Sécurisation de services réseaux
TP2

SOMMAIRE

Introduction

Ce rapport détaille l'implémentation et l'analyse de la sécurisation d'un service web, en mettant en place des mécanismes de haute disponibilité et de protection des communications grâce au protocole TLS. L'objectif principal est d'assurer la disponibilité, l'intégrité et la confidentialité des échanges tout en garantissant une répartition efficace des charges.

Objectifs

- Installer et configurer un serveur web Apache sécurisé.
- Mettre en place le protocole HTTPS avec un certificat TLS.
- Implémenter un équilibrage de charge avec HAProxy.
- Vérifier la disponibilité et la redondance du service.

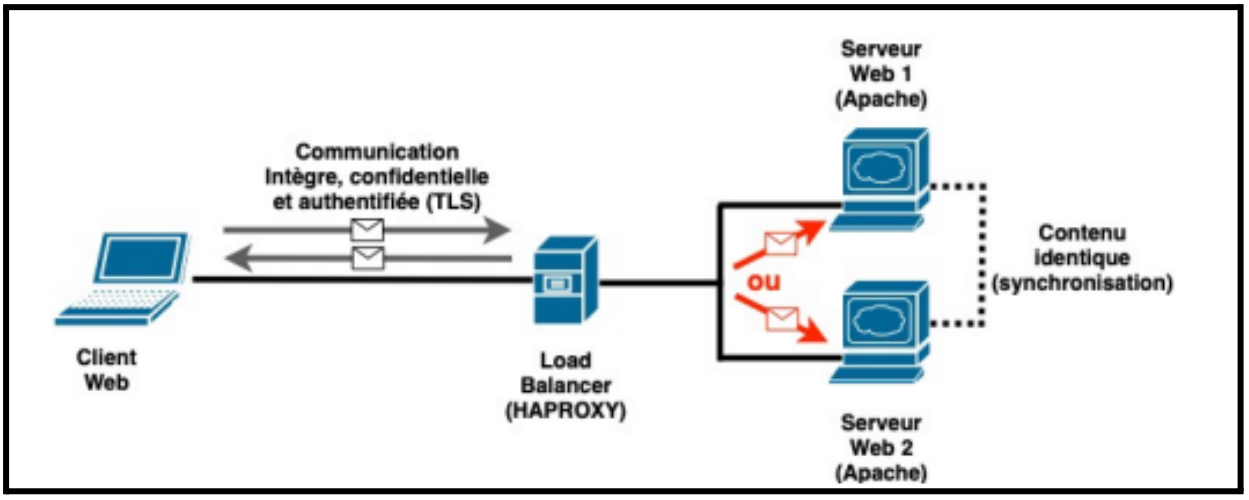
Matériel et Logiciels Utilisés

- **Matériel** : Machines virtuelles sous Debian.
- **Logiciels** : Apache2, OpenSSL, HAProxy.

Architecture du Système

L'infrastructure repose sur trois machines virtuelles distinctes :

- **Deux serveurs Apache2** assurant l'hébergement du site web.
- **Un serveur HAProxy** jouant le rôle de répartiteur de charge.
- **Un client web** utilisé pour effectuer des tests et vérifier la stabilité du système.



TÂCHE 1 – INSTALLATION DE APACHE (SUR LE PREMIER SERVEUR)

Étape 2 : Installation d'Apache sur le poste Serveur HTTP.

Installation d'Apache

Les commandes suivantes permettent d'installer et de mettre à jour Apache sur le serveur :

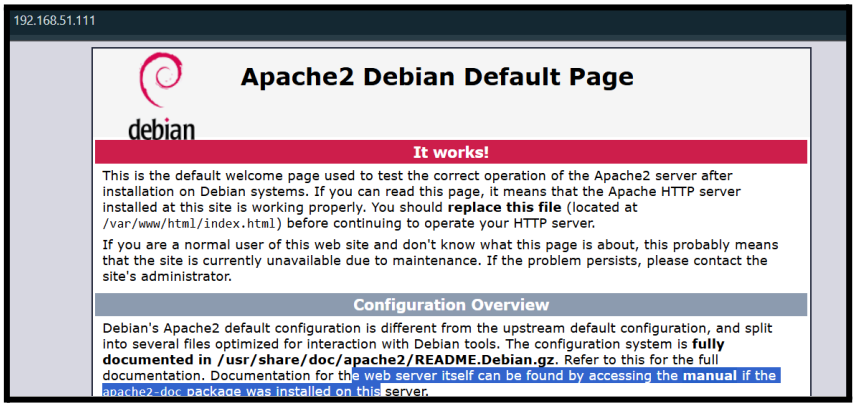
```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install apache2
```

Étape 3 : Vérifications

Une fois l'installation terminée, on peut vérifier que le serveur Apache est bien actif et écoute sur le port 80 avec la commande :

```
etudiant@debian:~$ netstat -ntl
Connexions Internet actives (seulement serveurs)
Proto Recv-Q Send-Q Adresse locale        Adresse distante      Etat
tcp6       0      0 :::80                   :::*                  LISTEN
```

- Il faut maintenant lancer un navigateur sur la machine physique et consulter l'URL http://Adresse_IP_du_serveur. Vous devez obtenir l'affichage d'une page web de présentation d'Apache.

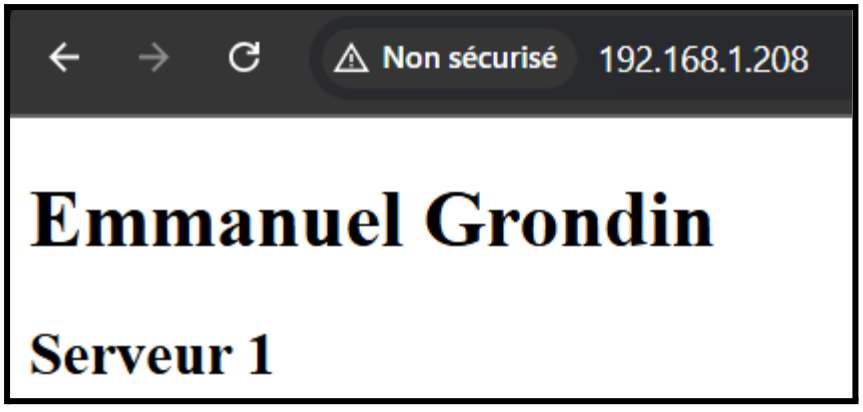


TÂCHE 2 – MODIFICATION DE LA PAGE D'ACCUEIL DU SITE

Quand il faut ouvrir le fichier *000-default.conf* dans le dossier *sites-available*, et lire l'argument de la directive *DocumentRoot* : il s'agit du dossier du système de fichiers local (de l'ordinateur serveur) qui contient les fichiers (code HTML, images, etc) du site web.

```
DocumentRoot /var/www/html
```

On modifie un minimum la page du serveur 1:



TÂCHE 3 – SÉCURISATION D'UN SITE WEB : HTTPS

Étape 1 : Génération d'un certificat auto-signé

Pour générer un certificat auto-signé, utilisation de l'outil OpenSSL.

Se placer dans le dossier /etc/apache2, puis exécuter la commande suivante. Quelques renseignements personnels seront demandés (pour le FQDN, indiquer l'URL du site sécurisé, par exemple www.site-de-emmanuelsecure.com, à adapter si nécessaire).

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -out
/etc/apache2/server.pem -keyout /etc/apache2/server.key
```

```
Country Name (2 letter code) [AU]:RE
State or Province Name (full name) [Some-State]:reunion
Locality Name (eg, city) []:saint-pierre
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IUT
Organizational Unit Name (eg, section) []:RT
Common Name (e.g. server FQDN or YOUR name) []:EG
Email Address []
```

Explications :

- X509 est un format de certificat numérique, standardisé par l'IETF (RFC 2527) ;
- On génère ici un couple de clés RSA, avec une longueur de clé de 2048 bits (valeur préconisée actuellement pour RSA) ;
- Le certificat (qui contient la clé publique), est stocké dans le fichier server.pem : il sera transmis aux clients HTTPS ;
- La clé privée est stockée dans le fichier server.key : elle ne sera jamais transmise sur le réseau.

Visualiser le contenu des fichiers server.pem et server.key (commande cat), puis obtenir un affichage plus explicite du certificat :

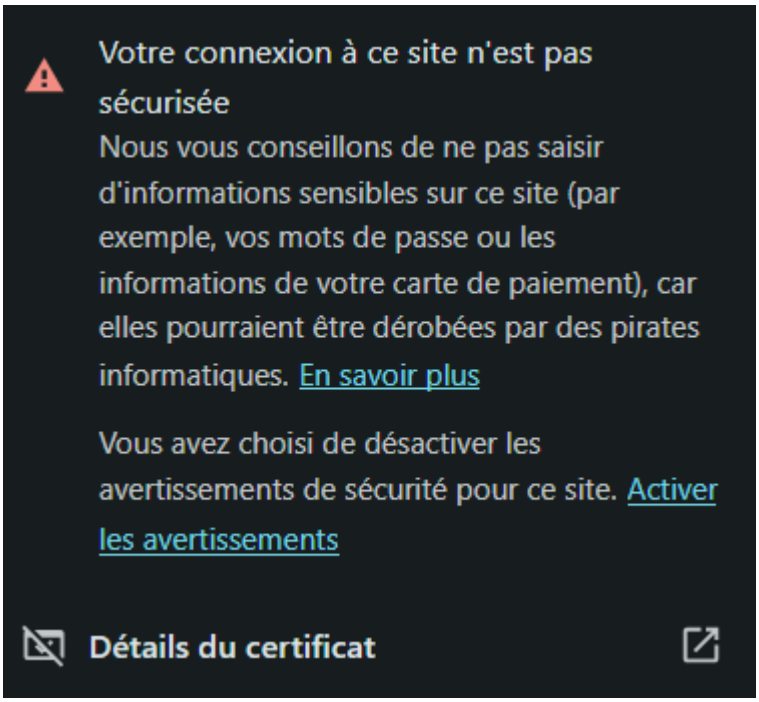
```
#openssl x509 -noout -text -in server.pem
```

```
root@debianVM:/etc/apache2# openssl x509 -noout -text -in server.pem
Certificate:
    Data:
```

```
    Version: 3 (0x2)
    Serial Number:
        76:3b:5f:f8:e8:17:2e:da:aa:df:2e:a7:01:5a:f4:e1:af:a8:4e:b6
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C = RE, ST = reunion, L = saint-pierre, O = IUT, OU = RT, CN = EG
    Validity
        Not Before: Feb 24 05:24:14 2025 GMT
        Not After : Feb 24 05:24:14 2026 GMT
    Subject: C = RE, ST = reunion, L = saint-pierre, O = IUT, OU = RT, CN = EG
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
            RSA Public-Key: (2048 bit)
            Modulus:
                00:a7:03:af:ac:f3:9c:63:de:73:ab:bf:af:15:e1:
                51:ef:67:96:0c:a2:f9:d1:3a:fa:1e:8d:34:08:21:
                b5:c6:45:1e:a4:ee:79:10:fb:e5:cc:e0:ef:b2:85:
                27:b9:a7:b9:b4:79:65:f1:68:4f:3a:e4:6f:1d:b6:
                ed:a3:fe:25:96:08:65:cd:91:be:48:68:d3:e9:eb:
                41:08:f5:2d:c0:e1:c1:9c:00:cd:c3:83:78:aa:f1:
                20:b6:63:66:02:4c:eb:ad:23:2e:89:13:c2:8a:6d:
                10:97:c1:37:ee:ae:ad:bf:ab:e8:b8:f2:b9:5c:45:
                88:0b:66:92:fd:33:a0:17:b8:30:0a:28:21:e1:04:
                b8:82:a8:05:09:88:f0:7c:e2:a3:6b:c8:b1:37:47:
                f2:e0:1f:56:f2:f4:86:e3:c1:85:e1:bf:f0:dd:1c:
                3b:8d:bd:08:ce:71:a7:cd:f3:6e:7b:cf:d4:aa:cf:
                54:11:15:df:32:f3:83:41:91:dd:48:45:cb:f4:52:
                9f:2d:23:b7:6d:8d:fc:77:71:72:cb:9f:70:3e:3b:
                78:99:be:35:90:70:88:1c:c3:87:dc:e2:e9:1c:87:
                56:31:c2:a5:6e:e3:35:90:f0:23:81:aa:ab:25:66:
                38:57:33:d3:3f:62:3c:96:1a:cc:d0:4d:de:d3:a9:
                0e:bd
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Subject Key Identifier:
            61:2D:CA:5C:F3:88:73:E6:8C:8D:3E:42:C5:A2:09:8E:AF:D8:D4:44
        X509v3 Authority Key Identifier:
            keyid:61:2D:CA:5C:F3:88:73:E6:8C:8D:3E:42:C5:A2:09:8E:AF:D8:D4:44

        X509v3 Basic Constraints: critical
            CA:TRUE
Signature Algorithm: sha256WithRSAEncryption
    35:a0:02:35:94:37:fb:3e:e3:e2:75:33:0c:33:9e:b6:e2:90:
    11:24:3e:8f:8f:b5:ae:a7:5b:9c:4c:67:3a:98:64:f7:da:91:
    10:fb:23:05:18:73:0f:3d:bf:6f:60:aa:85:6c:a6:c2:08:bf:
    bb:60:8b:d7:ab:81:4c:5e:1f:80:03:90:71:c1:da:1a:01:6b:
    3c:60:c1:31:e1:11:39:fb:12:d8:a7:ae:bb:71:99:28:e3:8a:
    f5:b8:a9:8a:4b:f3:93:b4:92:be:b9:f0:65:38:07:5e:f8:20:
    59:a0:63:75:58:9d:33:51:fa:36:2b:c3:c6:56:79:ec:fc:1d:
    b2:b5:85:fc:27:f8:1e:d7:1f:21:ef:3a:55:fb:05:ac:6e:b9:
    ab:54:d5:62:2d:e2:9d:c8:38:c5:c6:a7:27:70:67:0b:a6:5d:
    c5:44:4d:fe:5a:fd:47:73:3c:3f:22:93:43:8d:18:35:82:96:
    a5:db:e7:a8:63:1c:95:f4:2e:84:e3:65:1d:48:32:d2:88:09:
    c7:e8:f2:62:7c:5b:ac:18:94:49:33:c9:a7:6d:41:bb:e0:b7:
```

```
36:4c:14:0b:48:26:d4:fb:d3:13:aa:48:be:cf:a8:e9:4e:38:
c9:13:90:12:26:f8:f5:54:e6:f5:4e:8f:56:64:2e:3a:ec:8e:
b9:6e:95:5d
```



On peut vérifier le certificat de cette façon en ascendant sur le site.

Étape 2 : Création d'un site sécurisé et activation de HTTPS

Créer un dossier html_ssl dans /var/www, puis éditer un fichier index.html personnalisé dans ce dossier (différent de celui présent dans le dossier du site non sécurisé).

```
root@debian:/var/www# mkdir html_ssl
root@debian:/var/www# ls
html  html_ssl
root@debian:/var/www#
```

Activer le module ssl :

```
#sudo a2enmod ssl
```

La commande `sudo a2enmod ssl` est utilisée sur les systèmes Linux basés sur Debian (comme Ubuntu) pour activer le module **SSL** d'Apache.

- **`a2enmod`** signifie "**Apache2 enable module**" (activer un module Apache).
- **`ssl`** est le module qui permet à Apache de gérer les connexions sécurisées via HTTPS.

Puis le site sécurisé :

```
#sudo a2ensite default-ssl
```

La commande `sudo a2ensite default-ssl` est utilisée pour activer le site par défaut en HTTPS sur un serveur Apache sous Debian/Ubuntu.

- **`a2ensite`** signifie "**Apache2 enable site**" (activer un site Apache).
- **`default-ssl`** est la configuration par défaut d'Apache pour les connexions sécurisées en HTTPS.

```
etudiant@debian:~$ sudo a2enmod ssl
Considering dependency setenvif for ssl:
Module setenvif already enabled
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Enabling module socache_shmcb.
Enabling module ssl.
See /usr/share/doc/apache2/README.Debian.gz on how to configure SSL
and create self-signed certificates.
```

```
To activate the new configuration, you need to run:
  systemctl restart apache2

etudiant@debian:~$ sudo systemctl restart apache2
etudiant@debian:~$ sudo a2ensite default-ssl
Enabling site default-ssl.
To activate the new configuration, you need to run:
  systemctl reload apache2
etudiant@debian:~$ sudo systemctl restart apache2
etudiant@debian:~$
```

NB : des liens symboliques vers les fichiers ssl.conf et ssl.load, présents dans le dossier mods-available, doivent maintenant apparaître dans le dossier mods-enabled. De même, un lien symbolique vers le fichier default-ssl.conf, présent dans le dossier sites-available, a été créé dans le dossier sites-enabled.

```
root@debianVM:/etc/apache2/mods-available# ls ssl.conf ssl.load
ssl.conf  ssl.load

root@debianVM:/etc/apache2/mods-enabled# ls ssl.conf ssl.load
ssl.conf  ssl.load

root@debianVM:/etc/apache2/sites-available# ls default-ssl.conf
default-ssl.conf
```

La commande `sudo a2ensite default-ssl` active le site HTTPS par défaut d'Apache. Elle crée un **lien symbolique** du fichier de configuration `default-ssl.conf` (présent dans `sites-available`) vers `sites-enabled`, permettant ainsi son activation.

De même, la commande `sudo a2enmod ssl` active le module SSL d'Apache en créant des liens symboliques des fichiers `ssl.conf` et `ssl.load` (depuis `mods-available` vers `mods-enabled`).

En résumé : Ces commandes permettent d'activer le support HTTPS sur Apache en activant le module SSL et en appliquant la configuration par défaut.

Étape 3 : Configuration du site sécurisé

Dans le fichier default-ssl.conf, effectuer les modifications suivantes pour configurer le VirtualHost du site sécurisé. Ce fichier doit inclure au minimum les directives suivantes (certaines étant commentées dans le fichier par défaut généré lors de l'installation d'Apache) :

```
<VirtualHost *:443>
 DocumentRoot /var/www/html_ssl
 ServerName www.site-de-emmanuel-secure.com
 SSLEngine on
 SSLCertificateFile /etc/apache2/server.pem
 SSLCertificateKeyFile /etc/apache2/server.key
</VirtualHost>
```

NB : la directive SSLEngine permet d'activer le moteur SSL au sein d'un VirtualHost. Les deux dernières directives
sont (cf. étape 1).
Question : Pourquoi n'est-il pas nécessaire de modifier le fichier ports.conf ?

Il n'est pas nécessaire de modifier ports.conf parce qu'Apache écoute déjà sur le port 443 pour HTTPS.

Quand on active SSL avec a2enmod ssl et qu'on active le site sécurisé avec a2ensite default-ssl, Apache sait déjà qu'il doit utiliser le port 443.

Donc, pas besoin de toucher à ports.conf !

Étape 4 : Tests

Vérifier que le port 443 est ouvert : **#netstat -ntl**

```
root@debian:/etc/apache2/sites-available# netstat -ntl
Connexions Internet actives (seulement serveurs)
Proto Recv-Q Send-Q Adresse locale        Adresse distante       Etat
tcp6       0      0 :::443                  :::*                   LISTEN
```

Accéder au site sécurisé en HTTPS. Expliquer le message d'erreur (ou plutôt de warning) généré par le navigateur. Accepter une exception de sécurité (vous pouvez

faire confiance à vous même, ou à votre voisin). Que faudrait-il faire pour que le site soit « digne de confiance » ?

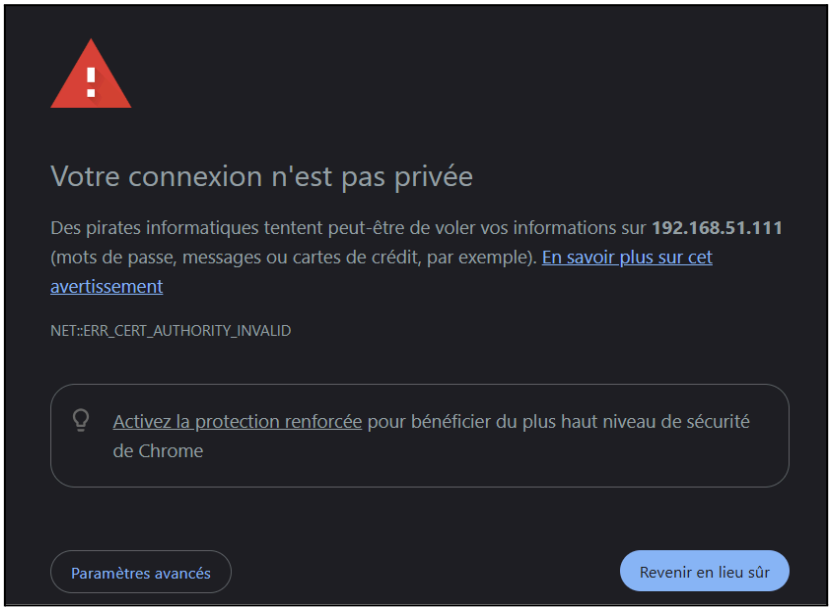
Pourquoi le navigateur affiche un warning ?

Le message d'alerte apparaît car le certificat SSL/TLS du site **n'est pas reconnu**. Cela peut être dû à un **certificat auto-signé**, **expiré**, ou **émis pour un autre domaine**.

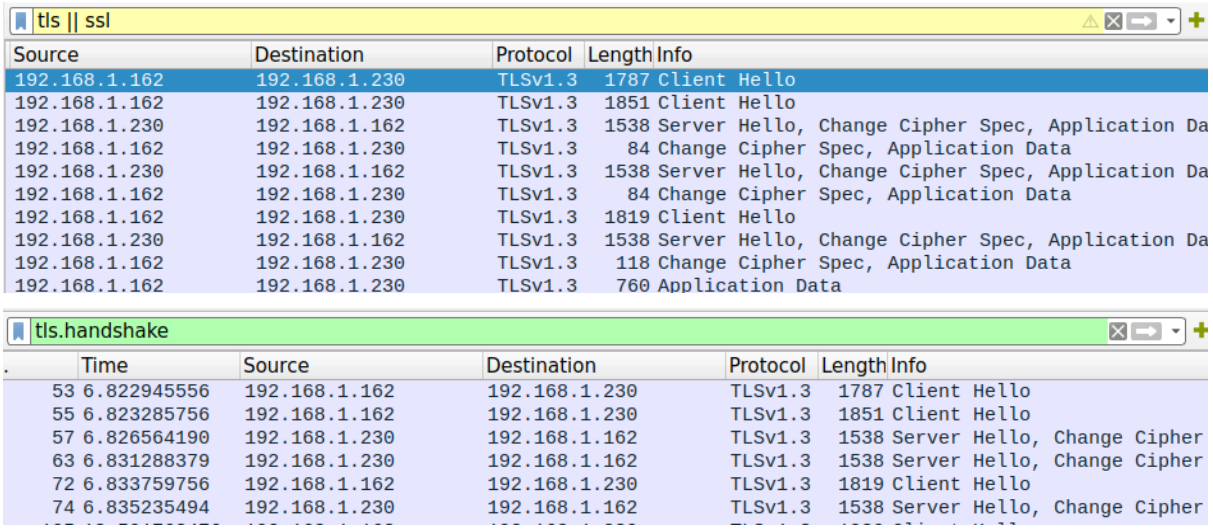
Comment rendre le site de confiance ?

- **Obtenir un certificat valide** d'une autorité reconnue (ex: Let's Encrypt).
- **Configurer correctement** le certificat pour qu'il corresponde au domaine.
- **Vérifier et renouveler** le certificat avant expiration.
- **Ajouter manuellement** le certificat à la liste de confiance en local.

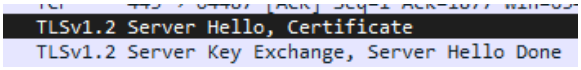
Exemple d'un WARNING :



Sur le client, lancer une capture Wireshark des trames échangées lors du chargement du site sécurisé.



On peut y trouver la trame où le serveur transmet le certificat donc une clé public pour permettre l'échange sécurisé :
"Server Hello, Certificate"



En particulier :

- Identifier les trames qui correspondent au Handshake Protocol ;
- Rechercher la trame dans laquelle le certificat est transmis par le serveur au client.
- Déterminer l'algorithme de chiffrement symétrique qui a été négocié pour le Record Protocol. Quelle est la longueur de la clé secrète ? Quels algorithmes ont permis l'échange de la clé secrète ?

Le serveur utilise le chiffrement **TLS_AES_256_GCM_SHA384**.
La clé publique du serveur est de **2048 bits** en RSA.

TACHE 4 – MISE EN PLACE DE LA HAUTE DISPONIBILITÉ DE HTTP

Étape 1 : Installation d'un second serveur Apache è Répéter les tâches 1 et 2 sur une seconde VM.

Étape 2 : Installation de HAPROXY

```
#sudo apt-get update
#sudo apt-get upgrade
#sudo apt-get install haproxy
```

Étape 3 : configuration de HAProxy

Elle s'effectue par l'édition du fichier de configuration /etc/haproxy/haproxy.cfg, dont la structure en quatre sections principales est la suivante :

```
global
    log /dev/log    local0
    log /dev/log    local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.sock mode 660 level admin expose-fd listeners
    stats timeout 30s
    user haproxy
    group haproxy
    daemon

    ca-base /etc/ssl/certs
    crt-base /etc/ssl/private

    ssl-default-bind-ciphers
ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-S
```

```
HA384:ECDHE-RSA-AES256-GCM-SHA384
    ssl-default-bind-ciphersuites
TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256
    ssl-default-bind-options ssl-min-ver TLSv1.2 no-tls-tickets

defaults
    log global
    mode http
    option httplog
    option dontlognull
    timeout connect 5000
    timeout client 50000
    timeout server 50000
    errorfile 400 /etc/haproxy/errors/400.http
    errorfile 403 /etc/haproxy/errors/403.http
    errorfile 408 /etc/haproxy/errors/408.http
    errorfile 500 /etc/haproxy/errors/500.http
    errorfile 502 /etc/haproxy/errors/502.http
    errorfile 503 /etc/haproxy/errors/503.http
    errorfile 504 /etc/haproxy/errors/504.http

frontend http_front
    bind 192.168.1.43:80
    default_backend web_servers

backend web_servers
    balance roundrobin
    server apache1 192.168.1.208:80 check
    server apache2 192.168.1.230:80 check
```

Explication rapide :

- **Section `global`**
  Définit les paramètres généraux de HAProxy :
  - Active les logs (`log /dev/log`).
  - Définit l'utilisateur (`user haproxy`) et le groupe (`group haproxy`).
  - Configure les options SSL par défaut (`ssl-default-bind-options` et `ssl-default-bind-ciphers`).

- **Section `defaults`**
  Définit les paramètres par défaut pour les proxys :
  - Mode HTTP (`mode http`).
  - Gestion des logs (`option httplog`).
  - Définit les **timeouts** pour les connexions et requêtes.
  - Spécifie les fichiers d'erreur HTTP personnalisés.
- **Section `frontend http_front`**
  - Lie HAProxy à l'IP `192.168.1.43` sur le port `80` (HTTP).
  - Redirige les requêtes vers le **backend** nommé `web_servers`.
- **Section `backend web_servers`**
  - Utilise l'algorithme **round-robin** pour équilibrer la charge entre les serveurs.
  - Deux serveurs web sont configurés :
    - `apache1` (`192.168.1.208:80`).
    - `apache2` (`192.168.1.230:80`).
  - Chaque serveur est surveillé (`check`).

Étape 4 : vérification du fonctionnement

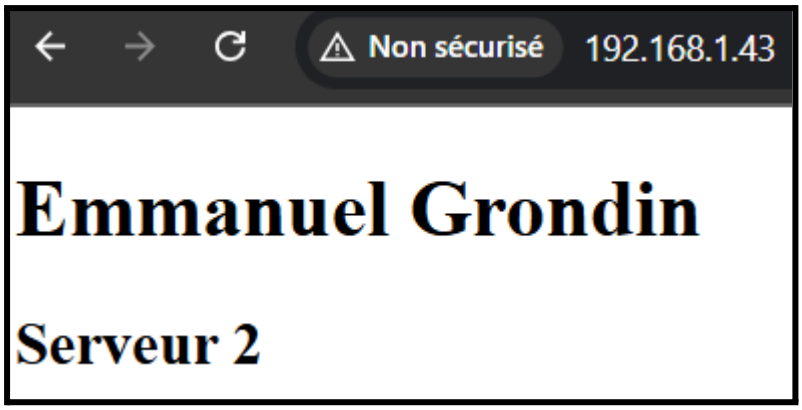
(1) Vérification de l'accessibilité des serveurs web en utilisant l'adresse IP du répartiteur de charge HAProxy dans le navigateur.

(2) Vérifier qu'à chaque rafraichissement de la page dans le navigateur, on change systématiquement de serveur répondant. Pourquoi ?

Round-Robin : Par défaut, HAProxy répartit les requêtes entrantes entre les serveurs de manière circulaire. À chaque nouvelle requête (comme un rafraîchissement de page), HAProxy sélectionne le serveur suivant dans la liste. Cela fait que, systématiquement, chaque rafraîchissement peut aboutir sur un serveur différent.
Sessions non persistantes : Sans configuration spécifique, chaque requête est traitée indépendamment des autres, ce qui signifie qu'il n'y a pas de lien entre les requêtes successives. À chaque rafraîchissement, HAProxy redirige la requête vers un autre serveur, selon l'ordre du round-robin.
Absence de cookies de session : Sans cookies de session ou mécanismes d'affinité, HAProxy ne garde pas en mémoire quel serveur a répondu à la requête précédente. Donc, à chaque rafraîchissement, il choisit un autre serveur, ce qui peut expliquer ce comportement.

(3) Éteindre l'un des deux serveurs (service apache2 stop) : vérifier que c'est maintenant toujours le serveur encore actif qui répond.

```
root@debianVM:~# systemctl stop apache2.service
```
sur serveur 1



(4) Remettre le serveur éteint en marche, et constater que l'alternance des deux serveurs est rétablie.

A chaque redémarrage du serveur on retrouve bien l'alternance entre les deux serveur à chaque rafraichissement de la page :

Étape 5 : Persistance basée sur des cookies

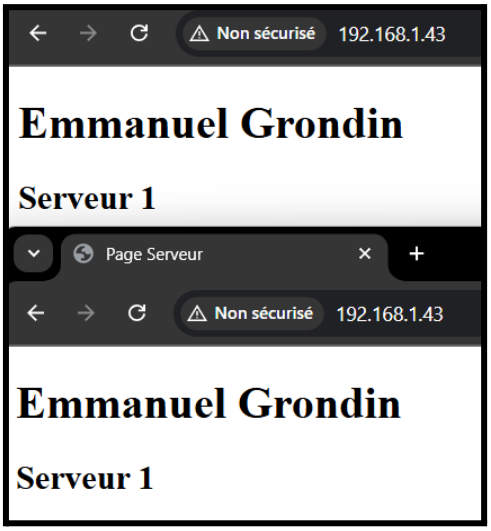
Configuration :
Dans la section frontend du fichier de configuration, ajouter la ligne suivante :

```
frontend http_front
    bind 192.168.1.43:80
    cookie SERVERUSED insert indirect nocache
    default_backend my_backend

backend my_backend
    balance roundrobin
    server apache1 192.168.1.208:80 cookie serveur1
    server apache2 192.168.1.230:80 cookie serveur2
```

Vérification :

Le client doit maintenant recevoir les réponses du même serveur, même si le second est opérationnel.
Tester la connexion d'un second client (un second navigateur).



Les deux connexion ramènes bien vers le serveur 1 chacun donc test validé

Étape 6 : Visualisation des statistiques

Avec HAProxy Stats, il est possible de visualiser des informations sur le nombre de connexions, le transfert de données, l'état du serveur, et plus encore. Comme il est basé sur un navigateur, il suffit d'utiliser un navigateur web pour obtenir des informations en temps réel sur l'implémentation HAProxy.

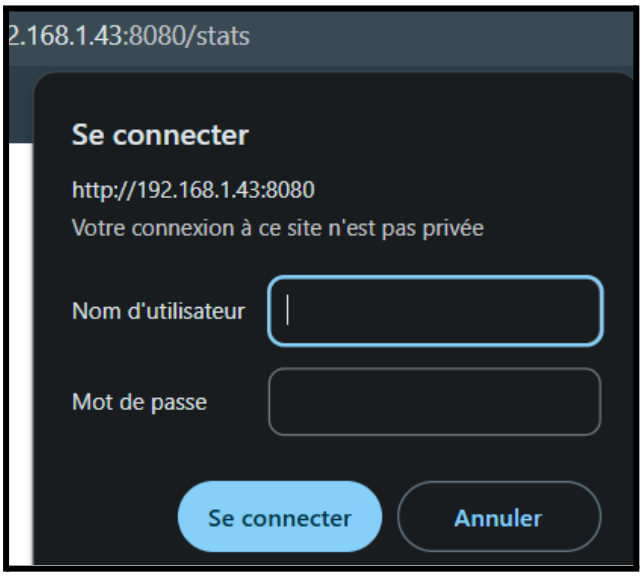
Configuration : ajouter et éditer une seconde section frontend dans le fichier haproxy.cfg :

```
frontend stats
    stats enable
    bind 192.168.1.43:8080  # HAProxy est en écoute sur le port 8080
pour afficher les stats
    log global
    stats auth admin:password  # Identifiants pour se connecter au
tableau de bord
    stats uri /stats  # L'URL pour accéder aux stats
```

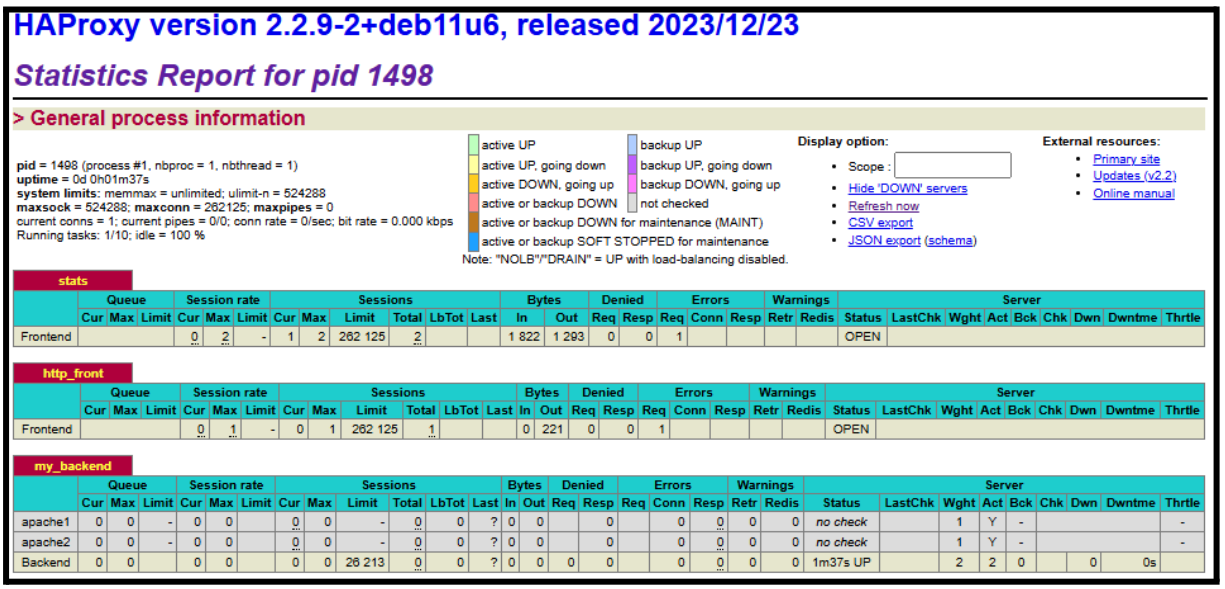
Vérifications : Exemple : si l'adresse IP du HAProxy est 192.168.1.26, si le port associé à la lecture des stats est 8080, l'URL à entrer dans le navigateur sera

```
http://192.168.1.26:8080/stats.
```

On nous demande l'accès de cette page par un login :



Et on accède au statistique du HAproxy :



HAProxy version 2.2.9-2+deb11u6, released 2023/12/23

Statistics Report for pid 1498

> General process information

pid = 1498 (process #1, nbproc = 1, nbthread = 1)
uptime = 0d 0h01m37s
system limits: memmax = unlimited; ulimit-n = 524288
maxsock = 524288; maxconn = 262125; maxpipes = 0
current conns = 1; current pipes = 0/0; conn rate = 0/sec; bit rate = 0.000 kbps
Running tasks: 1/10; idle = 100 %

active UP | backup UP
active UP, going down | backup UP, going down
active DOWN, going up | backup DOWN, going up
active or backup DOWN | not checked
active or backup DOWN for maintenance (MAINT)
active or backup SOFT STOPPED for maintenance
Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:
- Scope :
- Hide 'DOWN' servers
- Refresh now
- CSV export
- JSON export (schema)

External resources:
- Primary site
- Updates (v2.2)
- Online manual

stats

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	2	-	1	2	262 125	2			1 822	1 293	0	0	1					OPEN								

http_front

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	1	-	0	1	262 125	1			0	221	0	0	1					OPEN								

my_backend

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
apache1	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	*no check*		1	Y	-				-
apache2	0	0	-	0	0		0	0	-	0	0	?	0	0		0		0	0	0	0	*no check*		1	Y	-				-
Backend	0	0		0	0		0	0	26 213	0	0	?	0	0	0	0		0	0	0	0	1m37s UP		2	2	0		0	0s	

TÂCHE 5 – MISE EN PLACE DE LA HAUTE DISPONIBILITÉ DE HTTPS

Étape 1 : Créer un certificat et une clé privée

Pour générer un certificat SSL et une clé privée, on peut utiliser OpenSSL.

Générer la clé privée :

```
openssl genpkey -algorithm RSA -out /etc/ssl/private/server.key -aes256
```

1. Cette commande crée une clé privée RSA et la protège avec un mot de passe.

Générer le certificat auto-signé :

```
openssl req -new -x509 -key /etc/ssl/private/server.key -out
/etc/ssl/certs/server.crt -days 365
```

2. Cette commande génère un certificat auto-signé valide pour 365 jours.
   Ellenousdemandera des informations comme le nom de l'organisation, etc.
   Vérifie que les fichiers ont bien été créés :
   - /etc/ssl/private/server.key (clé privée)
   - /etc/ssl/certs/server.crt (certificat)

(2) Pour HAProxy, le certificat et la clé privée doivent être dans le même fichier, il faut donc fusionner ces deux
fichiers :

```
#echo /etc/ssl/certs/server.pem /etc/ssl/private/server.key >
/etc/ssl/certs/haproxy.pem
```

Étape 2 : Fusionner la clé privée et le certificat en un seul fichier PEM

Ensuite, on doit combiner la clé privée et le certificat dans un seul fichier .pem :

```
cat /etc/ssl/certs/server.crt /etc/ssl/private/server.key >
/etc/ssl/certs/haproxy.pem
```

Cela crée un fichier haproxy.pem dans le répertoire /etc/ssl/certs/ qui contient à la fois le certificat et la clé privée.

(3) Modifier la section frontend du fichier de configuration, en ajoutant la ligne suivante : bind <adresse_IP_HAProxy>:443 ssl cert /etc/ssl/certs/haproxy.pem

```
frontend https_front
   bind *:443 ssl cert /etc/ssl/certs/haproxy.pem
   mode https
   default_backend http_back
```

Cela configure HAProxy pour écouter sur le port 443 en SSL et utiliser le fichier PEM que l'on doit créer.

(1) Vérifier que le port 443 est maintenant en écoute sur le répartiteur.

```
tcp   LISTEN 0      4096                    0.0.0.0:443       0.0.0.0:*
```

(2) Vérifier que vous pouvez vous connecter sur les serveurs en HTTPS.

(3) Vérifier également que l'équilibrage de charge est fonctionnel.

```
root@debianVM:~# tail -f /var/log/haproxy.log
Mar  3 00:36:15 debianVM haproxy[1713]: Proxy servers started.
Mar  3 00:36:15 debianVM haproxy[1713]: Proxy my_backend started.
Mar  3 00:36:15 debianVM haproxy[1713]: Proxy my_backend started.
Mar  3 00:36:15 debianVM haproxy[1713]: [NOTICE] 061/003615 (1713) : New worker #1
(1715) forked
Mar  3 00:36:17 debianVM haproxy[1715]: [WARNING] 061/003617 (1715) : Server
servers/server2 is DOWN, reason: Layer4 connection problem, info: "No route to host",
check duration: 810ms. 1 active and 0 backup servers left. 0 sessions active, 0
requeued, 0 remaining in queue.
Mar  3 00:36:17 debianVM haproxy[1715]: Server servers/server2 is DOWN, reason: Layer4
connection problem, info: "No route to host", check duration: 810ms. 1 active and 0
```

```
backup servers left. 0 sessions active, 0 requeued, 0 remaining in queue.
Mar  3 00:36:17 debianVM haproxy[1715]: Server servers/server2 is DOWN, reason: Layer4
connection problem, info: "No route to host", check duration: 810ms. 1 active and 0
backup servers left. 0 sessions active, 0 requeued, 0 remaining in queue.
Mar  3 00:40:13 debianVM haproxy[1715]: [WARNING] 061/004013 (1715) : Server
servers/server2 is UP, reason: Layer4 check passed, check duration: 1ms. 2 active and 0
backup servers online. 0 sessions requeued, 0 total in queue.
Mar  3 00:40:13 debianVM haproxy[1715]: Server servers/server2 is UP, reason: Layer4
check passed, check duration: 1ms. 2 active and 0 backup servers online. 0 sessions
requeued, 0 total in queue.
Mar  3 00:40:13 debianVM haproxy[1715]: Server servers/server2 is UP, reason: Layer4
check passed, check duration: 1ms. 2 active and 0 backup servers online. 0 sessions
requeued, 0 total in queue.
```

L'équilibrage de charge semble **fonctionnel**, mais il y a eu une interruption sur `server2` :

1. **Serveur en panne** : À `00:36:17`, `server2` est passé **DOWN** (problème de connexion Layer 4 : "No route to host"). Cela signifie que HAProxy a détecté une **indisponibilité** du serveur cible.
2. **Serveur rétabli** : À `00:40:13`, `server2` est de nouveau **UP** après un **test Layer 4 réussi**, indiquant que la connexion est rétablie.
3. **Confirmation de l'équilibrage** : HAProxy détecte et ajuste dynamiquement la disponibilité des serveurs, garantissant ainsi une **répartition de charge adaptative**.

Conclusion : L'équilibrage fonctionne, mais une surveillance est nécessaire pour éviter des interruptions prolongées.

1. Redirection HTTP vers HTTPS

Pour rediriger les requêtes HTTP (port 80) vers HTTPS (port 443), on peut configurer un second frontend dans le fichier haproxy.cfg.

Ajouter une section frontend pour HTTP (port 80) :

Dans la section frontend qui gère les connexions HTTP (port 80), on va ajouter une règle de redirection vers HTTPS. Cela forcera tous les clients qui se connectent en HTTP à être redirigés automatiquement vers HTTPS.

Voici ce qu'on doit ajouter :

```
frontend http_front
```

```
    bind *:80
    option httplog
    redirect scheme https code 301 if { hdr(Host) -m found }
```

Explication :

- **bind *:80** : écoute sur le port HTTP (80).
- **redirect scheme https code 301** : redirige toutes les requêtes HTTP vers HTTPS (code HTTP 301, redirection permanente).
- **if { hdr(Host) -m found }** : cette condition s'applique pour toutes les requêtes avec un en-tête Host valide.

Cela permettra de rediriger toutes les requêtes HTTP vers HTTPS.

2. Imposer la version TLS 1.2 minimum

Ensuite, pour forcer l'utilisation de TLS 1.2 minimum sur le serveur HTTPS, on peut spécifier les versions de TLS que HAProxy doit accepter dans la configuration de la section frontend pour HTTPS.

Modifier la section frontend pour HTTPS (port 443) :

Dans la section où HAProxy écoute le port 443 pour HTTPS, on va ajouter une directive pour imposer TLS 1.2 ou une version plus récente.

Voici la modification à ajouter à frontend pour HTTPS :

```
frontend https_front
    bind *:443 ssl crt /etc/ssl/certs/haproxy.pem ciphers TLSv1.2
    option httplog
    default_backend servers
```

Explication :

- **bind *:443 ssl crt /etc/ssl/certs/haproxy.pem** : HAProxy écoute sur le port 443 avec SSL en utilisant le certificat haproxy.pem.
- **ciphers TLSv1.2** : définit que seulement la version TLS 1.2 et les suites de chiffrement compatibles sont autorisées.
- **option httplog** et **default_backend servers** : définissent l'enregistrement des logs et la connexion au backend de serveurs.

Avec cette configuration, HAProxy acceptera uniquement les connexions avec la version TLS 1.2 (et les versions plus récentes) pour sécuriser les communications.

Conclusion

La sécurisation des services réseaux est essentielle pour garantir la disponibilité, l'intégrité et la confidentialité des échanges. À travers ce projet, nous avons mis en place un serveur web sécurisé avec HTTPS, implémenté une haute disponibilité via HAProxy et assuré une répartition efficace des charges. L'utilisation de certificats TLS et de mécanismes de redondance permet d'améliorer la résilience du système face aux défaillances. Enfin, la redirection vers HTTPS et l'imposition de TLS 1.2 renforcent la sécurité des communications. Ces bonnes pratiques constituent une base solide pour toute infrastructure web fiable et sécurisée.