

Version	Nom du documents	Auteur	Date	Commentaire
1	Déni de service (DOS) HTTP et mise en place d'une contremesure via HAProxy	Guillaume De Vargas	06/03/25	Création du document

Déni de service (DOS) HTTP et mise en place d'une contremesure via HAProxy

Table des matières

1)	Présentation des solutions	3
a)	Debian 12	3
b)	Apache2	4
c)	Kali.....	5
d)	HAProxy.....	6
2)	Contexte	7
a)	Schéma du contexte	7
b)	Éléments nécessaires à la mise en place du contexte	8
3)	Réalisation d'un déni de service (DOS) http via la machine Kali	9
4)	Mise en place d'une contremesure via répartition de charge.....	13
5)	Conclusion du TP	19

1) Présentation des solutions

a) Debian 12



Debian est une distribution Linux, composée presque exclusivement de logiciels libres. Elle est développée par le « Debian Project », une organisation communautaire fondée le 16 août 1993.

Les versions de Debian :

Debian a toujours au moins trois versions activement entretenues : stable, testing et unstable

La distribution stable contient la dernière distribution officiellement sortie de Debian.

C'est la version de production de Debian, celle que nous recommandons en premier d'utiliser.

Actuellement, la distribution stable de Debian est la version 12, nom de code *Bookworm*.

Elle a été initialement publiée en tant que version 12.0 le 10 juin 2023 et sa dernière mise à jour, version 12.7, a été publiée le 31 août 2024.

La distribution testing contient les paquets qui n'ont pas encore été acceptés dans la distribution stable, mais qui sont en attente de l'être.

Le principal avantage d'utiliser cette distribution est qu'elle contient des versions plus récentes des logiciels.

La distribution unstable est celle sur laquelle les activités de développement se déroulent. Généralement, cette distribution est utilisée par les développeurs et par ceux qui aiment vivre sur le fil.

Il est recommandé que les personnes qui l'utilisent s'abonnent à liste de diffusion `debian-devel-announce` pour recevoir les notifications de modification majeure, par exemple, les mises à niveau pouvant casser le système.

Cycle de vie des distributions :

Debian annonce régulièrement une nouvelle version stable. Le cycle de vie d'une version couvre cinq ans : une prise en charge complète pendant les trois premières années suivies de deux années supplémentaires de prise en charge à long terme (Long Term Support – LTS).

b) Apache2

Le logiciel libre Apache HTTP Server (Apache) est un serveur HTTP créé et maintenu au sein de la fondation Apache. Jusqu'en avril 2019, ce fut le serveur HTTP le plus populaire du World Wide Web. Il est distribué selon les termes de la licence Apache.

Apache est conçu pour prendre en charge de nombreux modules lui donnant des fonctionnalités supplémentaires : interprétation du langage Perl, PHP, Python et Ruby, serveur proxy, Common Gateway Interface, Server Side Includes, réécriture d'URL, négociation de contenu, protocoles de communication additionnels, etc.

Néanmoins, il est à noter que l'existence de nombreux modules Apache complexifie la configuration du serveur web. En effet, les bonnes pratiques recommandent de ne charger que les modules utiles : de nombreuses failles de sécurité affectant uniquement les modules d'Apache sont régulièrement découvertes.

c) Kali



Kali Linux est une distribution GNU/Linux basée sur Debian, lancée le 13 mars 2013. Elle succède à BackTrack et est développée par Offensive Security. Kali Linux est principalement conçue pour les tests de sécurité des systèmes d'information, notamment pour les tests d'intrusion.

Kali Linux comprend plus de 600 outils dédiés à la cybersécurité, notamment pour :

- Tests d'intrusion: Metasploit, Armitage, Burp Suite, OWASP ZAP
- Analyse réseau : Nmap, Wireshark, Ettercap, Kismet
- Cassage de mots de passe : John the Ripper, Hashcat, Hydra
- Ingénierie sociale : Social-Engineer Toolkit
- Exploitation de vulnérabilités : Sqlmap, Nikto, BeEF

Différence entre les versions Red, Blue et Purple :

● Kali Red (Attaque)

Focalisée sur les tests d'intrusion (pentesting) et le hacking éthique.

Contient des outils d'exploitation, de post-exploitation et d'évasion (ex: Metasploit, Nmap, Burp Suite, etc.).

Destinée aux professionnels offensifs et aux chercheurs en cybersécurité.

● Kali Blue (Défense)

Conçue pour les équipes de défense (Blue Team), spécialisée en monitoring, réponse aux incidents et threat hunting.

Inclut des outils pour l'analyse des logs, la détection des attaques et la sécurité réseau (ex: Suricata, Splunk, Wireshark, etc.).

● Kali Purple (Hybride - Offense + Défense)

Une version orientée vers la cybersécurité défensive et offensive.

Convient aux professionnels qui veulent comprendre les attaques et les techniques de défense.

Intègre à la fois des outils de Red Team et de Blue Team.

En résumé, Red = attaque, Blue = défense, Purple = mixte.

d) HAProxy



HAProxy

HAProxy (**H**igh **A**vailability **P**roxy) est un logiciel open-source performant utilisé comme **reverse proxy** et **répartiteur de charge (load balancer)**. Il est particulièrement prisé pour sa **rapidité, sa fiabilité et son efficacité** dans la gestion du trafic réseau, notamment pour les architectures web et cloud.

Fonctionnalité de Reverse Proxy

En mode **reverse proxy**, HAProxy agit comme un intermédiaire entre les clients et les serveurs backend :

- Cache et compression** des réponses pour optimiser la performance.
 - Sécurité améliorée** : il masque l'IP des serveurs backend et protège contre les attaques DDoS.
 - Gestion SSL/TLS** : il peut terminer le chiffrement HTTPS avant de transférer les requêtes en HTTP aux serveurs backend.
 - Filtrage et redirection** des requêtes en fonction de règles définies (ex: redirection HTTP vers HTTPS).
-

Fonctionnalité de Répartiteur de Charge

HAProxy permet de **distribuer le trafic** entre plusieurs serveurs pour équilibrer la charge et éviter la surcharge d'un seul serveur :

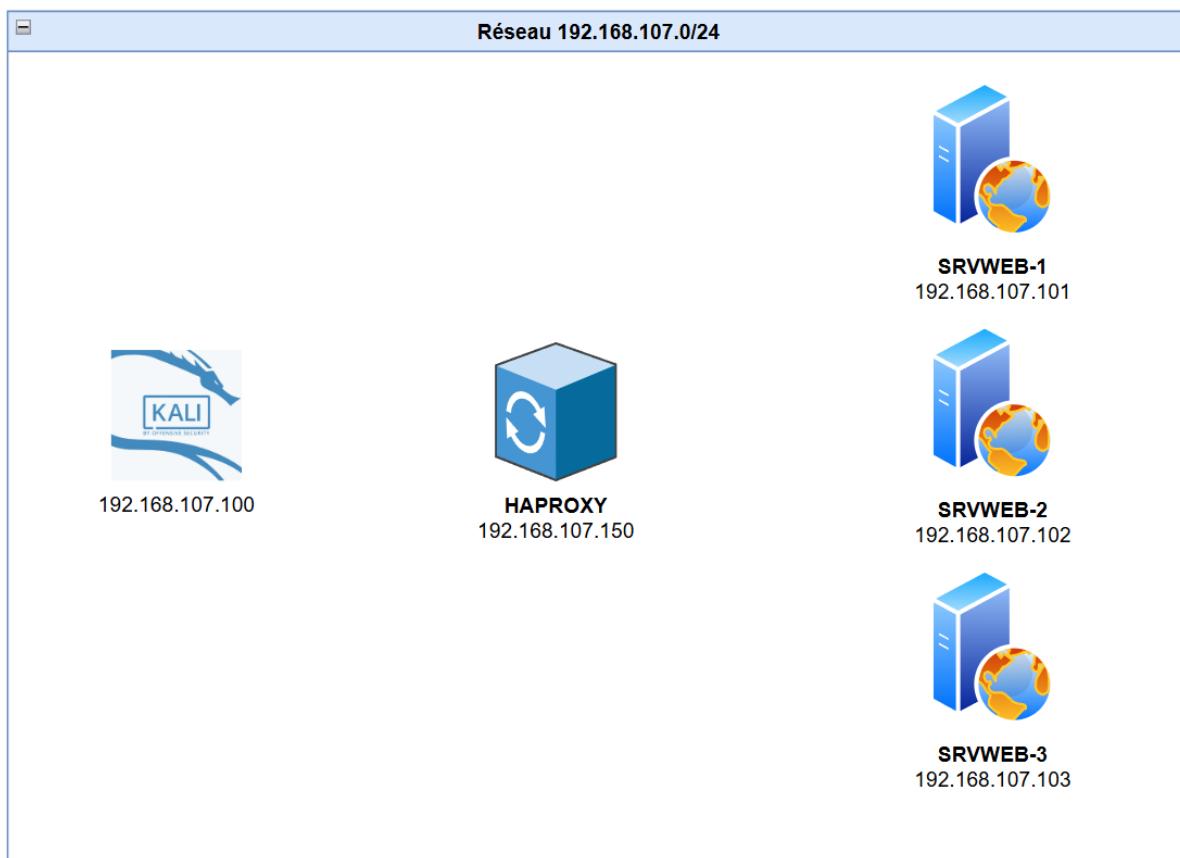
- Méthodes d'équilibrage** : Round Robin, Least Connections, Source IP, etc.
- Vérification de l'état des serveurs** (health checks) pour éviter d'envoyer du trafic à un serveur défaillant.
- Scalabilité** : il permet d'ajouter ou retirer des serveurs backend dynamiquement.
- Sticky sessions** : assure qu'un client est toujours dirigé vers le même serveur pour certaines applications.

2) Contexte

a) Schéma du contexte

Un reverse proxy est souvent configuré pour écouter les requêtes provenant de l'extérieur (réseau public) et les rediriger vers des machines en interne.

Dans notre cas, afin de faciliter la mise en place nous intégrerons l'ensemble des machines dans un réseau unique.



Vous pouvez évidemment adapter le troisième octet du réseau (192.168.X.0/24) en fonction de votre environnement de travail (réseau NAT de votre hyperviseur de niveau 2).

b) Éléments nécessaires à la mise en place du contexte

Dans ce contexte il faut :

- 4 Machines virtuelles Debian12 (2 à 4 cpu / 4go RAM / 20go de stockage / réseau NAT)
- 1 Machines virtuelles Kali dernière version dans le même réseau que les machines Debian

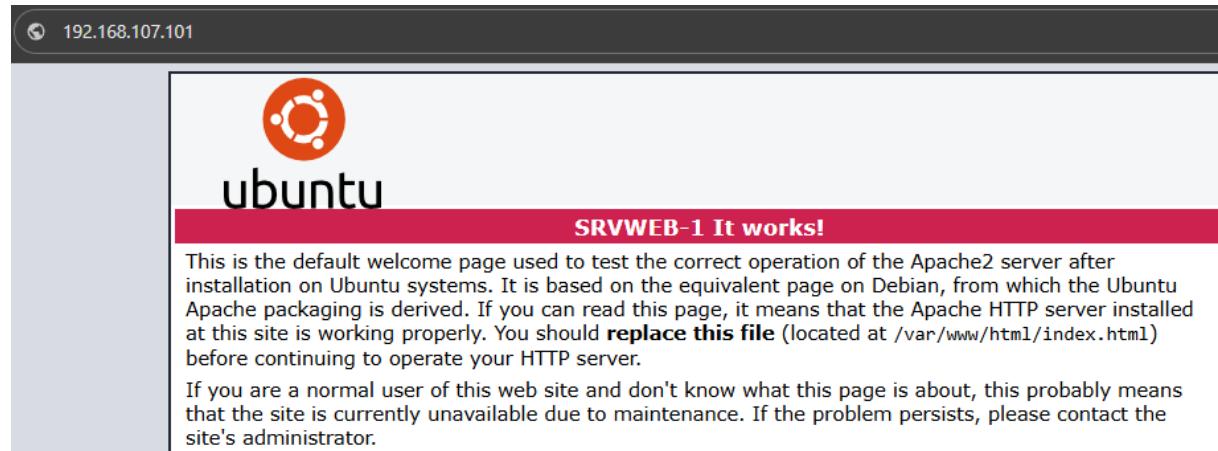
Configuration des machines Debian12 :

Sur les 3 machines SRVWEB-X :

- Configurer l'ip fixe du serveur
- Installer le serveur web Apache2
- Modifier le fichier /var/www/html/index.html afin que ce dernier affiche le numéro du serveur web :

```
<div class="content_section floating_element">
    <div class="section_header section_header_red">
        <div id="about"></div>
        SRVWEB-1 It works!
    </div>
```

Exemple pour le fichier index.html du SRVWEB-1



Page http d'accueil du SERVWEB-1

Tester l'accès aux 3 serveurs web avant de démarrer l'activité !

3) Réalisation d'un déni de service (DOS) http via la machine Kali

Sur votre machine Kali commencer par mettre à jour les paquets :

```
(kali㉿kali)-[~]
$ sudo apt update
```

Puis installer le paquet slowhttptest :

```
(kali㉿kali)-[~]
$ sudo apt install slowhttptest
```

Nous allons maintenant lancer notre attaque DOS afin de rendre indisponible le serveur web **SRVWEB-1** :

```
slowhttptest -c 1000 -H -g -o slowhttp -i 10 -r 200 -t GET -u http://192.168.107.101
```

Explication ligne par ligne :

- ◆ slowhttptest : C'est un outil utilisé pour tester la résistance d'un serveur web aux attaques de type Slow HTTP (attaque DoS qui envoie des requêtes incomplètes pour épuiser les connexions du serveur).
- ◆ -c 1000 : Spécifie le nombre total de connexions ouvertes simultanément (1000 connexions).
- ◆ -H : Lance une attaque Slowloris, qui consiste à envoyer des requêtes HTTP incomplètes pour garder les connexions ouvertes et saturer le serveur.
- ◆ -g : Active la génération de graphiques/statistiques pour analyser les résultats.
- ◆ -o slowhttp : Définit le nom du fichier de sortie pour enregistrer les résultats de l'attaque (slowhttp.html et autres fichiers).
- ◆ -i 10 : Définit l'intervalle d'envoi des en-têtes HTTP en secondes (10 secondes entre chaque envoi). Cela empêche le serveur de fermer la connexion en pensant qu'elle est inactive.
- ◆ -r 200 : Spécifie le taux de nouvelles connexions ouvertes par seconde (200 connexions/s).
- ◆ -t GET : Utilise la méthode HTTP GET pour l'attaque. D'autres méthodes possibles : POST (-t POST) pour une attaque Slow POST.
- ◆ -u <http://192.168.107.101> : Spécifie l'URL cible de l'attaque (dans ce cas, le serveur 192.168.107.101).

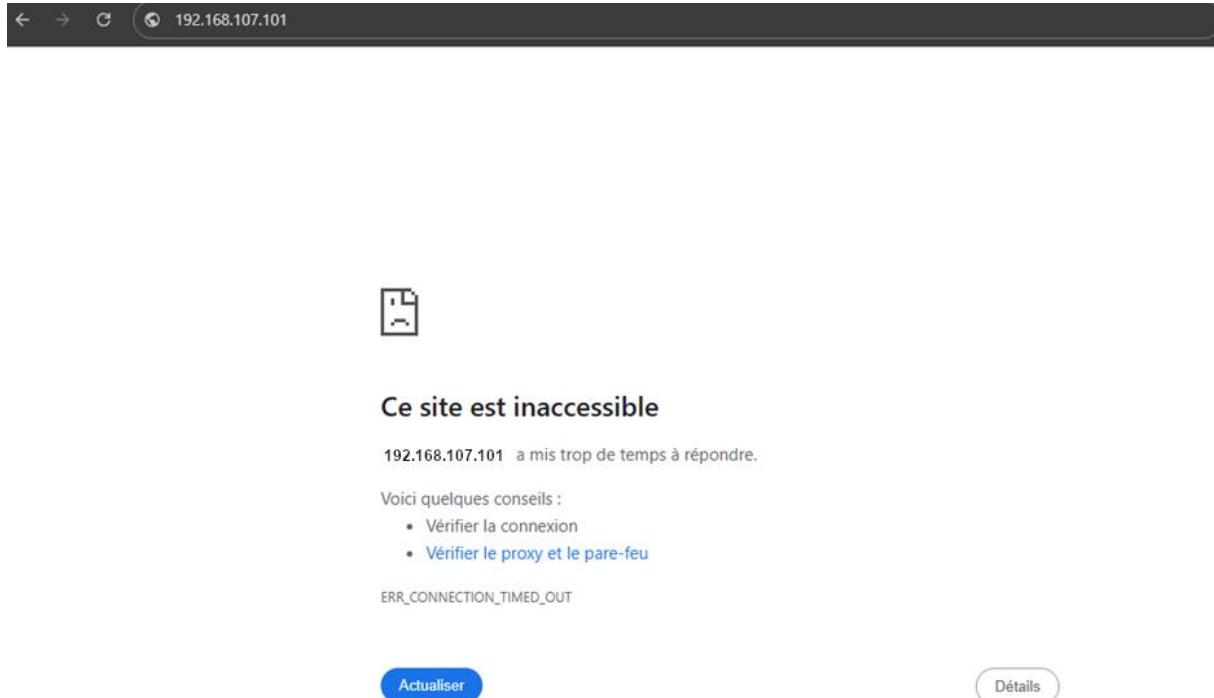
```
kali@kali: ~
File Actions Edit View Help
Thu Mar 6 09:28:07 2025:
slowhttptest version 1.9.0
- https://github.com/shekyan/slowhttptest -
test type: SLOW HEADERS
number of connections: 1000
URL: http://192.168.107.101/
verb: GET
cookie:
Content-Length header value: 4096
follow up data max size: 68
interval between follow up data: 10 seconds
connections per seconds: 200
probe connection timeout: 5 seconds
test duration: 240 seconds
using proxy: no proxy

Thu Mar 6 09:28:07 2025:
slow HTTP test status on 35th second:

initializing: 0
pending: 408
connected: 367
error: 0
closed: 225
service available: NO
```

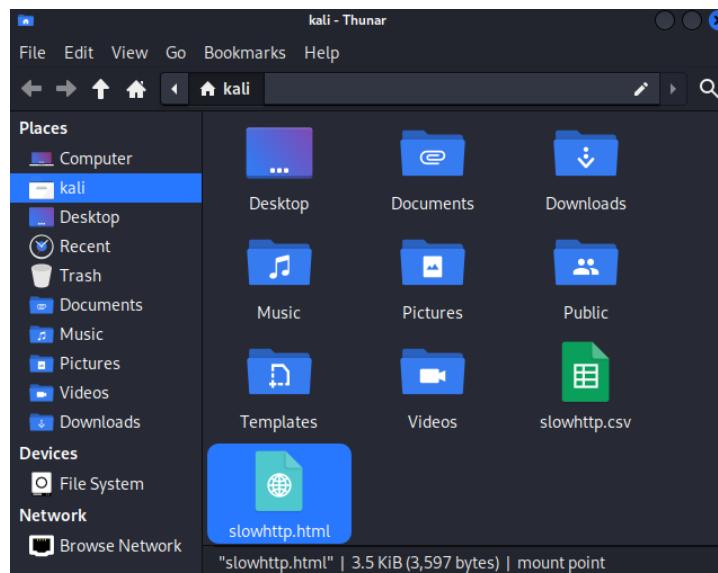
Lorsque vous voyez apparaître **NO** à côté de **service available** tester plusieurs fois via votre navigateur d'accéder ou de rafraîchir la page du serveur web de SRVWEB-1.

Logiquement le site devrait être indisponible ou mettre beaucoup de temps avant de retourner sa réponse :



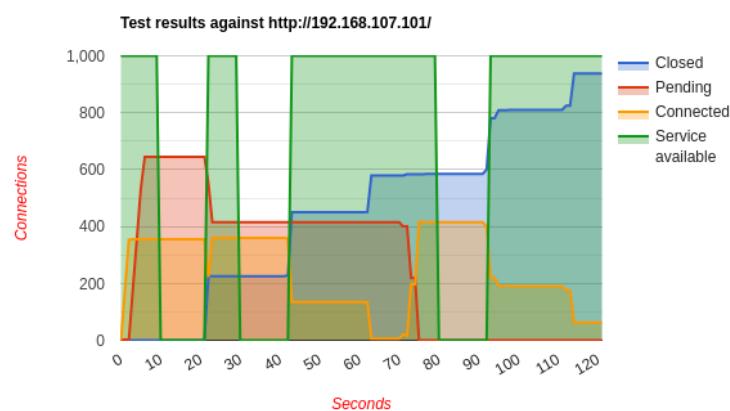
Une fois l'attaque terminée ouvrir le rapport slowhttp.html qui se trouve dans votre dossier /home/kali/ ou dans le répertoire courant que vous utilisez dans le terminal :

A screenshot of a terminal window on a Kali Linux system. The title bar says "kali@kali: ~". The terminal displays the output of a "slowhttp" test. The configuration parameters shown include: URL: http://192.168.107.101/, verb: GET, cookie: (empty), Content-Length header value: 4096, follow up data max size: 68, interval between follow up data: 10 seconds, connections per seconds: 200, probe connection timeout: 5 seconds, test duration: 240 seconds, and using proxy: no proxy. The test started at Thu Mar 6 09:29:32 2025 and ended at Thu Mar 6 09:29:34 2025. The status message "slow HTTP test status on 120th second:" is followed by connection statistics: initializing: 0, pending: 0, connected: 56, error: 0, closed: 944, and service available: YES. The terminal concludes with "Test ended on 121th second", "Exit status: No open connections left", "CSV report saved to slowhttp.csv", and "HTML report saved to slowhttp.html". The prompt "(kali㉿kali)-[~]" is visible at the bottom.



Test parameters

Test type	SLOW HEADERS
Number of connections	1000
Verb	GET
Content-Length header value	4096
Cookie	
Extra data max length	68
Interval between follow up data	10 seconds
Connections per seconds	200
Timeout for probe connection	5
Target test duration	240 seconds
Using proxy	no proxy



Comme constater depuis le navigateur, lors de l'attaque, le serveur web du SRVWEB1 a rencontré des problèmes de disponibilité. Sur le graphique on peut voir plusieurs périodes où le service était indisponible.

4) Mise en place d'une contremesure via répartition de charge

- Sur votre 4ème machine debian12 commencer par mettre à jour les paquets puis installer HAProxy :

```
sudo apt update
```

```
sudo apt install -y haproxy
```

- Mettre en place l'ip fixe 192.168.107.150/24 sur la machine

Nous allons maintenant configurer le reverse proxy et la répartition de charge d'HAProxy :

Editer les éléments suivant dans le fichier /etc/haproxy/haproxy.cfg :

```
timeout connect 5s
timeout client 10s
timeout server 10s
timeout http-request 10s
```

Nous allons maintenant déclarer **la FrontEnd** d'HAProxy en ajoutant les éléments suivant au fichier /etc/haproxy/haproxy.cfg :

```
frontend myfrontend
  bind 192.168.107.150:80
  default_backend myservers
```

Explication ligne par ligne :**◆ *frontend myfrontend***

Définit un frontend nommé myfrontend.

Le frontend est la partie qui reçoit les requêtes des clients avant de les rediriger vers un backend.

◆ *bind 192.168.107.150:80*

Spécifie l'adresse IP et le port sur lesquels HAProxy écoute les connexions entrantes.

Ici, HAProxy écoute sur l'IP 192.168.107.150 et le port 80 (HTTP).

Si on veut écouter sur toutes les interfaces, tu peux utiliser :

bind *:80

Pour activer HTTPS, il faut utiliser port 443 avec un certificat SSL :

bind 192.168.107.150:443 ssl crt /etc/haproxy/certs/mycert.pem

◆ *default_backend myservers*

Spécifie que les requêtes reçues par ce frontend seront envoyées au backend nommé myservers

Il ne nous reste plus qu'à déclarer **la BackEnd** d'HAProxy en ajoutant les éléments suivant au fichier /etc/haproxy/haproxy.cfg :

```
backend myservers
  balance roundrobin
  server server1 192.168.107.101:80 check
  server server2 192.168.107.102:80 check
  server server3 192.168.107.103:80 check
```

Explication ligne par ligne :**◆ *backend myservers***

Définit un backend nommé myservers.

Le backend est responsable de la gestion des serveurs vers lesquels HAProxy va envoyer les requêtes.

◆ *balance roundrobin*

Définit l'algorithme de répartition de charge entre les serveurs backend :

roundrobin signifie que HAProxy va distribuer les requêtes de manière équitable entre les serveurs, en passant au suivant à chaque nouvelle requête.

C'est l'algorithme par défaut, il fonctionne bien si tous les serveurs ont la même puissance.

◆ *server server1 192.168.107.101:80 check***◆ *server server2 192.168.107.102:80 check*****◆ *server server3 192.168.107.103:80 check***

Définit les serveurs backend qui recevront les requêtes :

server1, server2 et server3 sont les noms donnés aux serveurs (utilisés uniquement par HAProxy).

192.168.107.101:80, 192.168.107.102:80 et 192.168.107.103:80 sont leurs adresses IP et ports (ils écoutent sur le port 80 en HTTP).

check active le health check :

HAProxy envoie régulièrement une requête aux serveurs pour vérifier s'ils sont disponibles.

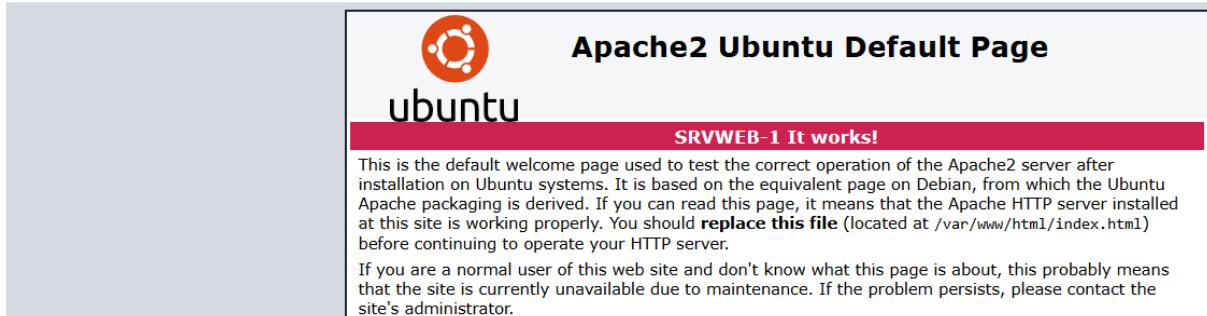
Si un serveur ne répond plus, il est temporairement retiré de la rotation pour éviter d'envoyer du trafic vers une machine en panne ou indisponible.

Il ne nous reste plus qu'à redémarrer le service haproxy sur la machine :

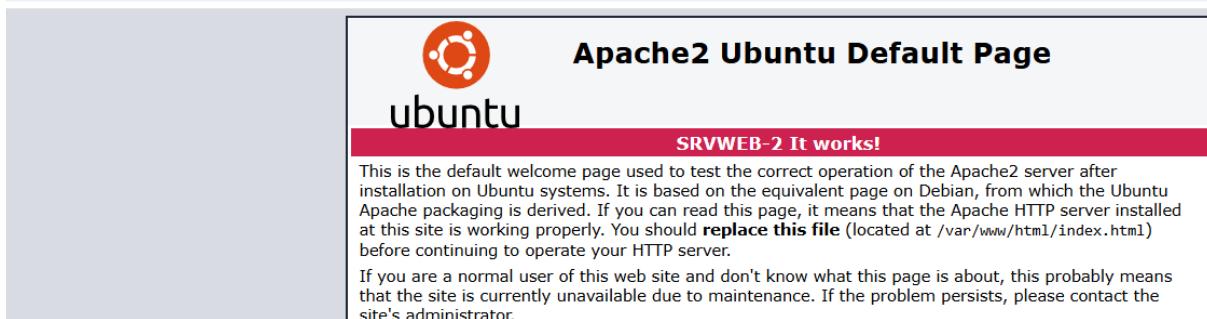
```
sudo systemctl restart haproxy
```

Afin de tester le bon fonctionnement de notre reverse proxy et de la répartition de charge saisissez l'adresse d'HAProxy 192.168.107.150 dans votre navigateur et rafraîchissez plusieurs fois la page :

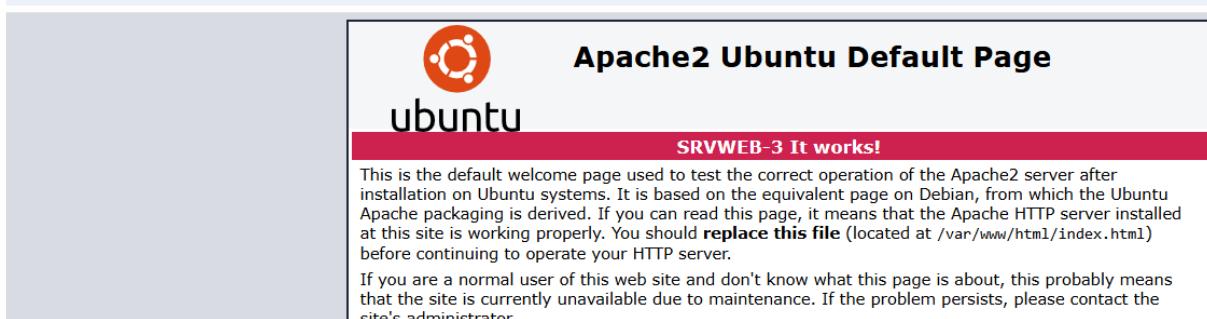
192.168.107.150



192.168.107.150



192.168.107.150



En passant par HAProxy nos requêtes sont bien réparties en RoundRobin à nos 3 serveurs. Nous pouvons le constater en voyant SRVWEB-1, SRVWEB-2, SRVWEB-3 s'afficher à tour de rôle à chaque rafraîchissement de la page.

Nous allons maintenant pouvoir effectuer de nouveau notre attaque de déni de service mais en passant cette fois par notre reverse proxy HAProxy qui assure la répartition de charge entre les 3 serveurs web.

```
Thu Mar 6 10:48:06 2025:  
      slowhttptest version 1.9.0  
      - https://github.com/shekyan/slowhttptest -  
test type:                      SLOW HEADERS  
number of connections:          1000  
URL:                            http://192.168.107.150/  
verb:                           GET  
cookie:  
Content-Length header value:   4096  
Follow up data max size:       68  
interval between follow up data: 10 seconds  
connections per seconds:      200  
probe connection timeout:     5 seconds  
test duration:                 240 seconds  
using proxy:                   no proxy  
  
Thu Mar 6 10:48:06 2025:  
slow HTTP test status on 15th second:  
  
initializing:      0  
pending:           0  
connected:         121  
error:             0  
closed:            879  
service available: YES  
Thu Mar 6 10:48:06 2025:  
Test ended on 15th second  
Exit status: No open connections left  
CSV report saved to slowhttp-haproxy.csv  
HTML report saved to slowhttp-haproxy.html  
  
[kali㉿kali)-[~]  
$
```

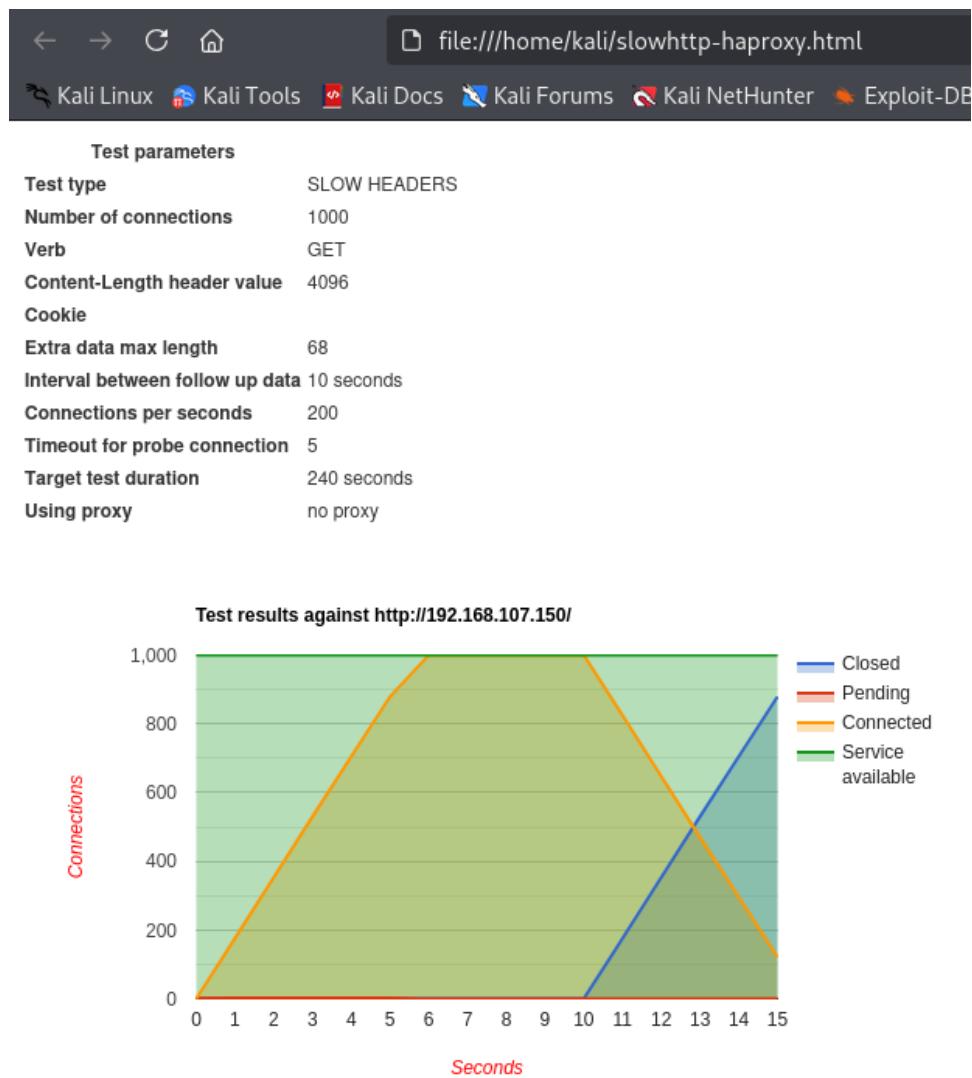
Sur la machine Kali relancer l'attaque en modifiant l'ip pour celle de HAProxy et le nom du rapport pour slowhttp-haproxy :

```
slowhttptest -c 1000 -H -g -o slowhttp-haproxy -i 10 -r 200 -t GET -u http://192.168.107.150
```

Pendant l'attaque (soyez rapide) essayer de d'accéder avec votre navigateur au site via l'adresse d'HAProxy (192.168.107.150). Logiquement vous devriez rencontrer beaucoup moins de problème !

Vous pouvez constater également que l'attaque a durée beaucoup moins longtemps car le nombre de requêtes HTTP à été traité beaucoup plus vite grâce à la répartition de charge entre les 3 serveurs !

Ouvrez maintenant le rapport slowhttp-haproxy.html qui se trouve dans votre dossier /home/kali/ ou dans le répertoire courant que vous utilisez dans le terminal :



Comme attendu la répartition de charge a permis de garantir la disponibilité du site pendant toute la durée de l'attaque !

5) Conclusion du TP

Ce TP nous a permis d'explorer une attaque **Slow HTTP DoS** et d'observer son impact sur un serveur web classique. Nous avons constaté qu'un simple serveur Apache, sans protection, est vulnérable aux attaques DoS, ce qui entraîne une saturation des connexions et une indisponibilité du service.

Afin de contrer cette attaque, nous avons mis en place **HAProxy** en tant que **reverse proxy** et **répartiteur de charge**. Grâce à la distribution du trafic entre plusieurs serveurs web, HAProxy a permis :

- Une meilleure résilience** face aux attaques en répartissant la charge.
- Une continuité de service** même sous une attaque DoS.
- Un équilibrage efficace** des requêtes avec l'algorithme *roundrobin*.
- Une détection des pannes** avec les *health checks* intégrés.

Finalement, la simulation d'attaque a montré une nette amélioration de la robustesse du service web une fois disponible en passant par la solution HAProxy. La répartition de charge a permis d'atténuer l'effet du DoS, rendant le site accessible malgré l'attaque.

Pourquoi un reverse proxy traite plus efficacement les requêtes TCP et HTTP ?

Un reverse proxy comme HAProxy agit comme un intermédiaire entre les clients et les serveurs backend, optimisant ainsi la gestion des connexions réseau.

Niveau TCP (Connexion bas niveau)

- ◆ Multiplexage des connexions : HAProxy garde des connexions TCP ouvertes vers les serveurs backend et réutilise ces connexions, réduisant ainsi la surcharge liée aux ouvertures/fermetures de connexions.
- ◆ Filtrage et sécurité : Il peut bloquer les connexions malveillantes ou limiter le nombre de connexions par client avant d'atteindre le serveur backend.

Niveau HTTP (Requêtes applicatives)

- ◆ Cache et compression : Un reverse proxy peut optimiser les requêtes HTTP en servant des pages en cache ou en compressant les réponses pour accélérer le chargement.
- ◆ Équilibrage de charge : HAProxy répartit les requêtes entre plusieurs serveurs backend, évitant qu'un serveur unique soit surchargé.
- ◆ Gestion SSL/TLS : Il peut terminer le chiffrement HTTPS avant de transmettre les requêtes en HTTP aux serveurs backend, réduisant leur charge de calcul.

Conclusion

- ✓ Au niveau TCP, HAProxy gère efficacement les connexions, réduisant les surcharges.
- ✓ Au niveau HTTP, il optimise et filtre les requêtes, améliorant la rapidité et la sécurité du service.

C'est pourquoi un reverse proxy est plus performant qu'un serveur web direct.