-flags RST RST –j DROP*

137.194.X.Y est l’adresse IP de votre machine

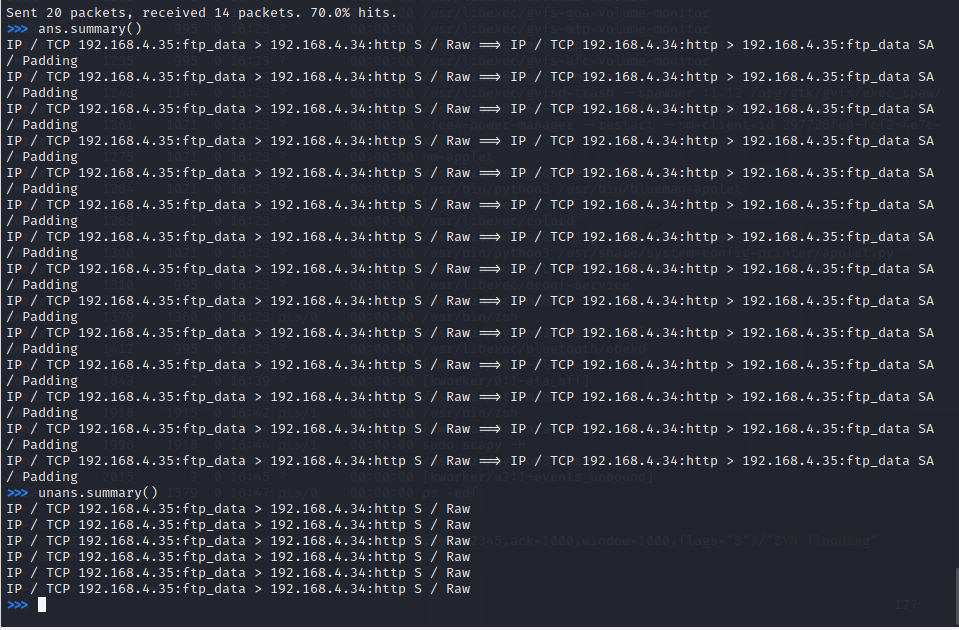
1. **Lancer l’attaque avec la commande srloop et vérifier la réponse *ans,unans=srloop(p,inter=0.01,timeout=5)*.**

Lors de l’attaque, la machine cible répond avec de SA (SYN-ACK). Mis à part les quatre premières requêtes, toutes les autres connexions TCP échouent.



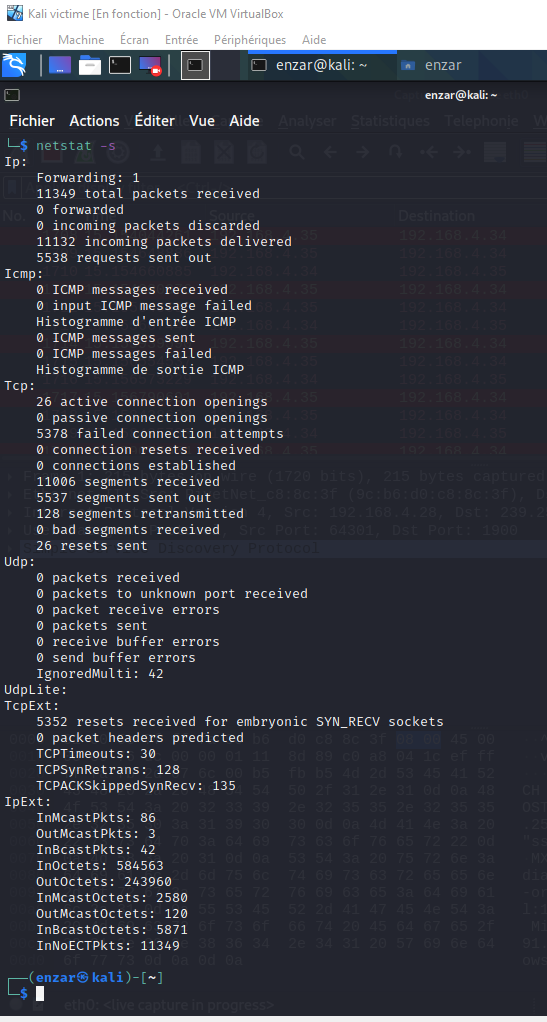
1. **Afficher le résultat (ans*.summary()* pour answered et unans*.summary()* pour unanswered)**

En affichant le résumé des paquets ayant reçu une réponse ou non, nous observons que toutes les requêtes TCP de l’attaquant ont obtenu une réponse de la cible. A l’inverse, les paquets envoyés par la machine de la victime n’ont fait l’objet d’aucune réponse. Les paquets RST précédemment observés ont donc bien été supprimés.



1. **Vérifier le résultat de l’attaque sur la machine de votre collègue avec la commande (*netstat –ant et v*oir les statistiques avec la commande netstat -s)**

Encore une fois, au vu des informations renvoyées par “netscan -s”, seules 26 connexions sont actives.



1. **Est-ce que l’attaque a réussi à ouvrir des milliers de connexions sur le serveur victime ? sinon, pourquoi ?**

L’attaque n’a **toujours pas réussi** malgré le blocage des paquets RST. Comme remarqué dans la capture Wireshark, la machine cible renvoie un paquet indiquant que la connexion faite par la paire IP/Port de la machine de l’attaquant a déjà lieu. Toutes les connexions **émanant d’un même port** de la machine qu’un connexion déjà initiée seront rejetées.

1. **Vérifier le temps d’attente du système après la réception d’un SYN avec la commande *cat /proc/sys/net/ipv4/tcpack\_retries* , si vous voulez le modifier, ouvrez le fichier /etc/sysctl.conf et y écrire net.ipv4.tcp\_synack\_retries = X, où X est la valeur souhaitée (1, par exemple). Ensuite appliquez le changement avec la commande *sysctl –p /etc/sysctl.conf*, vérifier la nouvelle valeur.**

Les fichiers /proc/sys/net/ipv4/tcpack\_retries et /etc/sysctl.conf n’existent pas sur le système Kali Linux. Nous n’avons pas trouvé d’équivalent.

1. **Ajouter la règle Firewall suivante sur la machine de l’attaquant et relancer l’attaque *iptables –A OUTPUT –p tcp –s 137.194.X.Y --tcp-flags RST RST –j DROP* avec 137.194.X.Y est l’adresse IP de la machine de l’attaquant si vous tapez la commande netstat –ant sur la machine de la victime, vous remarquez que des centaines de connexions semi ouvertes avec le label SYN\_RECEIVED sont affichées. Donc, l’attaque a fonctionné. Justifiez le rôle de la règle ci-dessus.**

Aucune modification, la question posée est la même que la précédente

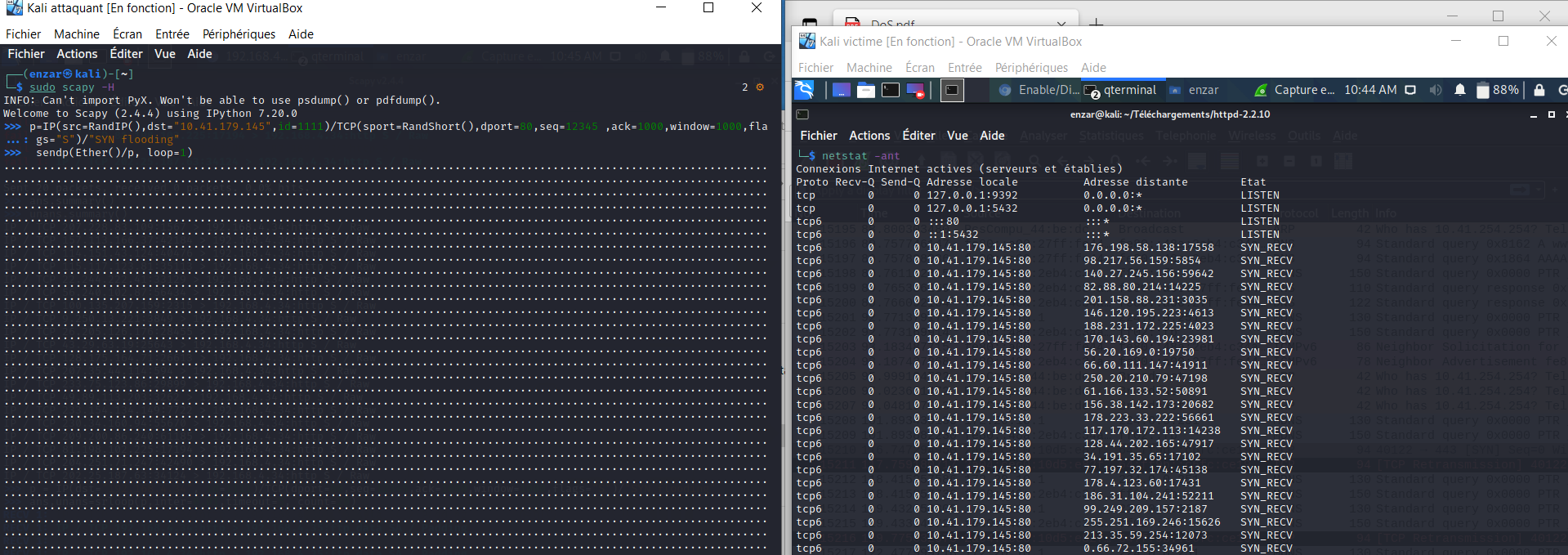
1. **Effacer la règle du Firewall avec la commande *iptables –D OUTPUT –p tcp –s 137.194.X.Y --tcp-flags RST RST –j DROP* Pour afficher la liste des règles : iptables -L**



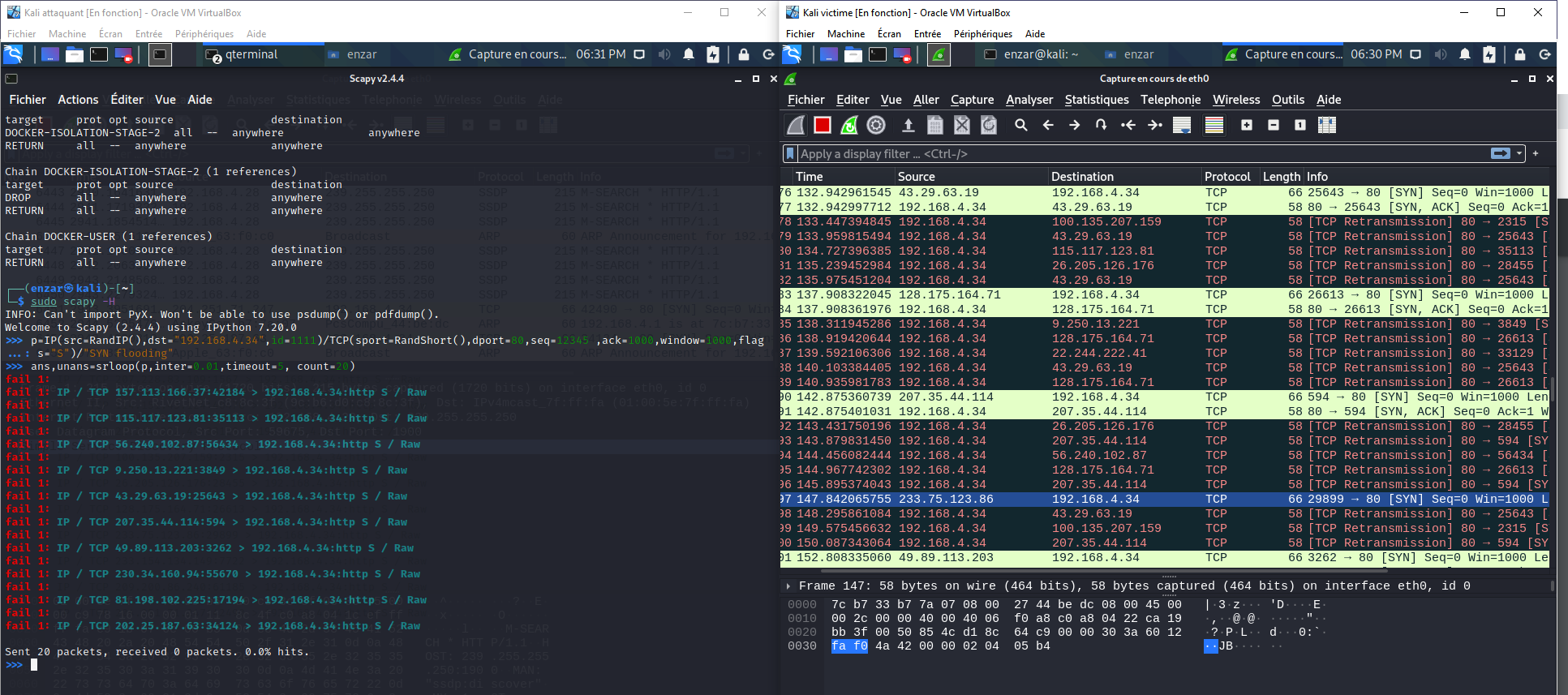
1. **Lancez maintenant l’attaque en utilisant des adresses IP sources aléatoires et des ports sources aléatoires. Vous constatez que l’attaque va fonctionner sans l’ajout d’une règle de firewall.** *p=IP(src=RandIP(),dst=”137.194.35.X”,id=1111)/TCP(sport=RandShort(),dport=80,seq=12345 ,ack=1000, window=1000,flags="S")/"SYN flooding"*

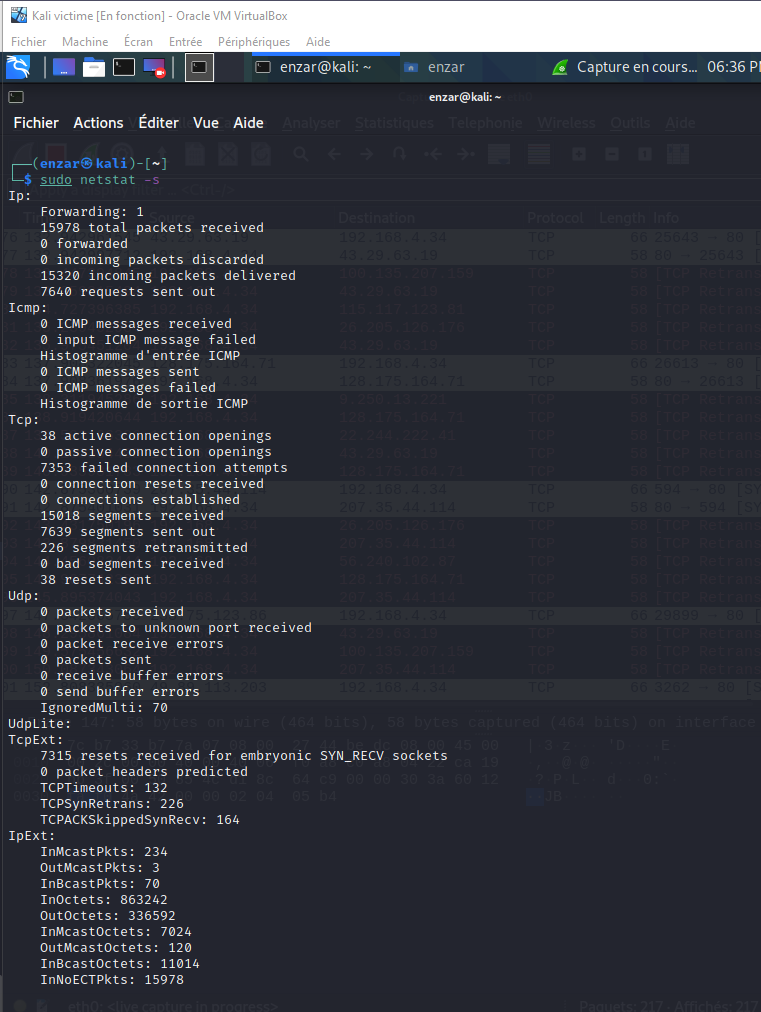
**Vérifierez le résultat avec la commande netstat –ant sur la machine victime.**

Avec la randomisation des ports et des IPs, l’attaque a pu marcher. A l’aide de la commande “*netstat -ant*”, nous avons remarqué un nombre important de connexions actives labellisées SYN-RECV indiquant qu’une requête de connexion a été reçue. Ceci vient du fait que l’attaquant ne cible plus le même port et donc plusieurs connexions simultanées peuvent désormais se faire.



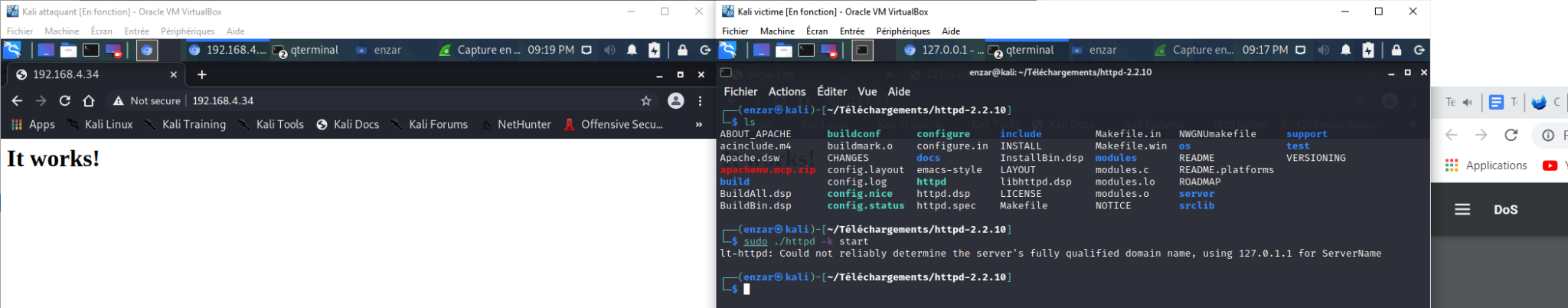
Voici une autre screen sur Wireshark.





1. **Déni de service au niveau applicatif par l’outil SlowLoris sur un serveur web Apache**

**Lancer le serveur apache: *sudo ./httpd –k start* et vérifiez que le serveur fonctionne en se connectant via un browser au serveur.**



**L’étape suivante consiste à installer le script en perl de l’attaque SlowLoris. Pour cela téléchargez SlowLoris avec l’outil git en tapant la commande:**

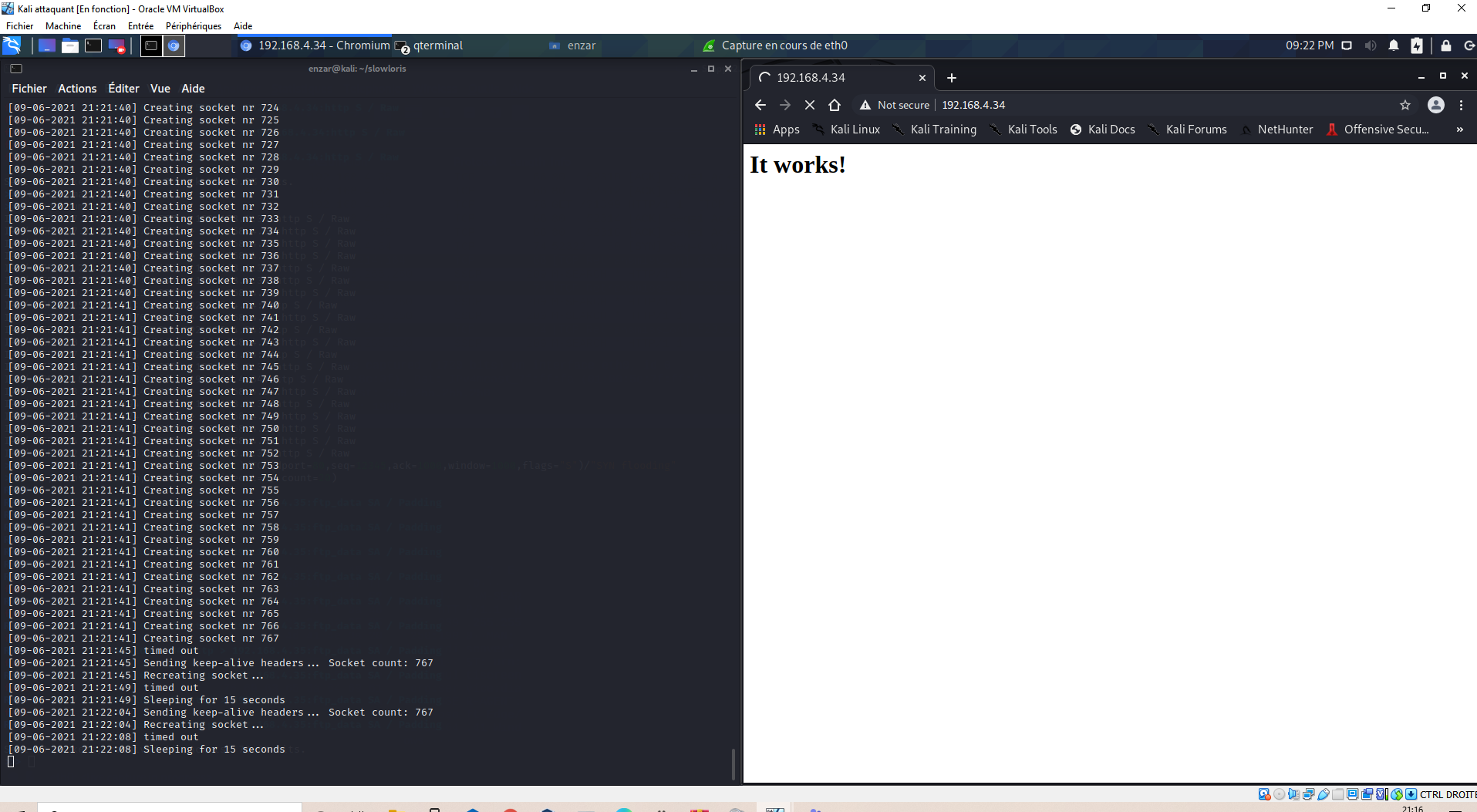
***git clone https://github.com/gkbrk/slowloris.git***

***cd slowloris***

***sudo ./slowloris.py –v –s 1500 137.194.183.150***

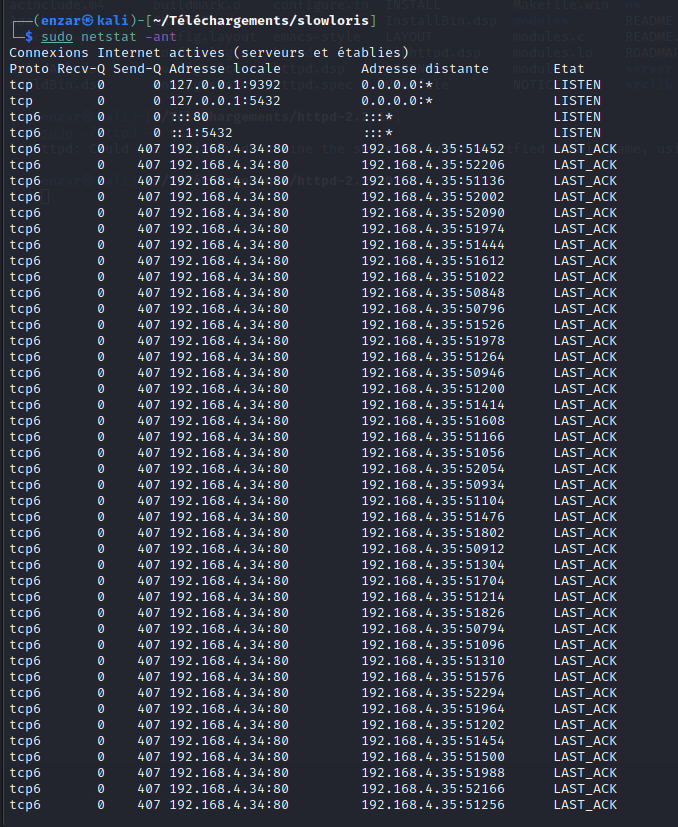
**Reconnectez-vous au serveur apache. Conclure !**

Pendant l'exécution du script, le serveur apache n’est plus capable de s’actualiser.Il charge en permanence sans répondre.



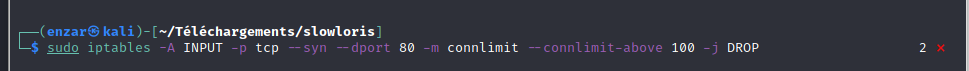
**Tapez la commande : *netstat –ant* sur la machine du serveur pour afficher le nombre de connexions.**

La commande “netstat -ant” remonte un nombre très important de connexions sur le serveur et semble sans limite (au-delà de 767 connexions).



**Pour contrer SlowLoris**

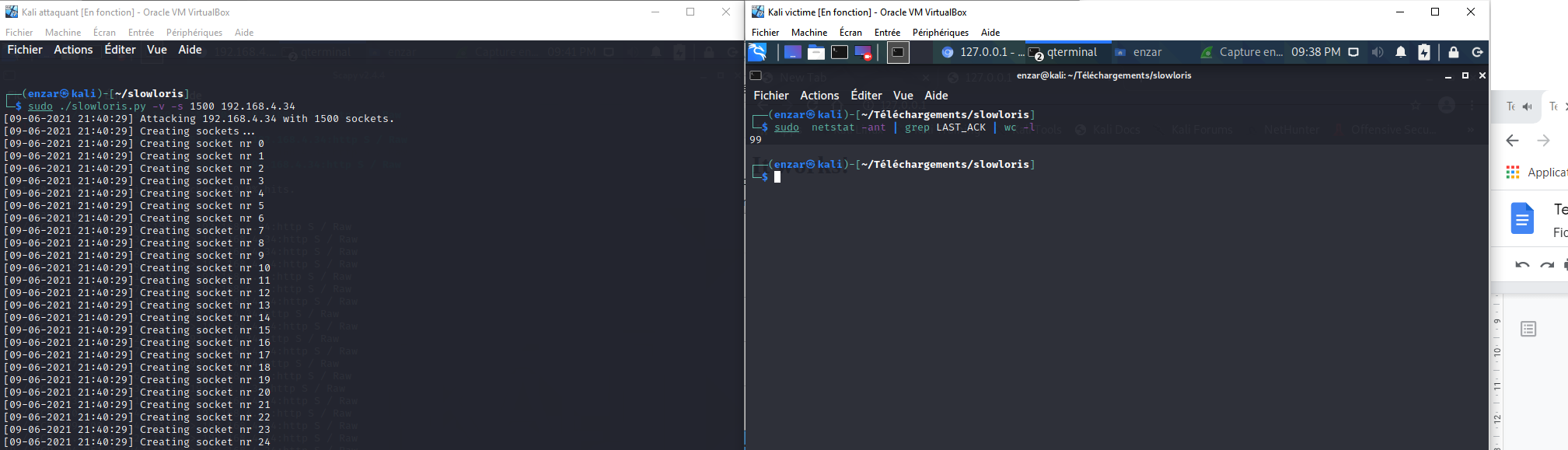
* **Première ligne de défense. Ajoutez une règle au firewall linux :**

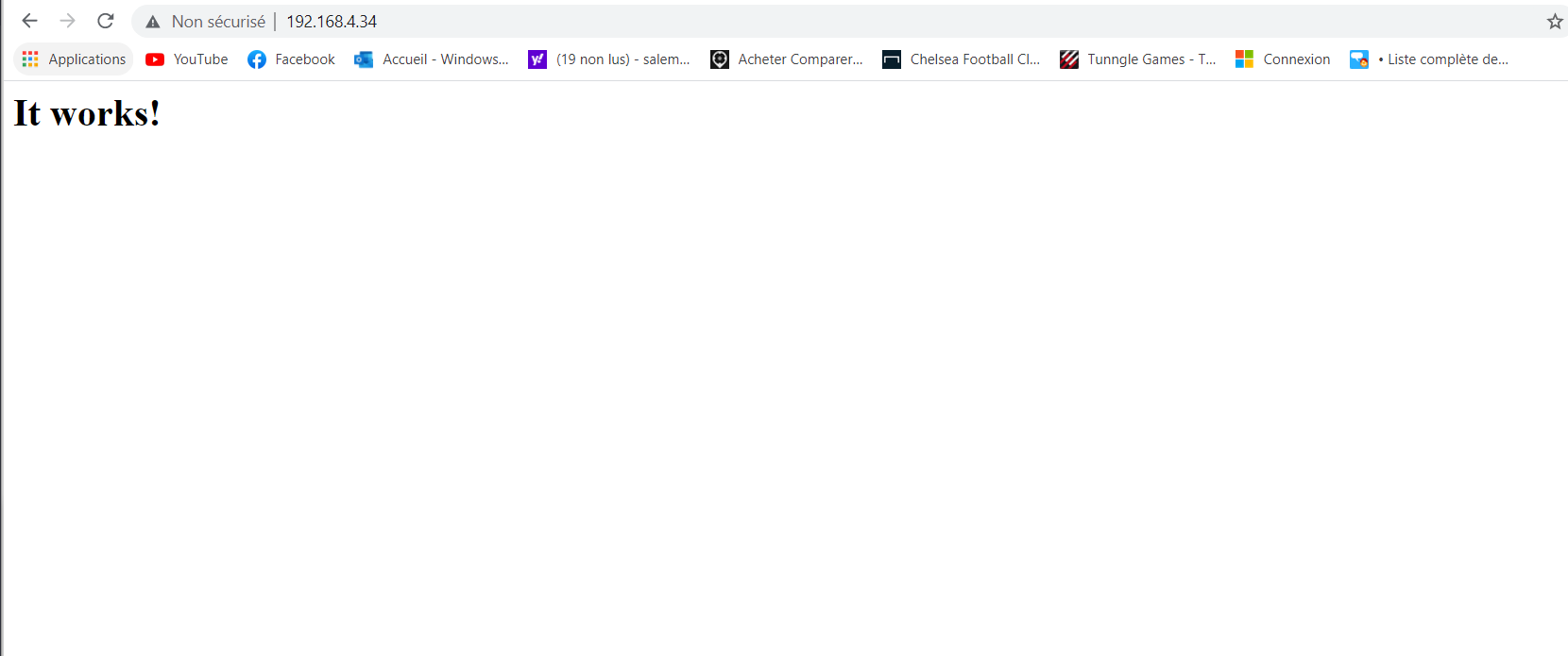
***iptables -A INPUT -p tcp --syn --dport 80 -m connlimit --connlimit-above 100 -j DROP***

**Relancez l’attaque avec sudo ./slowloris.py –v –s 1500 137.194.183.150**

**Remarquez le nombre de connexions simultanées.**

Après avoir ajouté la règle au firewall, le nombre de connexions simultanées au serveur s’est limité à 100 connexions. Le site est donc resté accessible.

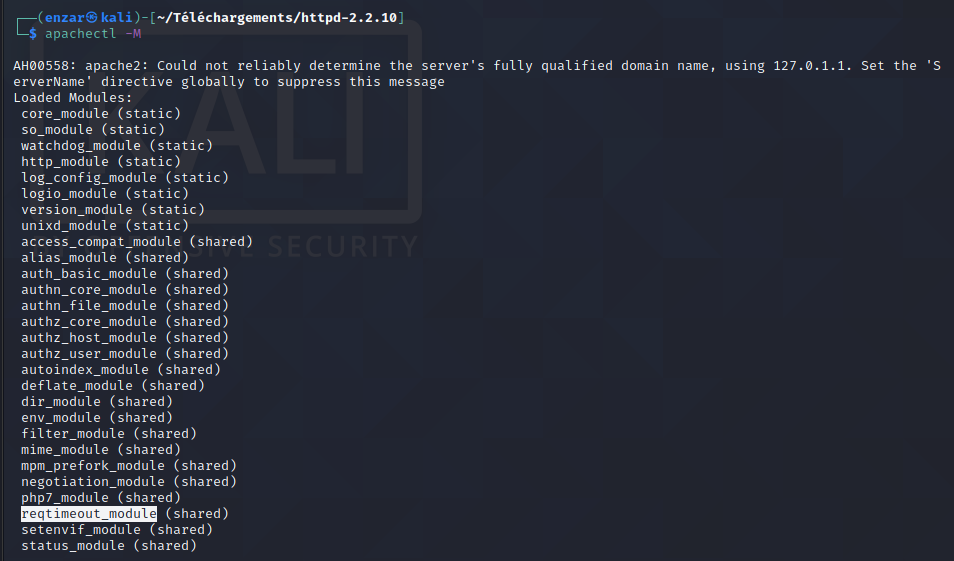


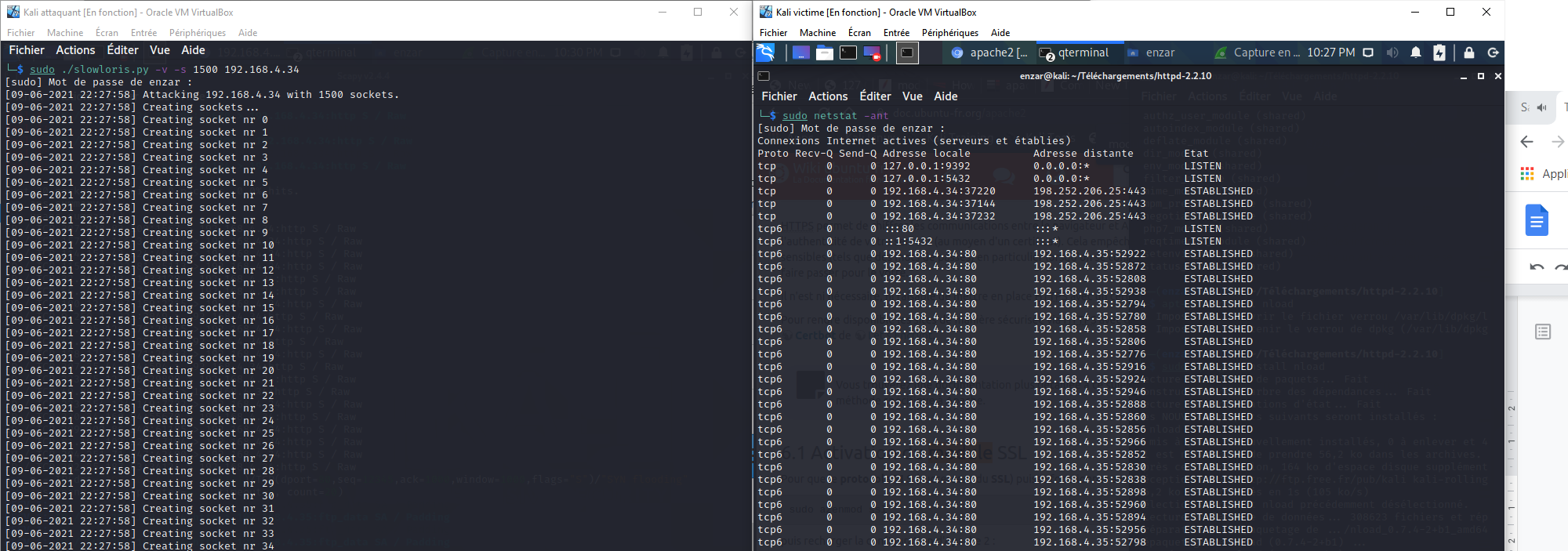


* **Deuxième ligne de défense**

**Installez l’un des modules mod\_reqtimeout ou mod\_qos.c dans apache.**

Ce module permet de définir aisément le délai maximum et le taux de transfert des données minimum pour la réception des requêtes. Si ce délai est dépassé ou ce taux trop faible, la connexion concernée sera fermée par le serveur.





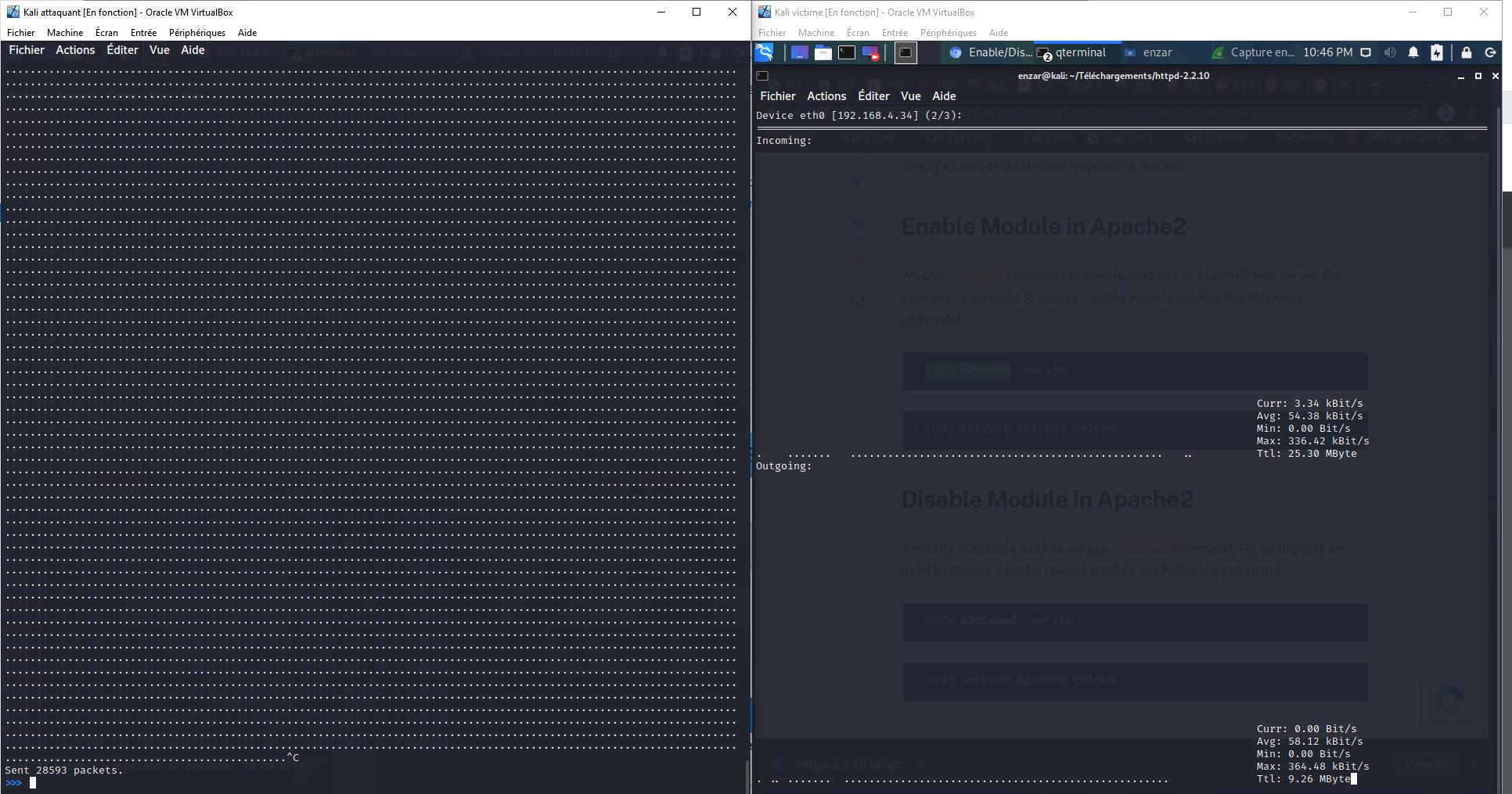
1. **Deni de service par UDP\_Flooding, ICMP\_flooding, Ping\_Death par scapy**

**1. Lancer un UDP flooding contre le serveur web en utilisant la fonction fuzz().**

**send(IP(src=RandIP(),dst="10.50.0.1")/fuzz(UDP()),loop=1)**

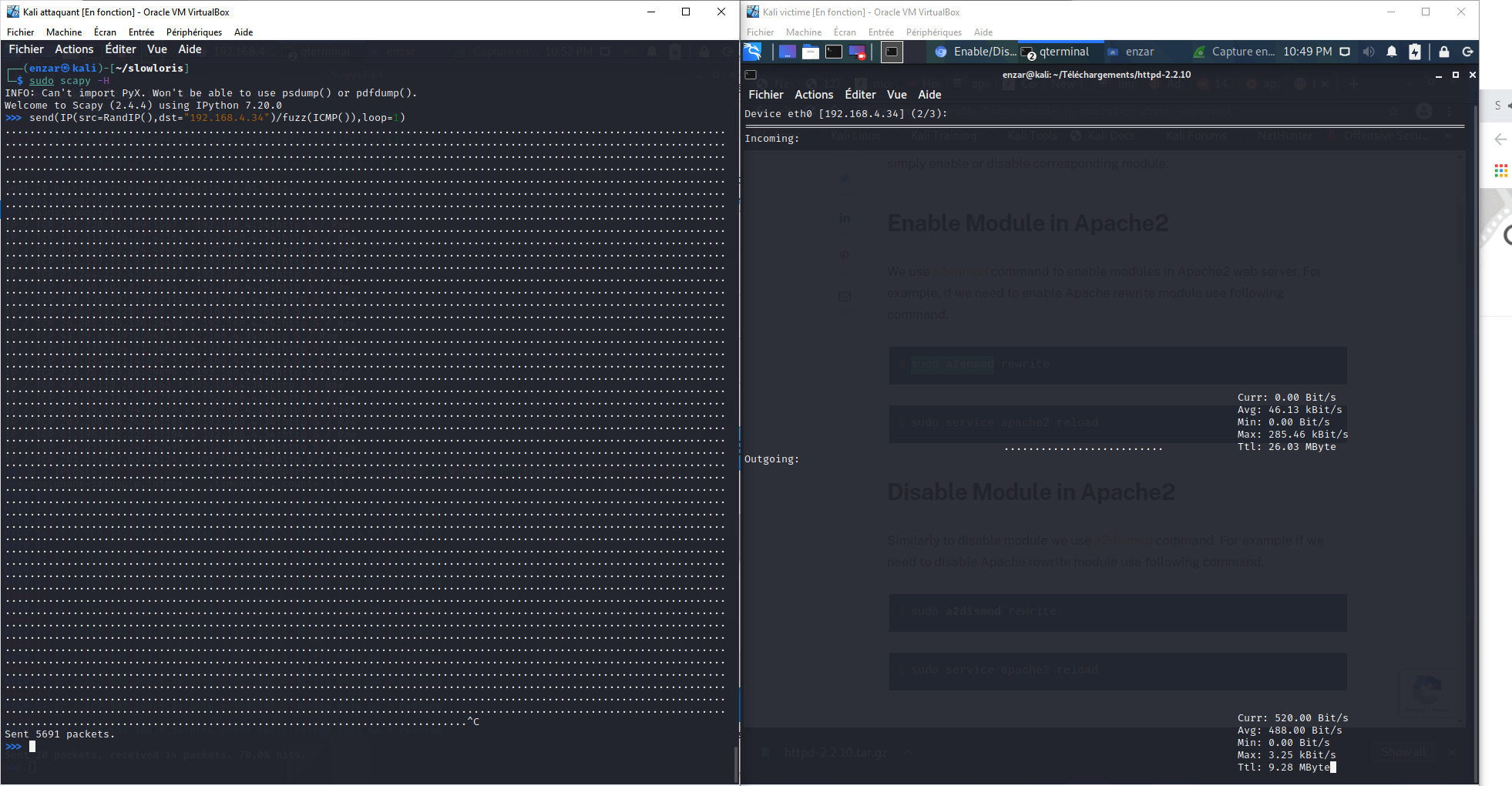
La fonction fuzz() est capable de changer n’importe quelle valeur par défaut par un objet dont la valeur est aléatoire et le type adapté au champ. Ce qui va nous permettre de créer rapidement un fuzzer et de l’envoyer dans une boucle.

Ici**, nload** nous permet de voir le trafic entrant qui comme nous le montre ce screen augmente significativement au moment de l’attaque.



**2. Lancer un ICMP flooding.**

Même constat que dans la question précédente.



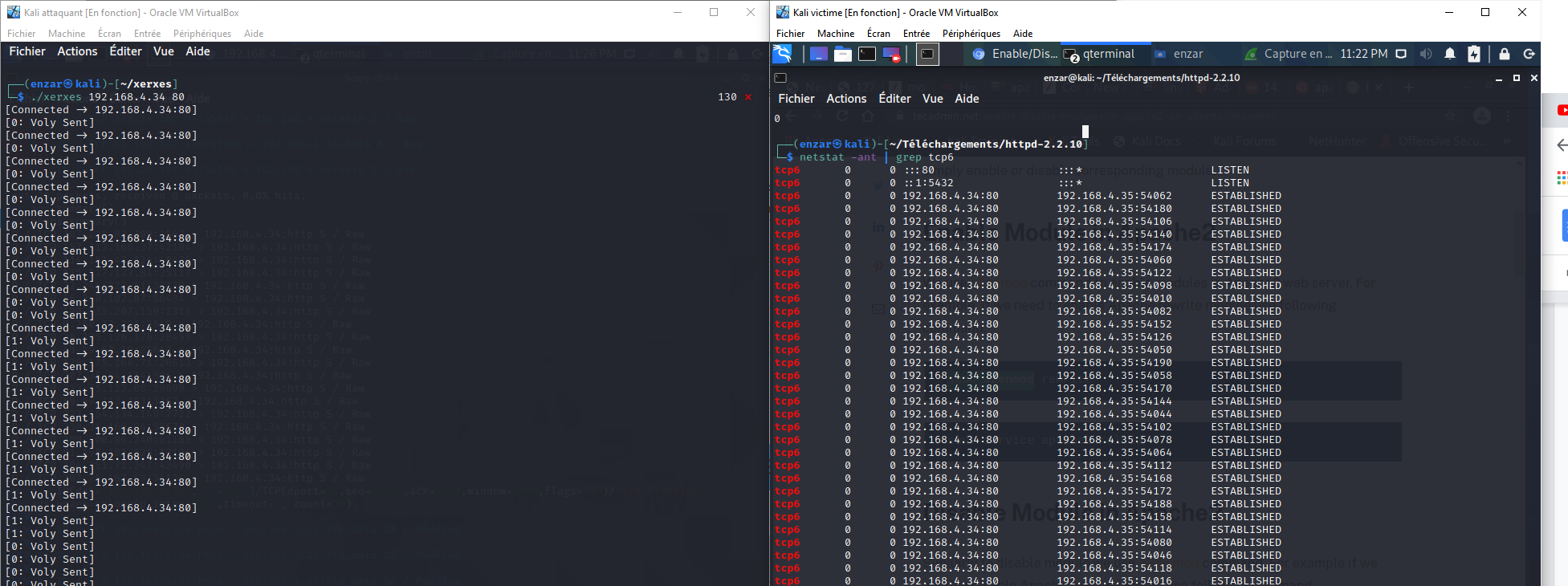
1. **Déni de service par connexions de sockets TCP**

**Allez au dossier xerxes et compiler le fichier gcc xerxes.c -o xerxes**

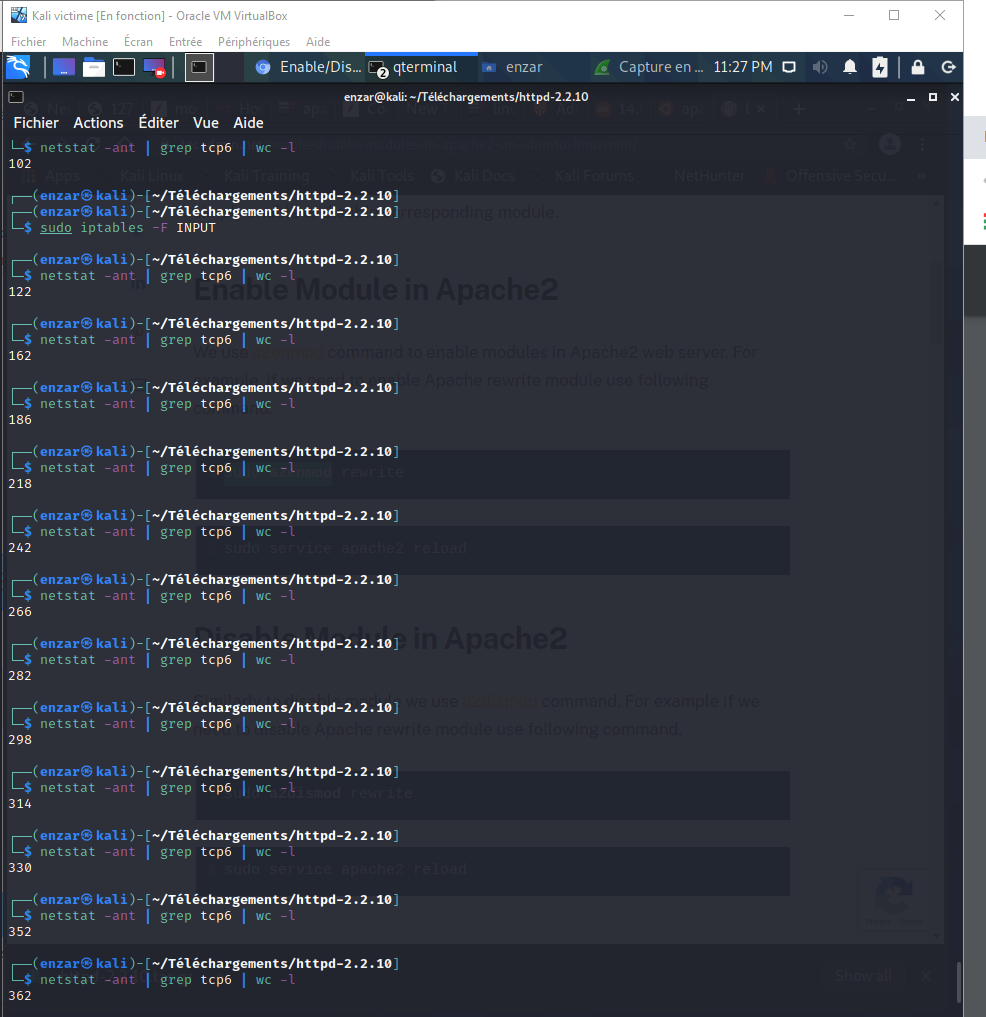
**Lancez le script contre le serveur web Apache ./xerxes 19.168.1.254 80**

1. **Tapez sur la machine victime la commande netstat –ant. Que remarquez-vous ?**

**Plusieurs connexion on été établi** avec le serveur cependant la limite fixée précédemment via la commande : “*iptables -A INPUT -p tcp --syn --dport 80 -m connlimit -- connlimit-above 100 -j DROP”***limite les connexions** établies à **100**.



Sans la limitation de la connexion, nous atteignons les 360 connexions établies.



1. **Ouvrez le fichier xerxes.c et analysez le code. Décrivez le principe de ce script.**



Ce script est un programme en C permettant de générer une attaque DOS sur une machine distante.

Le fonctionnement de ce programme consiste à ouvrir 8 connexions par processus (et non de thread comme il est indiqué dans le script). Par défaut, le script utilise 48 processus. Nous pouvons donc avoir jusqu'à 384 connexions simultanées.

Une fois le socket connecté un message vide est envoyé *“r=write(sockets[x], "\0", 1);”.*

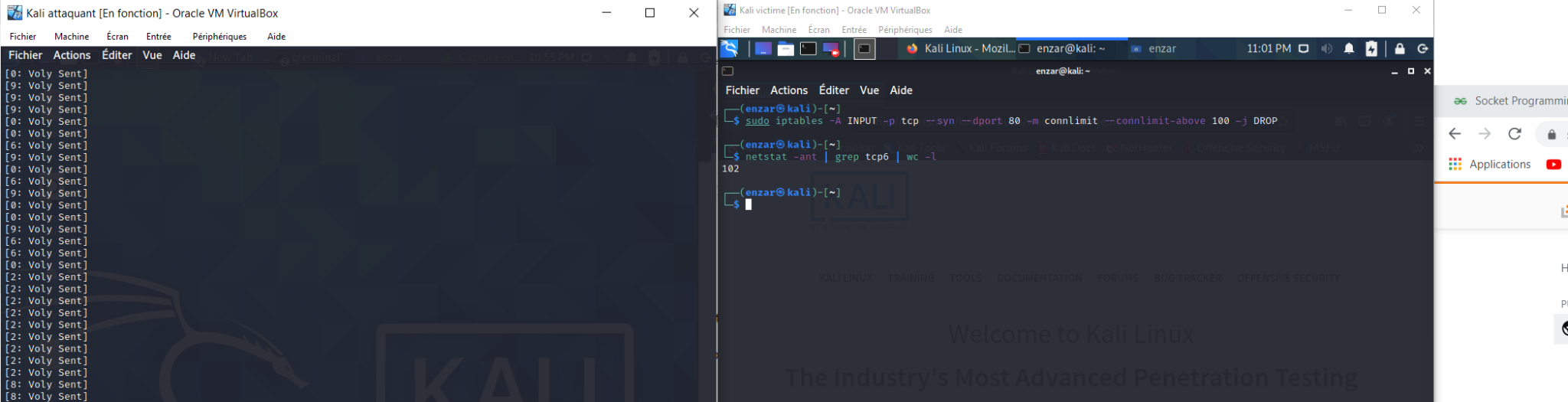
**Proposez une première ligne de défense contre ce type d’attaque et testez-la.**

Les pare-feu peuvent être utilisés pour arrêter les attaques DoS simples en bloquant tout le trafic provenant d'un attaquant en identifiant son IP.

Dans notre cas plusieurs solutions s'offrent à nous :

* Il est possible de limiter le nombre de connexions comme vu précédemment dans ce tpTP. Si l’attaquant ne randomise pas son adresse source alors son attaque pourra être bloquée via la commande : “*iptables -A INPUT -p tcp --syn --dport 80 -m connlimit --connlimit-above 100 -j DROP”.*

On peut voir queles connexions établies par l'outil xerxes ont été limitées à 100. L’attaquant aurait pu contourner cette protection en modifiant son adresse source.



* Une méthode un peu plus robuste est d’utiliser un script anti-ddos open source proposant des règles iptables utiles pour lutter contre les attaques dos et ddos.

<https://github.com/anti-ddos/Anti-DDOS>