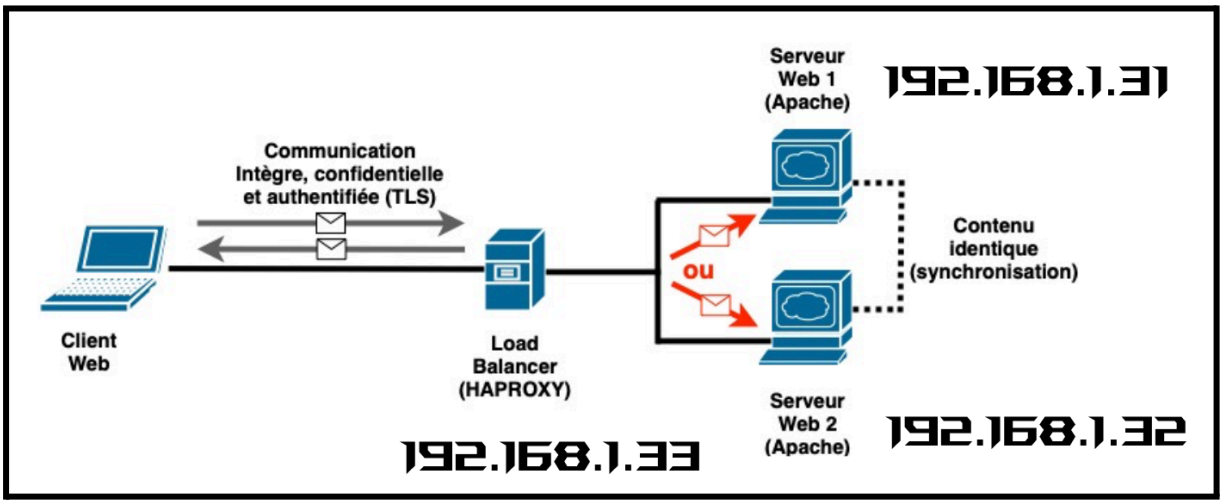
R4.Cyber.11 – TP 1 : Sécurisation d'un serveur web : TLS et haute disponibilité

Dans le cadre de cette séance de travaux pratiques, nous allons mettre en place un serveur web sécurisé en respectant les critères DICP (Disponibilité, Intégrité, Confidentialité et Authenticité). L'objectif principal est d'assurer une haute disponibilité du service en déployant un cluster de serveurs Apache, associé à un répartiteur de charge HAPROXY.

Pour cela, nous utiliserons trois machines virtuelles : deux pour les serveurs web Apache et une pour le répartiteur de charge. La machine hôte joue le rôle du client web. Nous configurons la répartition de charge pour garantir une distribution équilibrée des requêtes et assurer la continuité du service en cas de défaillance d'un serveur. Par ailleurs, nous mettrons en place une sécurisation des communications en activant le protocole TLS et en utilisant un certificat afin de garantir l'intégrité, la confidentialité et l'authenticité des échanges.

Ce compte rendu détaillera les différentes étapes de l'installation et de la configuration du serveur web sécurisé, en mettant en avant les bonnes pratiques pour assurer un service fiable et résilient.

Voici à quoi ressemblera la topologie finale :



SOMMAIRE

Tâche 1 : INSTALLATION DE APACHE (SUR LE PREMIER SERVEUR)

Étape 1 : Installation d'Apache sur le poste Serveur HTTP.

Étape 2 : Vérifications

Tâche 2 : MODIFICATION DE LA PAGE D'ACCUEIL DU SITE

Tâche 3 : SÉCURISATION D'UN SITE WEB : HTTPS

Étape 1 : Génération d'un certificat auto-signé

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

Étape 3 : Configuration du site sécurisé

Étape 4 : Tests

Tâche 4 : MISE EN PLACE DE LA HAUTE DISPONIBILITÉ DE HTTP

Étape 1 : Installation d'un second serveur Apache

Étape 2 : Installation de HAPROXY

Étape 3 : Configuration de HAPROXY

Étape 4 : Vérification du fonctionnement

Étape 5 : Persistance basée sur des cookies

Étape 6 : Visualisation des statistiques

Tâche 5 : MISE EN PLACE DE LA HAUTE DISPONIBILITÉ DE HTTPS

Tâche 1 : INSTALLATION DE APACHE (SUR LE PREMIER SERVEUR)

Étape 1 : Installation d'Apache sur le poste Serveur HTTP.

Nous commençons par installer le paquet Apache2 sur le premier serveur :

```
vboxuser@MasterDebian:~$ sudo apt install apache2
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Étape 2 : Vérifications

Nous affichons les ports actifs au moyen de la commande ss -ntl et nous observons ceci :

```
vboxuser@MasterDebian:~$ ss -ntl
State    Recv-Q    Send-Q        Local Address:Port        Peer Address:Port    Process
LISTEN   0         128               127.0.0.1:631              0.0.0.0:*
LISTEN   0         511                     *:80                     *:*
LISTEN   0         128                 [::1]:631                  [::]:*
```

Le port 80 est bel et bien en écoute, nous pouvons d'ailleur y accéder depuis la machine physique :

Tâche 2 : MODIFICATION DE LA PAGE D'ACCUEIL DU SITE

Nous lisons donc le contenu du fichier /etc/apache2/sites-available/000-default.conf :

Et nous observons que la racine du serveur est le dossier /var/www/html… Pour le moment, il ne contient que le fichier index.html par défaut :

Nous personnalisons le site avec ce nouveau code pour le fichier index :

```
<!DOCTYPE html>
<html lang="fr">
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Serveur Web 1 - BOUGHLEM Bilel</title>
        <style>
        body {
        font-family: Arial, sans-serif;
        background-color: #f4f4f4;
        text-align: center;
        padding: 50px;
        }
        .container {
        background-color: #fff;
        padding: 30px;
        border-radius: 8px;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        display: inline-block;
        }
        h1 {
        color: #333;
        }
        p {
        color: #666;
        }
```

```
        </style>
</head>
<body>
        <div class="container">
        <h1>Bienvenue sur le Serveur Web 1</h1>
        <h2>BOUGHLEM Bilel</h2>
        <p>Ce serveur fait partie d'un cluster avec répartition de charge via HAPROXY.</p>
        <p>Configuration sécurisée avec TLS pour garantir la confidentialité, l'intégrité et l'authenticité.</p>
        </div>
</body>
</html>
```

Et voici donc ce que ça donne :

Tâche 3 : SÉCURISATION D'UN SITE WEB : HTTPS

Étape 1 : Génération d'un certificat auto-signé

Pour générer un certificat auto-signé, nous allons utiliser l'outil openssl au moyen de cette commande exécutée dans le dossier /etc/apache2 *sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -out /etc/apache2/server.pem -keyout /etc/apache2/server.key* :

Nous pouvons donc visualiser le contenu des fichiers créés en commençant par les certificats :

```
vboxuser@MasterDebian:/etc/apache2$ cat server.pem
-----BEGIN CERTIFICATE-----
MIID3zCCAsegAwIBAgIUDN4Nr4vaz+dYnYt5l/UzK16D6UEwDQYJKoZIhvcNAQEL
BQAwfzELMAkGA1UEBhMCRlIxDzANBgNVBAgMBkZyYW5jZTEVMBMGA1UEBwwMU2Fp
bnQtUGllcnJlMSEwHwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQxJTAj
BgNVBAMMHHd3dy5zaXRlLWRlLWJpbGVsLXNlY3VyZS5jb20wHhcNMjUwMjI0MDUz
MTI2WhcNMjYwMjI0MDUzMTI2WjB/MQswCQYDVQQGEwJGUjEPMA0GA1UECAwGRnJh
bmNlMRUwEwYDVQQHDAxTYWludC1QaWVycmUxITAfBgNVBAoMGEludGVybmV0IFdp
ZGdpdHMgUHR5IEx0ZDElMCMGA1UEAwwcd3d3LnNpdGUtZGUtYmlsZWwtc2VjdXJl
LmNvbTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMT0HM0Oqybfz5WH
q/oGkKZ2S7UaeaGelxICA4OLZ/w1r+jpWlKsXOTebLyG4P/qChYOVqbaPL1ucDp0
7zE/NKh6NNAuxhlqx2BOD30ymiWaUqbMK2CtOT9tHFjM0iK6ygiTxUcjQPYUzGTx
1LlaB+kiQf1oGCc0kUWEJRUYHMWlFmWpYaMI9PiRD1UmqlHfyj6wOxac/elo89gz
F8VAcnnslqsYqG9uO6d5yBwjifr6F0tu+2iDaC4RoebhejTkxn6CUCGj1oINDtyp
bEDc8bhm0LmUWIyZeT5PcU+tN81h2cK1vvPQlqHNgrmlnW/JU0/YyVQVccaitKGn
xyGXqasCAwEAAaNTMFEwHQYDVR0OBBYEFNZOGCHJbH80lkuCEMcfwWCe9vWwMB8G
A1UdIwQYMBaAFNZOGCHJbH80lkuCEMcfwWCe9vWwMA8GA1UdEwEB/wQFMAMBAf8w
DQYJKoZIhvcNAQELBQADggEBAHYwOXVL6EPoeM0hqDFkpDKUoJEdYBDXjpfBrntr
exegrGr8/uxog5UqnVEAjEyaRkInvlgylltBF/ozn5WzhSe7wy2ehv45AZCbh8L8
mWERWMgfnCUpAwHCfnHCQjNilFWlGxXORA9uM5+LR1npkRw75OdfqCHSl4jvp3Ge
t5hrV/0eiDxiyBRItwO8+6sYIyqJauOi5JruO3qVmElCexutDLlolfwRk6hxnqDt
wLOYS4EgifmkZ2gNQob+eA5bycTcd7iqzkqzub+e+DJZgPzXgLaoizpC2oRU71f+
EFPsrHcrQwM2t4x0JYSVDU8mJ5I7iJrMlnAgxYJn0+fyCyU=
-----END CERTIFICATE-----
```

```
vboxuser@MasterDebian:/etc/apache2$ sudo cat server.key
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDE9BzNDqsm38+V
h6v6BpCmdku1GnmhnpcSAgODi2f8Na/o6VpSrFzk3my8huD/6goWDlam2jy9bnA6
dO8xPzSoejTQLsYZasdgTg99MpolmlKmzCtgrTk/bRxYzNIiusoIk8VHI0D2FMxk
8dS5WgfpIkH9aBgnNJFFhCUVGBzFpRZlqWGjCPT4kQ9VJqpR38o+sDsWnP3paPPY
MxfFQHJ57JarGKhvbjunecgcI4n6+hdLbvtog2guEaHm4Xo05MZ+glAho9aCDQ7c
qWxA3PG4ZtC51FiMmXk+T3FPrTfNYdnCtb7z0JahzYK5pZ1vyVNP2MlUFXHGorSh
p8ch16mrAgMBAAECggEALuJxrLrOMW3b65ula9p+OrI0aYNI6xlmmwnRrlHUSiy5
mHdcYgvX/T+ZoPN2wlUDW6bEJ+oRxYJxwuitPArwzR5yThCkZc6OcbWDaGlQWp03
Ca/Wr6fs1dVyhw0TBON0TsBmmP2ibDIdSfZrjkgJVA0sqldKUKLp5iOD9FaOGto7
0vfSN0W/MnwrKSSW2OSluAZWjA7XV6Ab9sMt0CjP9LCra0p3tCDop4QiRqs1zDZy
ffJis1BRhM59wSx6jzgDQ+ggXiTtCbjQARHRsvhSQj0vPGPN6R2m2jfgm4/yup+M
ECaBLsp8x8n1DNB+7twbTVm5Xe7j/DbJmbuKb0I5SQKBgQDpbuA8KoiNZuAyOrh4
zA7yOn0yK4m6D0CrXNdpKp5Wuzo/p4y0POCnB23g1J4+XJ9MrTRYemvt6ifoMXwh
+XABJMYLz36uJgULX9UkABefLEejFR9sg3Civ0NhWh1aeJafC4IoHE9O+YXEzGwx
B7Xk1Wo8tBZHZ6X2diSB7Xj/dwKBgQDX/mv9AQsN/Mmp7ynZae2uYA6hPQhpeIgW
OPsFuzpiR6M5qI88Xy0QsC4jLhlU4gF7JE+H/baI5nZgPBQsNhjrrwpakTvVChHd
F430yGiwURcyFHLqSAOStkS+GlegHZKGOCby3BrgiZFKZOul4dTgg8SrYadpEaNV
BxT/ayU8bQKBgQDPntBOcz41FlU+F8BPoJ+Cjscfwl2jYUGRqOfnIALo/WnXh/y7
qpHdvdvarhT5PoRpPHycXXZ1i5inCqxvw139hmmnw4vW9PXzoe7/MB3Scx3cTUM/
zsB14Evb4LzO5QaCu0euyU843PI/kdqA2WzM9YyiJULTi7qvNXFPJppCDQKBgQCG
Cc89/M29amhDWrXfxArJsbaifNuhiDj7gXTVw7ChmM9EUsRAtNekiZC4KD10HaTJ
pGjufZYvMsW3RtFF9nRUnhWQypwUwuD/EUT46P1+nXQFkgcWOIZvH0ldp79FD9UE
EgipocnUGcreHe7WzGRsx5jeugVIALgZZgGz1+wNqQKBgD414S4gGELlr9xtEikD
x0NPrReYpts4LTQXQAmgd1sU8Cbs4FfHPWrjRPJ8kP08S/I1+3eSZFchpkaXrNDn
FyahpWZ/q04aIbjID7B12xhyllnAHxb2/Y93VnkevpH+8eb4y7aG7AzLVcZgPwVV
DkqJ7K1ZPf1343xzPXjx1DOM
-----END PRIVATE KEY-----
```

Et voici l'affichage explicite du certificat :

```
vboxuser@MasterDebian:/etc/apache2$ openssl x509 -noout -text -in server.pem
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            0c:de:0d:af:8b:da:cf:e7:58:9d:8b:79:97:f5:33:2b:5e:83:e9:41
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = FR, ST = France, L = Saint-Pierre, O = Internet Widgits Pty Ltd, CN = www.site-de-bilel-secure.com
        Validity
            Not Before: Feb 24 05:31:26 2025 GMT
            Not After : Feb 24 05:31:26 2026 GMT
        Subject: C = FR, ST = France, L = Saint-Pierre, O = Internet Widgits Pty Ltd, CN = www.site-de-bilel-secure.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c4:f4:1c:cd:0e:ab:26:df:cf:95:87:ab:fa:06:
                    90:a6:76:4b:b5:1a:79:a1:9e:97:12:02:03:83:8b:
                    67:fc:35:af:e8:e9:5a:52:ac:5c:e4:de:6c:bc:86:
                    e0:ff:ea:0a:16:0e:56:a6:da:3c:bd:6e:70:3a:74:
                    ef:31:3f:34:a8:7a:34:d0:2e:c6:19:6a:c7:60:4e:
                    0f:7d:32:9a:25:9a:52:a6:cc:2b:60:ad:39:3f:6d:
                    1c:58:cc:d2:22:ba:ca:08:93:c5:47:23:40:f6:14:
                    cc:64:f1:d4:b9:5a:07:e9:22:41:fd:68:18:27:34:
                    91:45:84:25:15:18:1c:c5:a5:16:65:a9:61:a3:08:
                    f4:f8:91:0f:55:26:aa:51:df:ca:3e:b0:3b:16:9c:
                    fd:e9:68:f3:d8:33:17:c5:40:72:79:ec:96:ab:18:
                    a8:6f:6e:3b:a7:79:c8:1c:23:89:fa:fa:17:4b:6e:
                    fb:68:83:68:2e:11:a1:e6:e1:7a:34:e4:c6:7e:82:
                    50:21:a3:d6:82:0d:0e:dc:a9:6c:40:dc:f1:b8:66:
                    d0:b9:94:58:8c:99:79:3e:4f:71:4f:ad:37:cd:61:
                    d9:c2:b5:be:f3:d0:96:a1:cd:82:b9:a5:9d:6f:c9:
                    53:4f:d8:c9:54:15:71:c6:a2:b4:a1:a7:c7:21:97:
                    a9:ab
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                D6:4E:18:21:C9:6C:7F:34:96:4B:82:10:C7:1F:C1:60:9E:F6:F5:B0
            X509v3 Authority Key Identifier:
                D6:4E:18:21:C9:6C:7F:34:96:4B:82:10:C7:1F:C1:60:9E:F6:F5:B0
            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
```

```
Signature Value:
    76:30:39:75:4b:e8:43:e8:78:cd:21:a8:31:64:a4:32:94:a0:
    91:1d:60:10:d7:8e:97:c1:ae:7b:6b:7b:17:a0:ac:6a:fc:fe:
    ec:68:83:95:2a:9d:51:00:8c:4c:9a:46:42:27:be:58:32:96:
    5b:41:17:fa:33:9f:95:b3:85:27:bb:c3:2d:9e:86:fe:39:01:
    90:9b:87:c2:fc:99:61:11:58:c8:1f:9c:25:29:03:01:c2:7e:
    71:c2:42:33:62:94:55:a5:1b:15:ce:44:0f:6e:33:9f:8b:47:
    59:e9:91:1c:3b:e4:e7:5f:a8:21:d2:97:88:ef:a7:71:9e:b7:
    98:6b:57:fd:1e:88:3c:62:c8:14:48:b7:03:bc:fb:ab:18:23:
    2a:89:6a:e3:a2:e4:9a:ee:3b:7a:95:98:49:42:7b:1b:ad:0c:
    b9:68:95:fc:11:93:a8:71:9e:a0:ed:c0:b3:98:4b:81:20:89:
    f9:a4:67:68:0d:42:86:fe:78:0e:5b:c9:c4:dc:77:b8:aa:ce:
    4a:b3:b9:bf:9e:f8:32:59:80:fc:d7:80:b6:a8:8b:3a:42:da:
    84:54:ef:57:fe:10:53:ec:ac:77:2b:43:03:36:b7:8c:74:25:
    84:95:0d:4f:26:27:92:3b:88:9a:cc:96:70:20:c5:82:67:d3:
    e7:f2:0b:25
```

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

Pour se distinguer de notre site HTTP, nous créons un dossier html_ssl qui contiendra notre serveur HTTPS, avec un index.html adapté à la nouvelle version :

```
vboxuser@MasterDebian:/var/www$ sudo mkdir html_ssl
```

Puis nous activons le module SSL :

```
vboxuser@MasterDebian:~$ 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-sig
ned certificates.
To activate the new configuration, you need to run:
  systemctl restart apache2
```

Et le site sécurisé :

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

Comme précisé par la sortie de la commande, il faut redémarrer le service apache2 :

```
vboxuser@MasterDebian:~$ sudo systemctl reload apache2
```

Et nous avons deux fichiers créés :

```
vboxuser@MasterDebian:/etc/apache2/mods-available$ ls | grep "ssl"
ssl.conf
ssl.load
```

Étape 3 : Configuration du site sécurisé

Nous pouvons donc modifier le fichier configurant notre suite sécurisé :

```
 GNU nano 7.2                          default-ssl.conf
<VirtualHost *:443>
        ServerAdmin webmaster@localhost

        DocumentRoot /var/www/html_ssl
        ServerName www.site-de-bilel-secure.com

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined


        #   SSL Engine Switch:
        #   Enable/Disable SSL for this virtual host.
        SSLEngine on

        SSLCertificateFile      /etc/apache2/server.pem
        SSLCertificateKeyFile   /etc/apache2/server.key
```

Il n’est pas nécessaire de configurer le fichier ports.conf car il gère l'écoute globale des ports pour Apache. Étant donné que le port 443 est déjà pris en charge lors de l'activation du module SSL, il suffit de configurer le VirtualHost comme demandé.

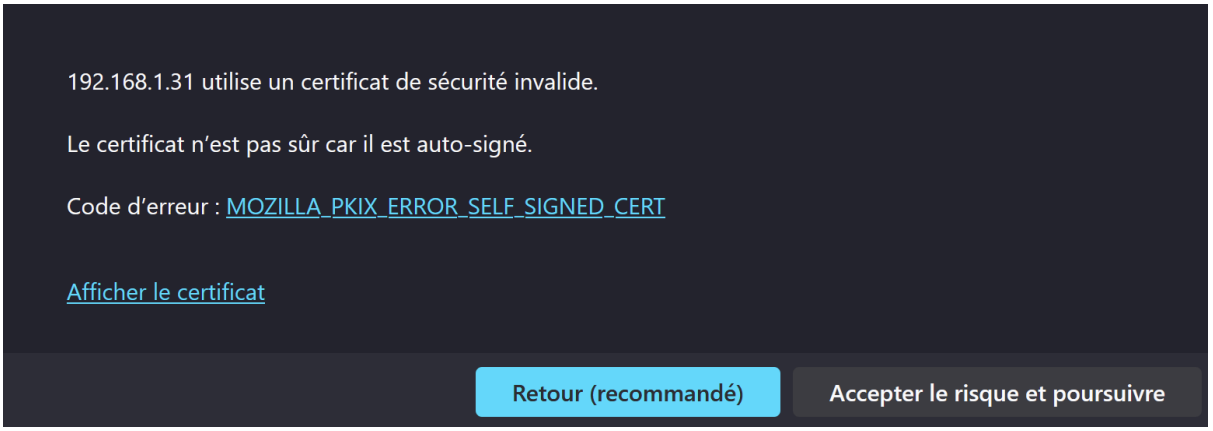
Étape 4 : Tests

Et nous pouvons donc tester le fonctionnement de nos configurations en listant les ports en écoute sur le serveur sachant que c'est le port 443 que nous avons mis en écoute :

```
vboxuser@MasterDebian:~$ ss -ntl
State    Recv-Q    Send-Q        Local Address:Port        Peer Address:Port    Process
LISTEN   0         128               127.0.0.1:631              0.0.0.0:*
LISTEN   0         128                 0.0.0.0:22               0.0.0.0:*
LISTEN   0         128                    [::]:22                  [::]:*
LISTEN   0         511                       *:80                     *:*
LISTEN   0         128                  [::1]:631                  [::]:*
LISTEN   0         511                       *:443                    *:*
```

Et nous pouvons accéder au site sécurisé :

Le site n'est pas considéré comme sûr puisque le certificat est auto signé, il faudrait qu'une entité spécialisée certifie officiellement le site pour que le navigateur n'affiche plus cet avertissement :



Nous pouvons afficher des informations sur le certificat depuis le navigateur :

Informations sur la clé publique

Algorithme	RSA
Taille de la clé	2048
Exposant	65537
Module	CE:B5:8F:B4:D8:BD:1C:40:3F:74:EA:F6:AC:F1:62:43:5F:47:AD:44:07:C4:1B:3A:2F:17:D9:82:E4:42:1D:65:B6:2C:6D:64:AC:51:AD:FD:E7:10:6D:48:B0:29:A0:FC:3D:A5:5F:8E:ED:BC:05:98:65:2B:2B:30:41:39:8B:E0:62:15:89:C0:07:EF:6F:B7:ED:48:8D:BF:5F:C1:72:0B:7A:12:3D:D0:7D:BE:FF:5F:1F:D1:60:BE:A0:92:83:D1:33:A4:C9:CD:B9:B8:D6:AE:70:14:C3:86:03:7C:FB:10:AE:E8:89:27:72:D3:C4:58:DA:68:1D:77:37:68:91:57:D6:C7:C7:80:74:1B:23:02:12:55:9A:4D:7A:7B:E7:0F:07:B7:8B:13:45:FD:A4:05:CF:94:9A:11:99:08:7C:D9:1A:29:0B:54:F9:43:AC:DA:BB:57:C0:42:0E:F0:58:57:50:23:BC:ED:01:35:3E:2B:F8:DD:F7:3E:FB:2D:29:B9:9A:64:5E:DE:18:CA:BA:11:EE:60:8C:3C:8E:9A:54:62:41:BD:D0:2F:7E:D1:30:FC:38:F9:90:BF:F5:4A:9C:BF:8D:09:6C:B4:AB:37:79:DF:4B:23:67:98:D1:A8:8B:BB:5B:B4:01:70:71:75:F4:EC:80:BB:D5:D0:98:25:DE:9F

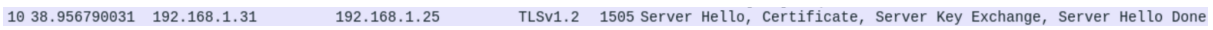
Divers

Numéro de série	17:B5:11:BC:04:06:F7:DA:DC:B4:5C:E3:9A:FB:81:4D:28:C7:65:FB
Algorithme de signature	SHA-256 with RSA Encryption
Version	3
Télécharger	PEM (cert) PEM (chain)

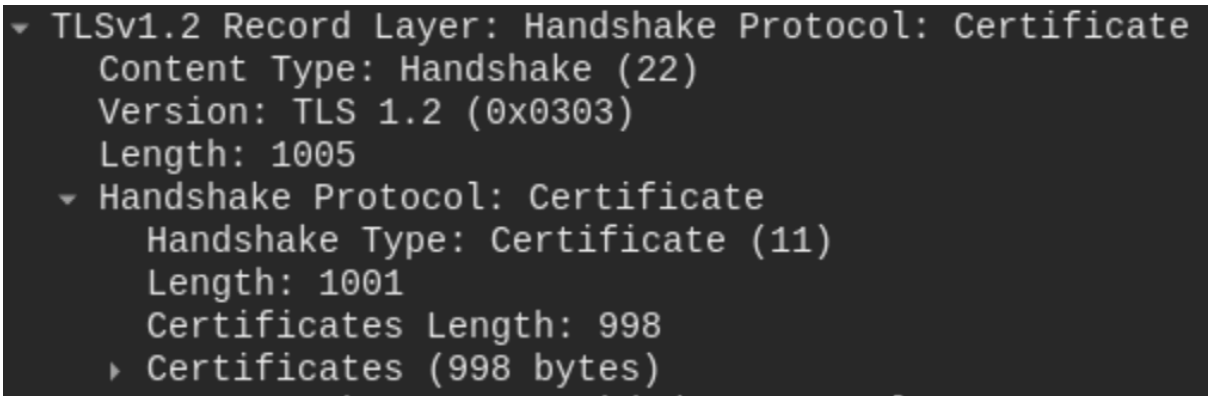
Empreintes numériques

SHA-256	AA:55:34:DA:85:15:AE:F9:66:8E:BA:EE:B6:44:23:78:87:62:CD:A2:89:4C:83:08:E2:62:3C:79:28:4B:6C:97
SHA-1	C9:D0:F4:D6:C6:D0:FC:F4:49:4D:FB:C2:C7:5E:0C:BE:48:DA:C6:84

Si nous voulons le faire en utilisant Wireshark, nous devons trouver la trame Server Hello qui contient le certificat recherché :



Et nous pouvons donc analyser cette trame et finissons par trouver le certificat :



Et nous reconnaissons les informations que nous avions complété dans le certificat :

Et nous avons donc pu analyser le certificat transmis par notre page web sécurisée avec Wireshark, à noter que cela n'aurait pas été possible avec TLSv1.3 car il chiffre la plus grande partie du handshake…

Tâche 4 : MISE EN PLACE DE LA HAUTE DISPONIBILITÉ DE HTTP

Étape 1 : Installation d'un second serveur Apache

Nous clonons donc le premier serveur en un second, nous décidons de modifier les hostnames des deux machines afin de les rendre plus faciles à reconnaître :

```
  GNU nano 7.2                                  /etc/hostname
Serveur1
```

Et voici les deux sites :

Et en adaptant les différents fichiers index.html (non sécurisé et sécurisé) au nouveau serveur, nous sommes prêts à configurer le serveur HAPROXY…

Étape 2 : Installation de HAPROXY

Nous avons donc une autre machine HAPROXY sur laquelle nous installons le paquet dont elle porte le nom :

```
vboxuser@HAPROXY:~$ sudo apt install haproxy
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Et le paquet est installé.

Étape 3 : Configuration de HAPROXY

Nous pouvons donc modifier le fichier /etc/haproxy/haproxy.cfg :

```
global
        maxconn 1000
        log /dev/log local0
        log /dev/log local1 notice
        chroot /var/lib/haproxy
        stats socket /run/haproxy/admin.sock mode 660 level admin
        stats timeout 30s
        user haproxy
        group haproxy
        daemon
        ca-base /etc/ssl/certs
        crt-base /etc/ssl/private
        ssl-default-bind-ciphers
ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!
MD5:!DSS
        ssl-default-bind-options no-sslv3
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 bilelfrontend
        bind 192.168.1.33:80
        default_backend bilelbackend
backend bilelbackend
```

```
mode http
balance roundrobin
server Serveur1 192.168.1.31:80 check
server Serveur2 192.168.1.32:80 check
```

Expliquons donc quelques paramètres :

- Le paramètre maxconn définit le nombre maximal de connexions simultanées que HAPROXY peut gérer pour chaque processus. Une valeur trop basse peut limiter le nombre d'utilisateurs pouvant accéder au service, tandis qu'une valeur trop haute peut consommer trop de ressources système. Dans notre cas, *maxconn 1000* limite le nombre de connexions simultanées à 1000.

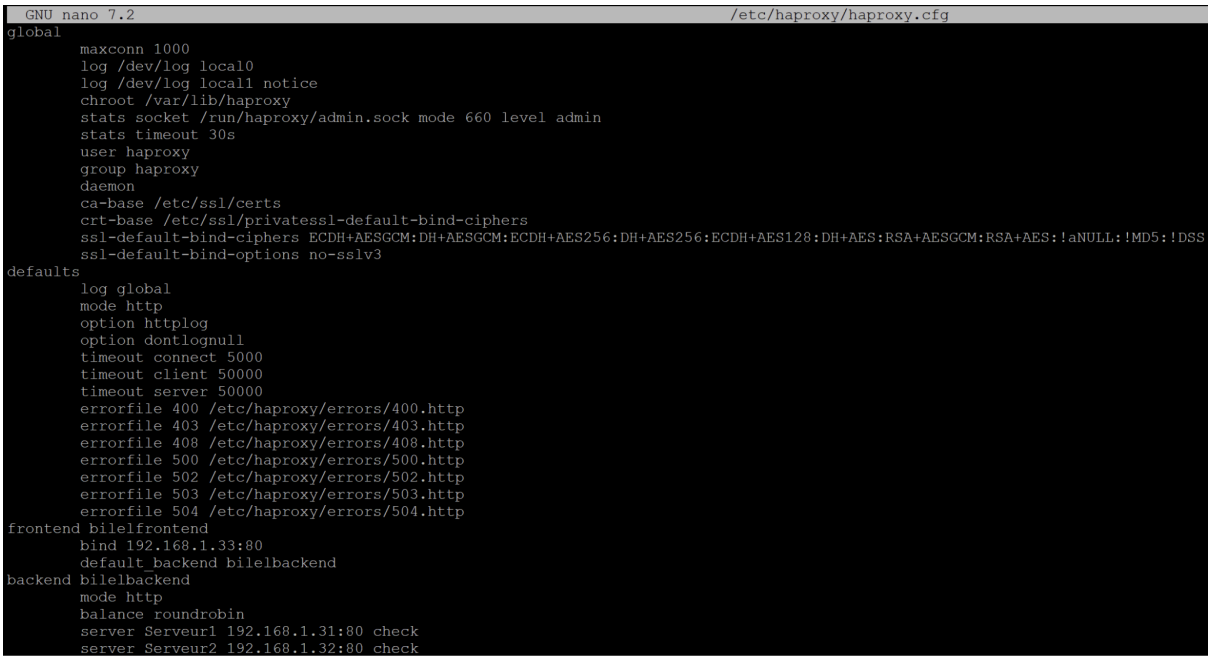
- Les timeout définissent des délais pour différentes interactions réseau, évitant les blocages indéfinis :

    - Timeout connect : Temps maximum pour établir une connexion entre HAPROXY et un serveur backend. Dans notre cas, *timeout connect 5000* signifie 5 secondes d'attente avant abandon.
    - Timeout client : Temps maximum pendant lequel HAPROXY attend qu'un client (navigateur, application) envoie une requête complète. Dans notre cas, *timeout client 50000* signifie 50 secondes d'attente avant abandon.
    - Timeout server : Temps maximum pendant lequel HAPROXY attend une réponse complète d'un serveur backend avant d'abandonner. Dans notre cas, *timeout server 50000* signifie 50 secondes d'attente avant abandon.

- Le paramètre balance roundrobin est utilisé pour répartir équitablement les requêtes entrantes entre plusieurs serveurs backend. Chaque serveur est sélectionné à tour de rôle, assurant une distribution équilibrée de la charge. Dans notre cas, la ligne *balance roundrobin* suivie des lignes *server Serveur1 192.168.1.31:80* et *server Serveur2 192.168.1.32:80* signifie que la première requête ira à Serveur1, la deuxième à Serveur2...

- Le mot-clé check permet à HAPROXY de vérifier l'état des serveurs backend en envoyant régulièrement des requêtes. Si un serveur ne répond pas, il est marqué comme indisponible et HAPROXY ne lui enverra plus de trafic jusqu'à ce qu'il soit de nouveau fonctionnel. Dans notre cas, les lignes *server Serveur1 192.168.1.31:80 check* et *server Serveur2 192.168.1.32:80* check signifient que si Serveur1 tombe en panne, HAPROXY redirigera tout le trafic vers Serveur2.

Etant donné la longueur de certaines lignes (qui seront affichées différemment sur ce compte rendu que dans le fichier de configuration) qui pourraient causer certaines confusions, voici à quoi ressemble le contenu du fichier haproxy.cfg sur l'éditeur vim :



```
 GNU nano 7.2                                                    /etc/haproxy/haproxy.cfg
global
        maxconn 1000
        log /dev/log local0
        log /dev/log local1 notice
        chroot /var/lib/haproxy
        stats socket /run/haproxy/admin.sock mode 660 level admin
        stats timeout 30s
        user haproxy
        group haproxy
        daemon
        ca-base /etc/ssl/certs
        crt-base /etc/ssl/privatessl-default-bind-ciphers
        ssl-default-bind-ciphers ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!MD5:!DSS
        ssl-default-bind-options no-sslv3
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 bilelfrontend
        bind 192.168.1.33:80
        default_backend bilelbackend
backend bilelbackend
        mode http
        balance roundrobin
        server Serveur1 192.168.1.31:80 check
        server Serveur2 192.168.1.32:80 check
```

Vérifions donc si ces configurations sont correctes…

Étape 4 : Vérification du fonctionnement

Et nous pouvons donc redémarrer le service pour voir si tout fonctionne :

```
vboxuser@HAPROXY:~$ sudo systemctl restart haproxy
```

Aucun erreur, vérifions donc que les serveurs web sont accessibles via l'adresse IP du répartiteur de charge HAPROXY dans le navigateur :

Sans trop de surprises, cela fonctionne, et lorsque nous rafraichissons la page, nous changeons automatiquement de serveur :

Ce changement est relatif à la configuration roundrobin expliquée plus haut, qui fait que nous changeons de site pour chaque connexion, de manière à équilibrer la charge des serveurs.

De plus, si nous stoppons le Serveur1 au moyen de la commande sudo systemctl stop apache2 :

```
vboxuser@Serveur1:~$ sudo systemctl stop apache2
```

Nous pouvons donc tester si le seul serveur disponible est le Serveur2 et c'est le cas.

Nous lançons le Serveur1 au moyen de la commande sudo systemctl start apache2 et la page alterne de serveur à chaque rafraichissement :

```
vboxuser@Serveur1:~$ sudo systemctl start apache2
```

Étape 5 : Persistance basée sur des cookies

Et si nous ne voulons pas changer de serveur à chaque rafraichissement de la page, à moins que cela ne soit nécessaire (car un serveur est down) ?

Nous pouvons modifier la section backend du fichier de configuration de HAPROXY comme ceci :

```
backend bilelbackend
        mode http
        balance roundrobin
        cookie SERVEURUTILISE insert indirect nocache
        server Serveur1 192.168.1.31:80 cookie S1 check
        server Serveur2 192.168.1.32:80 cookie S2 check
```

Nous pouvons ensuite redémarrer le service HAPROXY et les modifications sont effectives.

Et à présent, il ne suffit plus de rafraîchir la page pour changer de serveur, il faut également supprimer les cookies associés au site en plus de s'assurer d'avoir changé de parité d'utilisateur (passer de 4ème à 12ème utilisateur n'aura aucun effet)... Nous pouvons afficher les cookies de session associés au site en inspectant la page :

Nom	Valeur	Domain	Path	Expiration / Durée maximum
SERVEURUTILISE	S1	192.168.1.33	/	Session

Étape 6 : Visualisation des statistiques

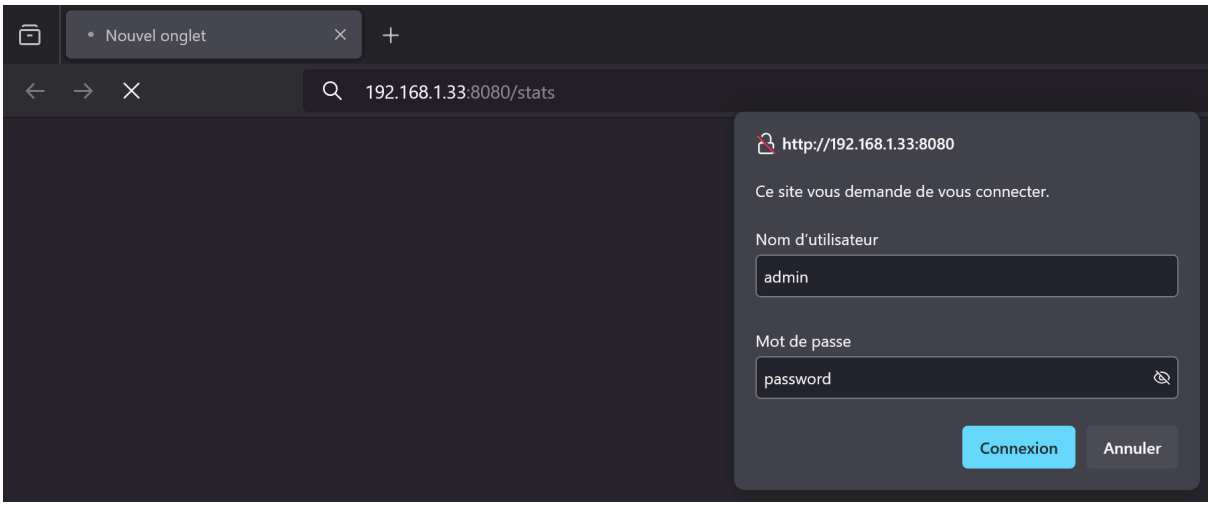
Avec HAPROXY Stats, il est possible de visualiser des informations sur le nombre de connexions, le transfert de données, l'état du serveur, et plus encore. Comme il est basé sur un navigateur, il suffit d'utiliser un navigateur web pour obtenir des informations en temps réel sur l'implémentation HAPROXY.

Nous allons donc créer une seconde section frontend dans le fichier de configuration haproxy.cfg qui permettra d'avoir cette page de visualisation d'informations :

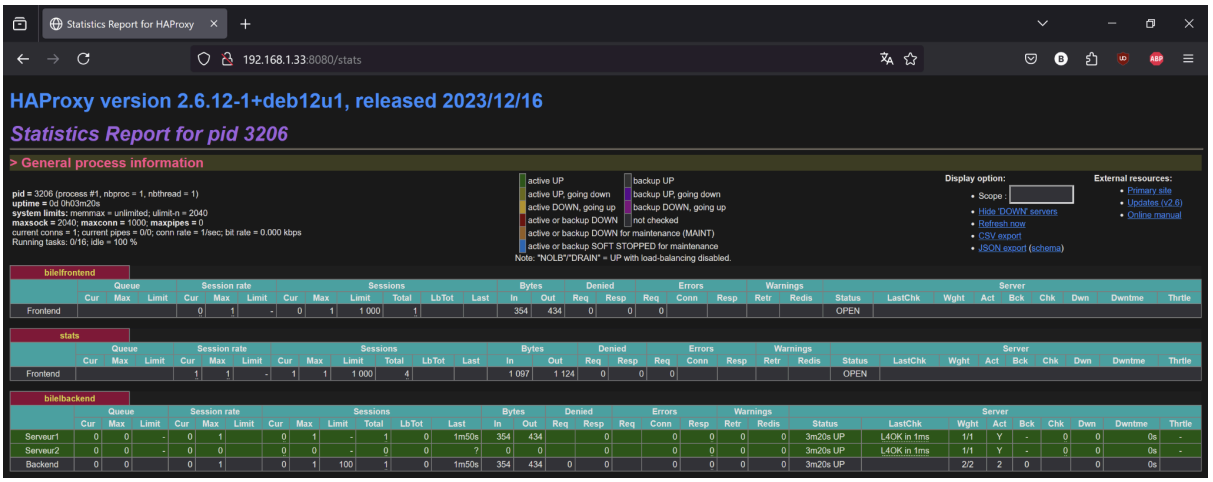
```
frontend stats
        stats enable
        bind 192.168.1.33:8080
        log global
        stats auth admin:password
        stats uri /stats
```

A noter que ce serveur n'est qu'un serveur de test, une première mesure de sécurité aurait dû être le choix d'une combinaison login/password plus sécurisée mais pour l'instant, nous voulons faire dans la simplicité en nous assurant simplement que le service est assuré conformément aux attentes du TP…

Nous pouvons donc redémarrer le service et accéder à la page d'authentification configurée pour pouvoir accéder aux statistiques :



Puis nous pouvons accéder à cette page récemment créée :



Et nous avons donc les statistiques du serveur HAPROXY affichées tout en couleur…

Tâche 5 : MISE EN PLACE DE LA HAUTE DISPONIBILITÉ DE HTTPS

Depuis le début, les pages étaient les versions non sécurisées de nos deux serveurs web, nous allons donc offrir les mêmes services et options mais avec nos versions sécurisées.

Pour cela, nous allons commencer par créer un clé et un certificat sur notre serveur HAPROXY au moyen de la commande *sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -out /etc/ssl/certs/server.pem -keyout /etc/ssl/private/server.key* :

```
vboxuser@HAPROXY:~$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -out /etc/ssl/certs/server.pem -keyout /etc/ssl/private/server.key
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:www.site-de-bilel-secure.com
Email Address []:
```

Puis sachant qu’il faut que les deux fichiers créés soient dans un seul et même fichier pour HAPROXY, nous allons devoir les fusionner en un seul fichier avec la commande *cat /etc/ssl/certs/server.pem /etc/ssl/private/server.key > /etc/ssl/certs/haproxy.pem* que nous avons donc nommé haproxy.pem :

```
root@HAPROXY:/home/vboxuser# echo /etc/ssl/certs/server.pem /etc/ssl/private/server.key > /etc/ssl/certs/haproxy.pem
```

Et nous pouvons donc lire le fichier haproxy.pem :

```
root@HAPROXY:/home/vboxuser# cat /etc/ssl/certs/haproxy.pem
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
-----BEGIN PRIVATE KEY-----
-----END PRIVATE KEY-----
```

Il contient donc le certificat suivi de la clé. Il faut à présent modifier la section bilelfrontend du fichier de configuration haproxy.cfg en ajoutant le certificat et la clé fusionnés :

```
frontend bilelfrontend
        bind 192.168.1.33:443 ssl crt /etc/ssl/certs/haproxy.pem
        default_backend bilelbackend
```

Et nous pouvons donc redémarrer le service et accéder à la version sécurisée des sites :

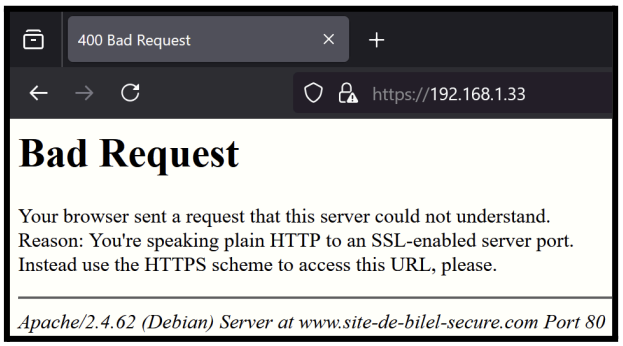
Cependant, le site HAPROXY est certe sécurisé mais pas le site web Serveur 1 dans notre cas, c'est parce que le backend utilisé la version non sécurisée du site, ce que nous allons faire est créer un backend et un frontend pour chacune des versions de ce site, avec donc une version normale et l'autre complètement sécurisée, nous allons également séparer les cookies entre les version sécurisées ou pas des sites optimiser la répartition de la charge sur les serveurs :

```
global
        maxconn 1000
        log /dev/log local0
        log /dev/log local1 notice
        chroot /var/lib/haproxy
        stats socket /run/haproxy/admin.sock mode 660 level admin
        stats timeout 30s
        user haproxy
        group haproxy
        daemon
        ca-base /etc/ssl/certs
        crt-base /etc/ssl/privatessl-default-bind-ciphers
        ssl-default-bind-ciphers
ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!
MD5:!DSS
        ssl-default-bind-options no-sslv3
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 bilel_frontend
        bind 192.168.1.33:80
        default_backend bilel_backend
frontend bilel_secure_frontend
        bind 192.168.1.33:443 ssl crt /etc/ssl/certs/haproxy.pem
        default_backend bilel_secure_backend
frontend stats
        stats enable
        bind 192.168.1.33:8080
        log global
        stats auth admin:password
        stats uri /stats
backend bilel_backend
        mode http
        balance roundrobin
        cookie SERVEURUTILISE insert indirect nocache
        server Serveur1 192.168.1.31:80 cookie S1 check
        server Serveur2 192.168.1.32:80 cookie S2 check
backend bilel_secure_backend
        mode http
        balance roundrobin
        cookie SERVEURSECURISEUTILISE insert indirect nocache
        server Serveur1 192.168.1.31:443 cookie SS1 check
        server Serveur2 192.168.1.32:443 cookie SS2 check
```

C’est beaucoup plus propre comme ceci, avec une version complètement HTTP et l’autre complètement HTTPS :

Ou pas :



Il semble que le serveur envoie des requêtes sécurisées à 192.168.1.31:443 et 192.168.1.32:443 au lieu de https://192.168.1.31:443 et https://192.168.1.32:443…

Nous allons donc modifier la section bilel_secure_backend comme ceci :

```
backend bilel_secure_backend
        mode http
        balance roundrobin
        cookie SERVEURSECURISEUTILISE insert indirect nocache
        server Serveur1 192.168.1.31:443 ssl cookie SS1 check
        server Serveur2 192.168.1.32:443 ssl cookie SS2 check
```

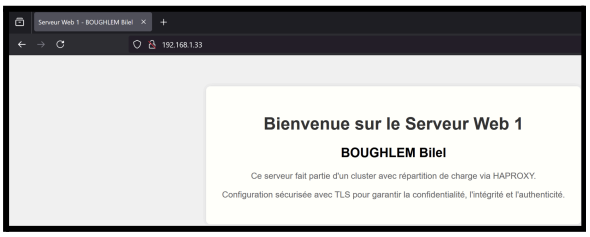
Nous n'arrivons cependant pas à redémarrer le service pour cette raison :

```
vboxuser@HAPROXY:/etc/ssl/certs$ sudo haproxy -c -f /etc/haproxy/haproxy.cfg
[NOTICE]   (3564) : haproxy version is 2.6.12-1+deb12u1
[NOTICE]   (3564) : path to executable is /usr/sbin/haproxy
[ALERT]    (3564) : config : [/etc/haproxy/haproxy.cfg:52] : 'server bilel_secure_backend/Serveur1' : verify is enabled by default but no CA file specified. If you're running on
 a LAN where you're certain to trust the server's certificate, please set an explicit 'verify none' statement on the 'server' line, or use 'ssl-server-verify none' in the global
 section to disable server-side verifications by default.
[ALERT]    (3564) : config : [/etc/haproxy/haproxy.cfg:53] : 'server bilel_secure_backend/Serveur2' : verify is enabled by default but no CA file specified. If you're running on
 a LAN where you're certain to trust the server's certificate, please set an explicit 'verify none' statement on the 'server' line, or use 'ssl-server-verify none' in the global
 section to disable server-side verifications by default.
[ALERT]    (3564) : config : Fatal errors found in configuration.
```

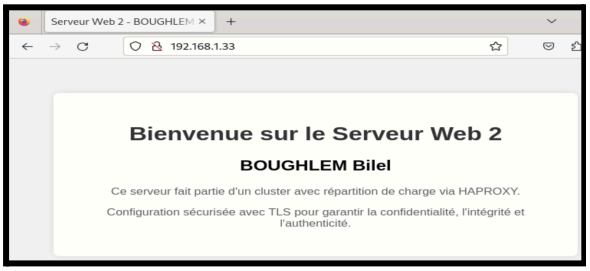
Comme indiqué par le retour de la commande, nous décidons de faire confiance aux certificats de nos serveurs web et ajoutons donc verify none dans les requêtes comme ceci :

```
backend bilel_secure_backend
        mode http
        balance roundrobin
        cookie SERVEURSECURISEUTILISE insert indirect nocache
        server Serveur1 192.168.1.31:443 ssl verify none cookie SS1 check
        server Serveur2 192.168.1.32:443 ssl verify none cookie SS2 check
```

Et tout fonctionne :



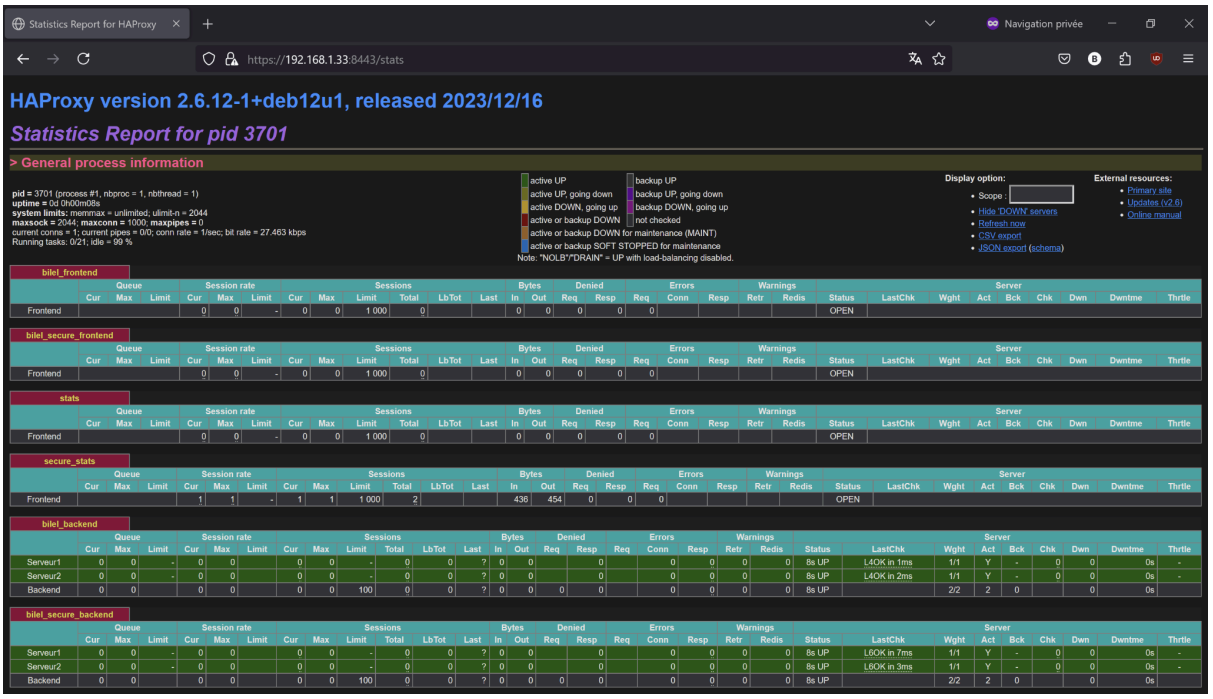
Et la répartition des charges à bel et bien lieu avec notre second client :



Une toute dernière modification que nous pourrions faire serait de créer également une version sécurisée du site d'affichage des statistiques, nous pouvons donc créer cette section dans le fichier de configuration (à noter que les password des deux sites d'affichage des statistiques doivent être les mêmes sinon c'est celui de la première section qui sera choisi) :

```
frontend secure_stats
        stats enable
        bind 192.168.1.33:8443 ssl crt /etc/ssl/certs/haproxy.pem
        log global
        stats auth admin:password
        stats uri /stats
```

Comme précédemment, nous accédons à la page d'authentification puis voici le résultat :



A noter que si nous ne voulons garder que la version sécurisé de tous les services à des fins de sécurité et de simplicité, voici comment nous pourrions configurer notre serveur :

```
global
        maxconn 1000
        log /dev/log local0
        log /dev/log local1 notice
        chroot /var/lib/haproxy
        stats socket /run/haproxy/admin.sock mode 660 level admin
        stats timeout 30s
        user haproxy
        group haproxy
        daemon
        ca-base /etc/ssl/certs
        crt-base /etc/ssl/privatessl-default-bind-ciphers
        ssl-default-bind-ciphers
ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!
MD5:!DSS
        ssl-default-bind-options no-sslv3
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 bilel_secure_frontend
        bind 192.168.1.33:443 ssl crt /etc/ssl/certs/haproxy.pem
        default_backend bilel_secure_backend
frontend secure_stats
        stats enable
        bind 192.168.1.33:8443 ssl crt /etc/ssl/certs/haproxy.pem
        log global
        stats auth admin_drihJR:5aj0TufYOJlyvbMn
        stats uri /stats
backend bilel_secure_backend
        mode http
        balance roundrobin
        cookie SERVEURSECURISEUTILISE insert indirect nocache
        server Serveur1 192.168.1.31:443 ssl verify none cookie SS1 check
        server Serveur2 192.168.1.32:443 ssl verify none cookie SS2 check
```

Et bien sûr, nous avons imposé une combinaison login/password plus sécurisé que le simple admin/password…

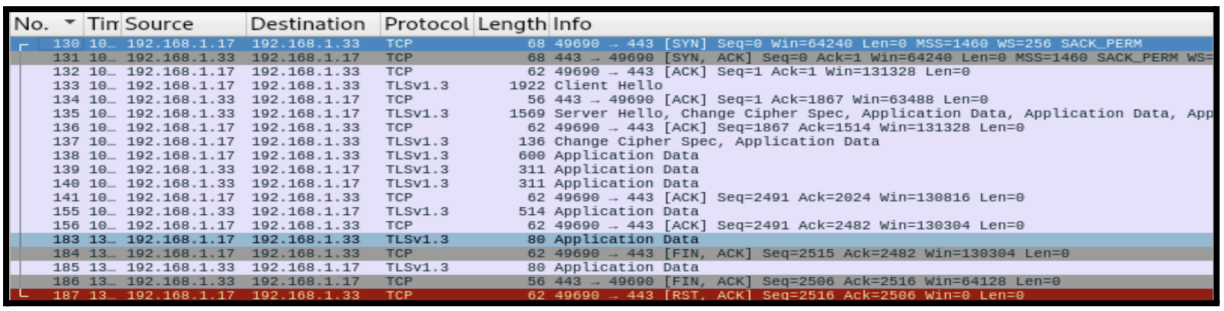
Et voici le code complet si nous voulons proposer les versions sécurisées et non-sécurisées en choisissant celle qui est sécurisée par défaut si rien n'est précisé, tout en imposant la version 1.2 de TLS au minimum :

```
global
        maxconn 1000
        log /dev/log local0
        log /dev/log local1 notice
        chroot /var/lib/haproxy
        stats socket /run/haproxy/admin.sock mode 660 level admin
        stats timeout 30s
        user haproxy
        group haproxy
        daemon
        ca-base /etc/ssl/certs
        crt-base /etc/ssl/private
        ssl-default-bind-ciphers
ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:RSA+AESGCM:RSA+AES:!aNULL:!
MD5:!DSS
        ssl-default-bind-options no-sslv3
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 bilel_frontend
        bind 192.168.1.33:80
        redirect scheme https code 301 if !{ ssl_fc }
        default_backend bilel_backend
frontend bilel_secure_frontend
        bind 192.168.1.33:443 ssl crt /etc/ssl/certs/haproxy.pem ssl-min-ver TLSv1.2
        default_backend bilel_secure_backend
frontend stats
        stats enable
        bind 192.168.1.33:8080
        log global
        stats auth admin:password
        stats uri /stats
frontend secure_stats
        stats enable
        bind 192.168.1.33:8443 ssl crt /etc/ssl/certs/haproxy.pem ssl-min-ver TLSv1.2
        log global
        stats auth admin:password
        stats uri /stats
backend bilel_backend
        mode http
        balance roundrobin
        cookie SERVEURUTILISE insert indirect nocache
        server Serveur1 192.168.1.31:80 cookie S1 check
        server Serveur2 192.168.1.32:80 cookie S2 check
backend bilel_secure_backend
        mode http
        balance roundrobin
        cookie SERVEURSECURISEUTILISE insert indirect nocache
        server Serveur1 192.168.1.31:443 ssl verify none cookie SS1 check
        server Serveur2 192.168.1.32:443 ssl verify none cookie SS2 check
```

Nous imposant la version 1.2 de TLS au minimum avec la directive *ssl-min-ver TLSv1.2*, et nous priorisons l'usage de HTTPS au lieu de HTTP si aucun n'est précisé avec cette directive *redirect scheme https code 301 if !{ ssl_fc }*, à noter que nous ne l'appliquons pas à la page de statistiques ce qui est un choix car nous estimons que celui utilisant cette page possède les compétences pour ne pas avoir besoin d'être redirigé vers une version plus qu'une autre.

Et voici les trames capturées :



No.	Tim	Source	Destination	Protocol	Length	Info
130	10…	192.168.1.17	192.168.1.33	TCP	68	49690 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
131	10…	192.168.1.33	192.168.1.17	TCP	68	443 → 49690 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=
132	10…	192.168.1.17	192.168.1.33	TCP	62	49690 → 443 [ACK] Seq=1 Ack=1 Win=131328 Len=0
133	10…	192.168.1.17	192.168.1.33	TLSv1.3	1922	Client Hello
134	10…	192.168.1.33	192.168.1.17	TCP	56	443 → 49690 [ACK] Seq=1 Ack=1867 Win=63488 Len=0
135	10…	192.168.1.33	192.168.1.17	TLSv1.3	1569	Server Hello, Change Cipher Spec, Application Data, Application Data, App
136	10…	192.168.1.17	192.168.1.33	TCP	62	49690 → 443 [ACK] Seq=1867 Ack=1514 Win=131328 Len=0
137	10…	192.168.1.17	192.168.1.33	TLSv1.3	136	Change Cipher Spec, Application Data
138	10…	192.168.1.17	192.168.1.33	TLSv1.3	600	Application Data
139	10…	192.168.1.33	192.168.1.17	TLSv1.3	311	Application Data
140	10…	192.168.1.33	192.168.1.17	TLSv1.3	311	Application Data
141	10…	192.168.1.17	192.168.1.33	TCP	62	49690 → 443 [ACK] Seq=2491 Ack=2024 Win=130816 Len=0
155	10…	192.168.1.33	192.168.1.17	TLSv1.3	514	Application Data
156	10…	192.168.1.17	192.168.1.33	TCP	62	49690 → 443 [ACK] Seq=2491 Ack=2482 Win=130304 Len=0
183	13…	192.168.1.17	192.168.1.33	TLSv1.3	80	Application Data
184	13…	192.168.1.17	192.168.1.33	TCP	62	49690 → 443 [FIN, ACK] Seq=2515 Ack=2482 Win=130304 Len=0
185	13…	192.168.1.33	192.168.1.17	TLSv1.3	80	Application Data
186	13…	192.168.1.33	192.168.1.17	TCP	56	443 → 49690 [FIN, ACK] Seq=2506 Ack=2516 Win=64128 Len=0
187	13…	192.168.1.17	192.168.1.33	TCP	62	49690 → 443 [RST, ACK] Seq=2516 Ack=2506 Win=0 Len=0

C’est la version 1.2 du protocole TLS qui est utilisée :

```
TLSv1.3 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
```

Le client échange sa clée :

```
▼ Extension: key_share (len=1327)
      Type: key_share (51)
      Length: 1327
```

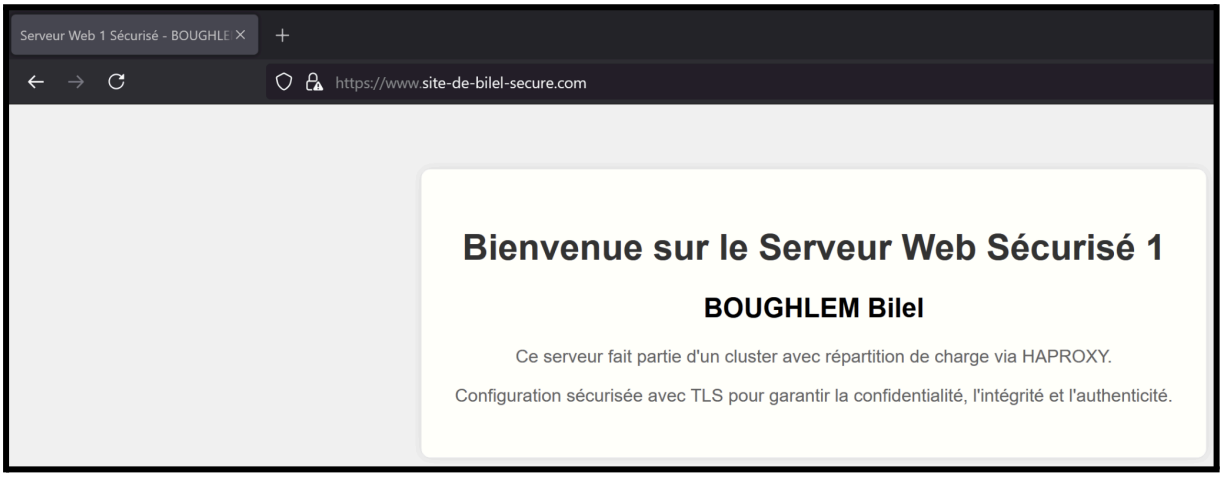
Puis le serveur fait de même :

```
▼ Extension: key_share (len=36)
      Type: key_share (51)
      Length: 36
```

Et voici le chiffrement utilisé :

```
Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
```

Et en modifiant le fichier /etc/hosts du client afin de pouvoir résoudre le FQDN www.site-de-bilel-secure.com, nous pouvons nous mettre à la place d’un client qui ne précisera pas forcément son utilisation de HTTP ou bien de HTTPS :



Au cours de ce TP, nous avons mis en place une architecture web sécurisée et hautement disponible en utilisant Apache, HAPROXY et le protocole TLS. En respectant les principes DICP (Disponibilité, Intégrité, Confidentialité et Authenticité), nous avons configuré un cluster de serveurs web avec un équilibrage de charge et une sécurisation des échanges via HTTPS.

L'installation et la configuration d'Apache ont permis de garantir l'accessibilité des services web, tandis que HAPROXY a joué un rôle clé dans la répartition des requêtes et la résilience du système face aux pannes. L'intégration du protocole TLS a assuré la confidentialité des échanges, bien que l'utilisation d'un certificat auto-signé nécessite une validation manuelle côté client.

Les tests réalisés ont confirmé le bon fonctionnement de l'ensemble des mécanismes mis en place : répartition de charge efficace, basculement automatique en cas de défaillance, persistance des sessions via les cookies et visualisation des statistiques via l'interface HAPROXY.

Ce TP nous a permis d'acquérir des compétences essentielles en administration système et en sécurisation des services web, renforçant ainsi notre capacité à concevoir des infrastructures fiables et sécurisées. Des améliorations pourraient être envisagées, telles que l'intégration d'un certificat signé par une autorité de certification (CA) ou l'optimisation des stratégies de répartition de charge pour des environnements à plus grande échelle.

R4.Cyber.11 – TP 2 : Pentesting Attaque SlowLoris

Dans le cadre de cette étude, nous avons mis en place plusieurs serveurs web sous Linux afin d'évaluer leur résilience face à une attaque Slowloris. Cette attaque, de type DDoS à connexion lente, exploite la gestion des connexions simultanées pour saturer les ressources du serveur, le rendant inopérant sans consommer une grande bande passante.

Notre méthodologie repose sur trois étapes principales :

- Déploiement de différents serveurs web (Apache, Nginx, Lighttpd, Tomcat, Caddy…) avec leurs configurations par défaut.
- Exécution de l'attaque Slowloris sur chaque serveur pour analyser leur comportement et identifier les plus vulnérables.
- Mise en place de mesures de protection (configuration spécifique, modules de sécurité, règles firewall) afin d'atténuer ou d'éliminer les effets de l'attaque.

Ce rapport présente les résultats de ces tests, les vulnérabilités identifiées, ainsi que les solutions recommandées pour renforcer la sécurité des serveurs contre Slowloris.

SOMMAIRE

Mise en place de l'environnement de test

Mise en Place du serveur Apache2

Mise en Place du serveur Nginx

Mise en Place du serveur Lighttpd

Mise en Place du serveur Caddy

Mise en place et exécution de l'attaque Slowloris

Analyse du trafic

Comparaison entre serveurs web

Résultats sur le serveur Apache2

Résultats sur le serveur Nginx

Résultats sur le serveur Lighttpd

Résultats sur le serveur Caddy

Mise en place de la disponibilité

Installation de ModSecurity pour Apache2

Hardening de la configuration pour Nginx

Hardening de la configuration pour Lighttpd

Mise en place de l'environnement de test

Mise en Place du serveur Apache2

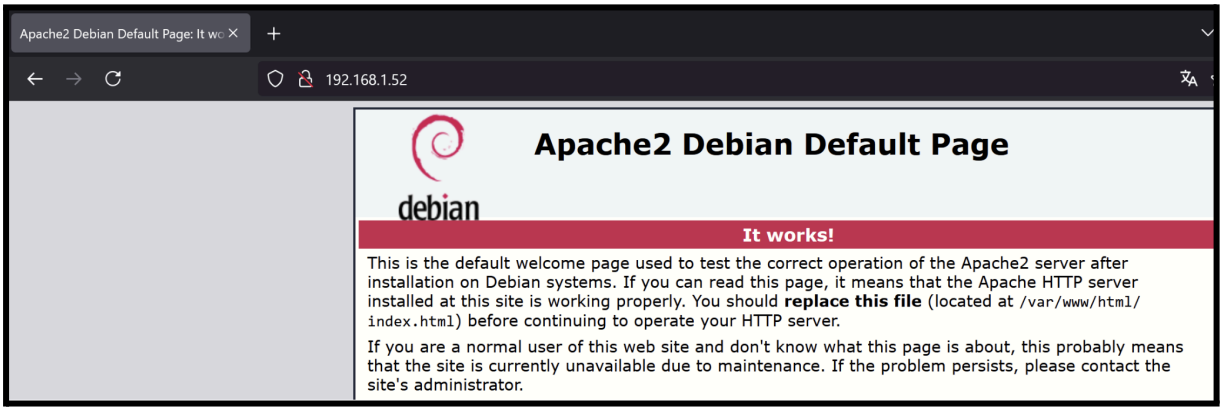
Nous installons donc le serveur Apache2 :

```
vboxuser@Cible:~$ sudo apt install apache2
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Puis vérifions que celui est bel et bien en écoute sur le port 80 au moyen de la commande *ss -ntl* :

```
vboxuser@Cible:~$ ss -ntl
State    Recv-Q    Send-Q        Local Address:Port        Peer Address:Port    Process
LISTEN   0         128               127.0.0.1:631              0.0.0.0:*
LISTEN   0         128                 0.0.0.0:22               0.0.0.0:*
LISTEN   0         128                    [::]:22                  [::]:*
LISTEN   0         128                   [::1]:631                 [::]:*
LISTEN   0         511                       *:80                     *:*
```

Et nous pouvons bel et bien accéder au service Apache2 depuis un client web :



Mise en Place du serveur Nginx

Puis faisons de même avec le serveur Nginx :

```
vboxuser@Cible:~$ sudo apt install nginx
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Mais avant tout, il faut gérer de potentiels problèmes liés à la présence de plusieurs serveurs web sur la même machine et notamment la confusion des fichiers d'index. C'est pour cela que nous modifions le fichier /etc/nginx/sites-available/default afin de définir le fichier index créé par nginx comme étant celui par défaut qui sera utilisé :

```
 GNU nano 7.2                    /etc/nginx/sites-available/default
        index    index.nginx-debian.html;
```

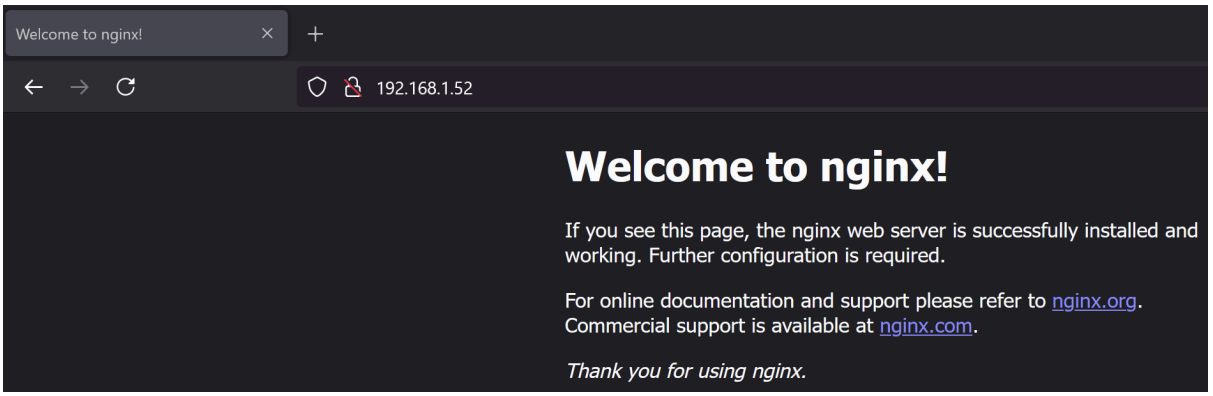
Et puisque le port 80 est déjà utilisé par notre serveur Apache2, nous stoppons ce service et lançons le service Nginx :

```
vboxuser@Cible:~$ sudo systemctl stop apache2
vboxuser@Cible:~$ sudo systemctl start nginx
vboxuser@Cible:~$ sudo systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/lib/systemd/system/nginx.service; enabled; preset: enabled)
     Active: active (running) since Mon 2025-03-03 11:32:57 +04; 4s ago
```

Et bien sûr, le port 80 est cette fois ci utilisé par Nginx :

```
vboxuser@Cible:~$ ss -ntl
State     Recv-Q     Send-Q         Local Address:Port         Peer Address:Port    Process
LISTEN    0          128                127.0.0.1:631               0.0.0.0:*
LISTEN    0          128                  0.0.0.0:22                0.0.0.0:*
LISTEN    0          511                  0.0.0.0:80                0.0.0.0:*
LISTEN    0          128                     [::]:22                   [::]:*
LISTEN    0          128                    [::1]:631                  [::]:*
LISTEN    0          511                     [::]:80                   [::]:*
```

Et nous pouvons bel et bien accéder au service Nginx depuis un client web :



Mise en Place du serveur Lighttpd

Le prochain serveur web qui sera installé est Lighttpd que nous installons :

```
vboxuser@Cible:~$ sudo apt install lighttpd
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Et puisque Lighttpd ne créée pas de fichier index, nous allons lui en créer un :

```
  GNU nano 7.2                          index_lighttpd.html
<h1>Bienvenue sur la page d'index du serveur Lighttpd</h1>
```

Et il faut s'assurer qu'il utilise cette page créée et pas une autre, alors modifions les fichiers de configuration du serveur Lighttpd :

```
  GNU nano 7.2                                  lighttpd.conf *
index-file.names                = ( "index_lighttpd.html" )
```

Et sachant que le port 80 est déjà utilisé par notre serveur Nginx, nous stoppons ce service et lançons le service Lighttpd :

```
vboxuser@Cible:~$ sudo systemctl stop nginx
vboxuser@Cible:~$ sudo systemctl start lighttpd
vboxuser@Cible:~$ sudo systemctl status lighttpd
● lighttpd.service - Lighttpd Daemon
     Loaded: loaded (/lib/systemd/system/lighttpd.service; enabled; preset: enabled)
     Active: active (running) since Mon 2025-03-03 11:37:12 +04; 3s ago
```

Et le port 80 est bel et bien en écoute :

```
vboxuser@Cible:~$ ss -ntl
State    Recv-Q    Send-Q        Local Address:Port        Peer Address:Port    Process
LISTEN   0         128               127.0.0.1:631              0.0.0.0:*
LISTEN   0         128                 0.0.0.0:22               0.0.0.0:*
LISTEN   0         1024                0.0.0.0:80               0.0.0.0:*
LISTEN   0         128                    [::]:22                  [::]:*
LISTEN   0         128                   [::1]:631                 [::]:*
LISTEN   0         1024                   [::]:80                  [::]:*
```

Et nous pouvons bel et bien accéder au service Lighttpd depuis un client web :

Mise en Place du serveur Caddy

Et le dernier serveur web qui sera installé est Caddy nous préparons son installation au moyen des trois commandes suivantes :

- *sudo apt install -y debian-keyring debian-archive-keyring apt-transport-https*
- *curl -fsSL https://dl.cloudsmith.io/public/caddy/stable/gpg.key | sudo gpg --dearmor -o /usr/share/keyrings/caddy-keyring.gpg*
- *echo "deb [signed-by=/usr/share/keyrings/caddy-keyring.gpg] https://dl.cloudsmith.io/public/caddy/stable/deb/debian any-version main" | sudo tee /etc/apt/sources.list.d/caddy-stable.list*

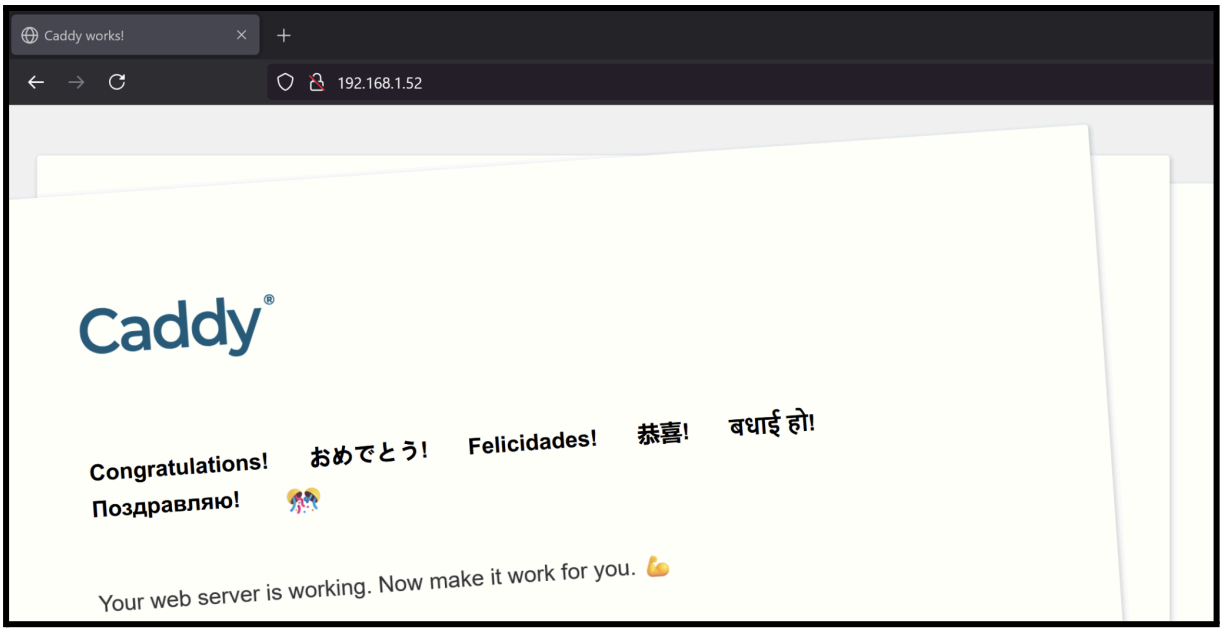
Une fois ces trois commandes exécutées, nous pouvons installer le paquet caddy :

```
vboxuser@Cible:~$ sudo apt install caddy
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Et puisque le port 80 est déjà utilisé par notre serveur Lighttpd, nous stoppons ce service et lançons le service Caddy :

```
vboxuser@Cible:~$ sudo systemctl stop lighttpd
vboxuser@Cible:~$ sudo systemctl start caddy
```

Et nous pouvons bel et bien accéder au service Caddy depuis un client web :



Tous nos services étant correctement installés et configurés, nous n'aurons plus qu'à stopper et relancer les services en fonction desquels nous voulons utiliser sur le moment…

Mise en place et exécution de l'attaque Slowloris

Nous pouvons alors préparer l'attaquant, nous commençons donc par installer le langage de programmation par lequel sera exécuté Slowloris sur la machine attaquante, nous avons choisi Python3 car c'est un langage de programmation plus répandu et auquel nous sommes plus familier :

```
┌──(kali㉿kali)-[~/slowloris]
└─$ sudo apt install python3
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
```

Puis nous installons l'outil qui attaquera les différents serveurs web, Slowloris au moyen de la commande *git clone https://github.com/gkbrk/slowloris.git* :

```
┌──(kali㉿kali)-[~]
└─$ git clone https://github.com/gkbrk/slowloris.git
Clonage dans 'slowloris'...
remote: Enumerating objects: 152, done.
remote: Counting objects: 100% (66/66), done.
remote: Compressing objects: 100% (29/29), done.
remote: Total 152 (delta 39), reused 37 (delta 37), pack-reused 86 (from 2)
Réception d'objets: 100% (152/152), 27.79 Kio | 889.00 Kio/s, fait.
Résolution des deltas: 100% (78/78), fait.
```

Et une fois dans le répertoire Slowloris, nous pouvons exécuter le script malveillant en précisant les paramètres identifiant la cible étant son adresse IP et son port au moyen de la commande *python3 slowloris.py 192.168.1.52 -p 80*, la commande sera la même pour tous les services que nous n'aurons qu'à stopper et relancer à notre guise, commençons donc par attaquer Apache2 :

```
┌──(kali㉿kali)-[~/slowloris]
└─$ python3 slowloris.py 192.168.1.52 -p 80
[03-03-2025 12:40:39] Attacking 192.168.1.52 with 150 sockets.
[03-03-2025 12:40:39] Creating sockets...
[03-03-2025 12:40:39] Sending keep-alive headers...
[03-03-2025 12:40:39] Socket count: 150
```

Cependant, après quelques tests, nous réalisons que 150 sockets n'est pas assez et nous décidons de tripler ce nombre en ajoutant l'option *-s 450* à la commande qui devient donc *python3 slowloris.py 192.168.1.52 -s 450 -p 80* :

```
┌──(kali㉿kali)-[~/slowloris]
└─$ python3 slowloris.py 192.168.1.52 -s 450 -p 80
[03-03-2025 12:45:13] Attacking 192.168.1.52 with 450 sockets.
[03-03-2025 12:45:13] Creating sockets...
[03-03-2025 12:45:14] Sending keep-alive headers...
[03-03-2025 12:45:14] Socket count: 450
```

Et c'est comme ceci que nous réglons l'intensité de l'attaque sur les différents serveurs, les résultats seront affichés plus bas mais d'abord, analysons les requêtes envoyées par Slowloris…

Analyse du trafic

Nous capturons ces requêtes au moyen du logiciel Wireshark et, de la même manière que montré précédemment, nous choisissons de ne créer qu'une seul socket afin de faciliter l'analyse :

6	4.142951386	192.168.1.25	192.168.1.52	TCP	76 43850 → 80 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TSval=24841859…
7	4.144420247	192.168.1.52	192.168.1.25	TCP	76 80 → 43850 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSv…
8	4.144451980	192.168.1.25	192.168.1.52	TCP	68 43850 → 80 [ACK] Seq=1 Ack=1 Win=32128 Len=0 TSval=2484185992 TSecr=3625…
9	4.144849176	192.168.1.25	192.168.1.52	TCP	88 43850 → 80 [PSH, ACK] Seq=1 Ack=1 Win=32128 Len=20 TSval=2484185992 TSec…
10	4.145341065	192.168.1.52	192.168.1.25	TCP	68 80 → 43850 [ACK] Seq=1 Ack=21 Win=65152 Len=0 TSval=3625532627 TSecr=248…
11	4.145350934	192.168.1.25	192.168.1.52	TCP	236 43850 → 80 [PSH, ACK] Seq=21 Ack=1 Win=32128 Len=168 TSval=2484185992 TS…
12	4.145668474	192.168.1.52	192.168.1.25	TCP	68 80 → 43850 [ACK] Seq=1 Ack=189 Win=65024 Len=0 TSval=3625532627 TSecr=24…
13	4.146480182	192.168.1.25	192.168.1.52	TCP	79 GET /?843 HTTP/1.1 [TCP segment of a reassembled PDU]
14	4.148337448	192.168.1.52	192.168.1.25	TCP	68 80 → 43850 [ACK] Seq=1 Ack=200 Win=65024 Len=0 TSval=3625532628 TSecr=24…

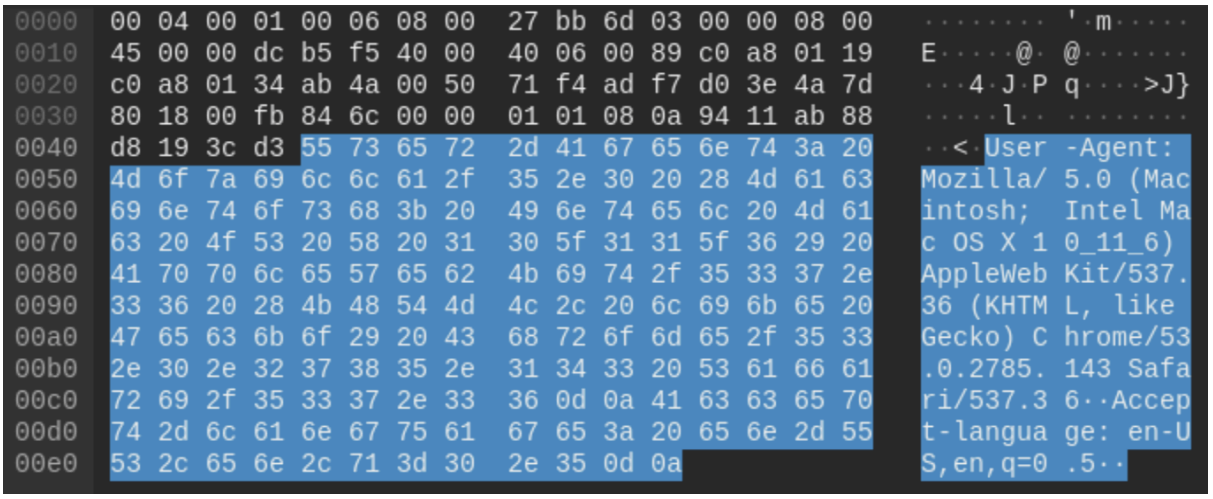
Et voici donc les trames transmises… Une connexion TCP semblant classique au départ avec les trois trames d'un mise en place de la connexion étant [SYN] puis [SYN, ACK] puis [ACK]. Mais après cela, l'attaquant transmet la trame 9 contenant une requête incomplète :

```
0000  00 04 00 01 00 06 08 00  27 bb 6d 03 00 00 08 00   ........ '·m·····
0010  45 00 00 48 b5 f4 40 00  40 06 01 1e c0 a8 01 19   E··H··@· @·······
0020  c0 a8 01 34 ab 4a 00 50  71 f4 ad e3 d0 3e 4a 7d   ···4·J·P q····>J}
0030  80 18 00 fb 83 d8 00 00  01 01 08 0a 94 11 ab 88   ........ ........
0040  d8 19 3c d2 47 45 54 20  2f 3f 38 34 33 20 48 54   ··<·GET  /?843 HT
0050  54 50 2f 31 2e 31 0d 0a                           TP/1.1··
```

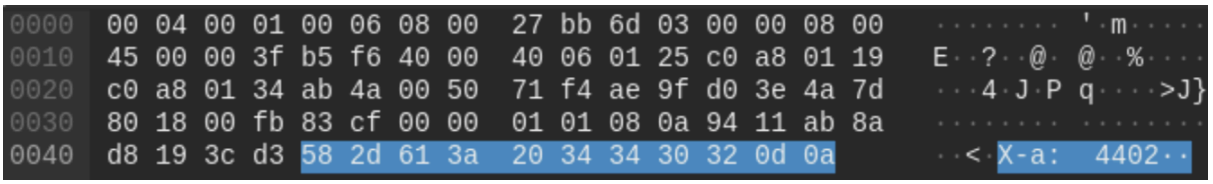
Car il ne précise pas de Host ou d'User Agent… Un exemple de trame valide serait ceci :

GET /?843 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0

Il ne transmet le Host et l'User Agent que dans la prochain trame 11 :



Et puis dans la prochain trame 13 qu'il transmet, il maintient la connexion :



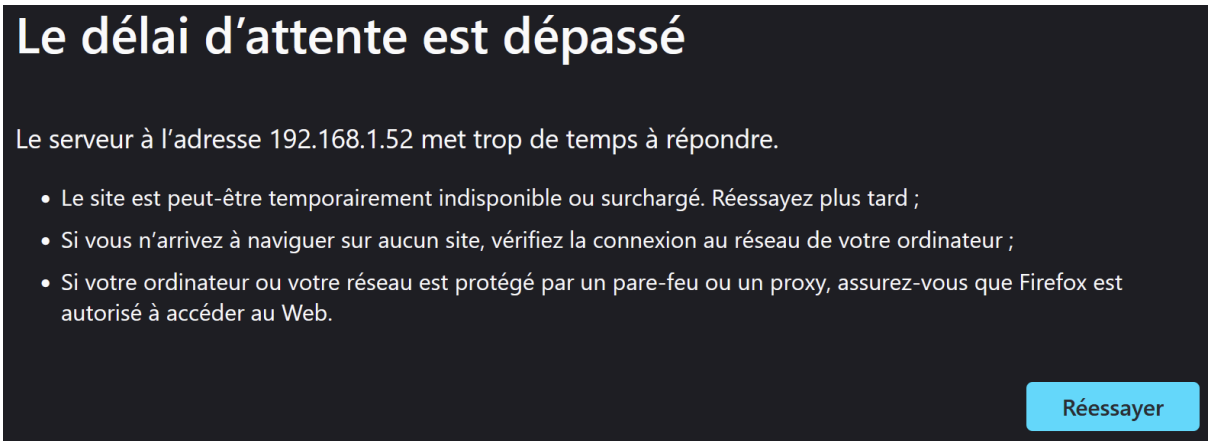
A aucun moment, il ne met fin à la connexion dans le but de faire travailler le serveur le plus longtemps possible.

Comparaison entre serveurs web

Comparons le nombre de sockets nécessaires avant de rendre le service indisponible sachant que nous augmentons le nombre de sockets graduellement de 50 en 50 en commençant par 50…

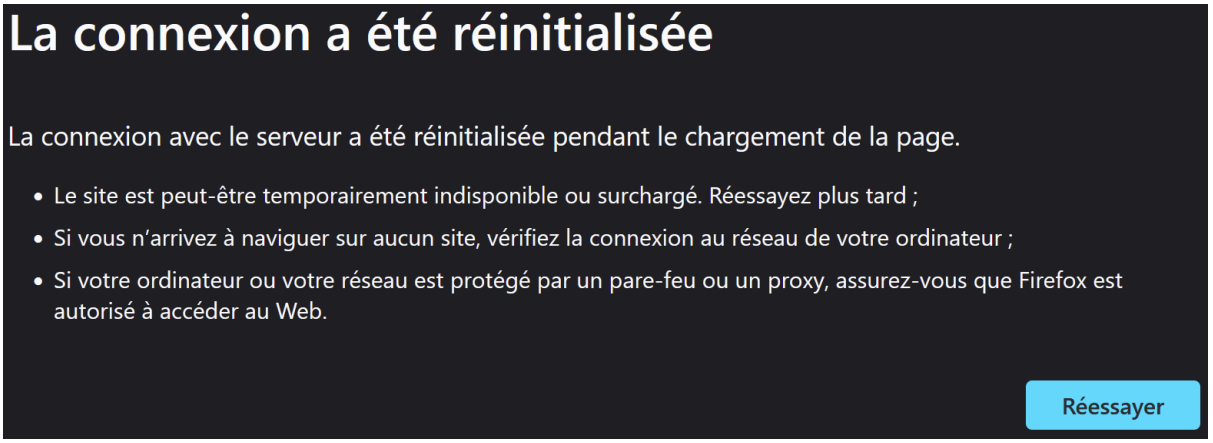
Résultats sur le serveur Apache2

Au bout de 150 sockets, le service commence à ralentir et plus nous augmentons le nombre et plus le service devient pratiquement inutilisable jusqu'à ce que nous atteignons le nombre 800 :



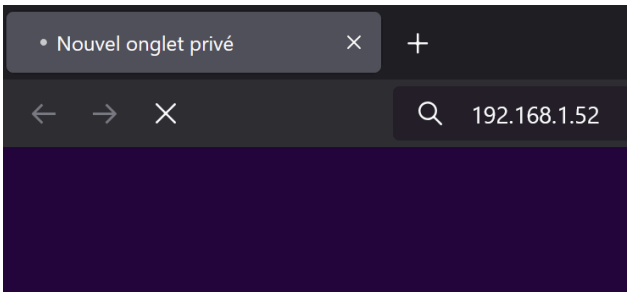
Résultats sur le serveur Nginx

Au bout de 800 sockets, le service qui jusque là était totalement fonctionnel et n'était pas ralenti s'arrête subitement :



Résultats sur le serveur Lighttpd

Au bout de 400 sockets, le service commence à ralentir et plus nous augmentons le nombre et plus le service devient pratiquement inutilisable jusqu'à ce que nous atteignons le nombre 600, la page continue de chercher la page sans la trouver et sans retourner d'erreur :



Résultats sur le serveur Caddy

Quant au serveur Caddy, il tient bon et au bout de 2000 sockets, il ne ralentit toujours pas alors nous décidons d'arrêter l'expérience car c'est la machine attaquante qui arrive à bout en première.

Caddy gère bien les attaques DDoS grâce à son architecture asynchrone et son modèle de gestion des connexions basé sur Go, qui lui permet de traiter un grand nombre de requêtes simultanées sans bloquer les ressources. Contrairement à d'autres serveurs, qui utilisent des modèles basés sur des threads, Caddy utilise des goroutines légères, réduisant ainsi l'impact des connexions lentes comme celles exploitées par Slowloris. De plus, il intègre HTTP/2 et QUIC, qui optimisent la gestion des connexions et limitent la surcharge. Enfin, Caddy dispose de timeouts configurables et d'options de rate-limiting, renforçant sa résistance aux attaques volumétriques.

Mise en place de la disponibilité

Cependant, nous avons trois services n'ayant pas réussi à gérer le grand nombre de requêtes, nous allons donc les configurer afin d'assurer la disponibilité pour les utilisateurs légitimes…

Installation de ModSecurity pour Apache2

Nous allons donc commencer par installer l'extension sécurisé pour Apache2 étant ModSecurity ainsi que les règles OWASP pour ModSecurity :

```
vboxuser@Cible:~$ sudo apt install libapache2-mod-security2 modsecurity-crs
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Une fois installée, nous pouvons activer l'extension avec la commande *sudo a2enmod security2* :

```
vboxuser@Cible:~$ sudo a2enmod security2
Considering dependency unique_id for security2:
Module unique_id already enabled
Module security2 already enabled
```

Il se trouve que l'extension était déjà activée, mais il fallait s'en assurer et nous pouvons continuer en redémarrant le service Apache2 puis en vérifiant le statut de l'extension au moyen de la commande *sudo apache2ctl -M | grep security2* :

```
vboxuser@Cible:~$ sudo apache2ctl -M | grep security2
AH00558: apache2: Could not reliably determine the server's fully qualified domain name
, using fe80::a00:27ff:fefd:84c1%enp0s3. Set the 'ServerName' directive globally to sup
press this message
 security2_module (shared)
```

Il semble qu'il y ait un problème, il faut que nous définissons un ServerName afin qu'il n'aille pas en chercher un incorrect… Nous allons donc ajouter la ligne ServerName 127.0.0.1 à la fin du fichier de configuration d'Apache2 /etc/apache2/apache2.conf :

```
  GNU nano 7.2                        /etc/apache2/apache2.conf
ServerName      127.0.0.1
```

Et si l'extension est en mode détection de menaces par défaut, nous allons changer ceci en mode prévention, passer en quelque sorte de IDS à IPS en modifiant le fichier /etc/modsecurity/modsecurity.conf-recommended :

```
  GNU nano 7.2                          modsecurity.conf-recommended
#SecRuleEngine DetectionOnly
SecRuleEngine On
```

Nous pouvons donc redémarrer le service et tester la configuration une nouvelle fois :

```
vboxuser@Cible:~$ sudo systemctl restart apache2
vboxuser@Cible:~$ sudo apache2ctl -M | grep security2
 security2_module (shared)
```

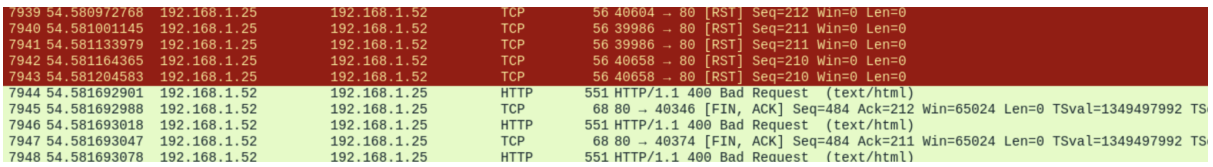
Tout fonctionne, et sachant que les règles de base du module sont déjà installées dans le paquet modsecurity-crs précédemment installé, nous pouvons donc tester notre configuration et sa robustesse face aux attaques de type Slowloris…

Et le serveur tient toujours malgré la machine attaquante utilisant sa pleine puissance :

```
┌──(kali㉿kali)-[~/slowloris]
└─$ python3 slowloris.py 192.168.1.52 -s 2000 -p 80
[07-03-2025 12:05:01] Attacking 192.168.1.52 with 2000 sockets.
[07-03-2025 12:05:01] Creating sockets...
[07-03-2025 12:05:04] Sending keep-alive headers...
[07-03-2025 12:05:04] Socket count: 0
[07-03-2025 12:05:04] Creating 2000 new sockets...
[07-03-2025 12:05:22] Sending keep-alive headers...
[07-03-2025 12:05:22] Socket count: 0
[07-03-2025 12:05:22] Creating 2000 new sockets...
```

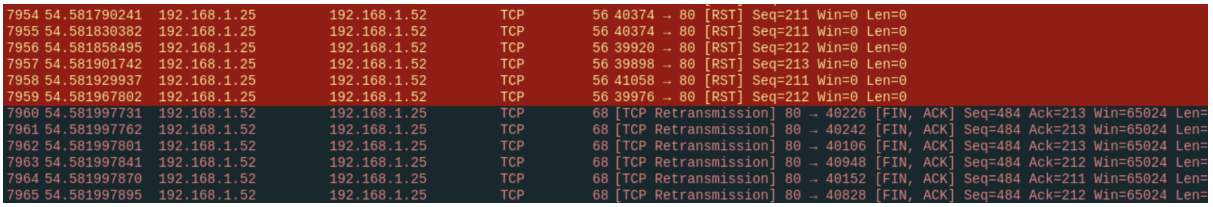
Sur les 2000 sockets que la machine attaquante devrait créer, aucune n'est réellement créée…

Et nous pouvons analyser les paquets transmis lors de cet échange :



Les paquets transmis par la machine attaquante sont affichés en rouge à partir car étant incomplets, pour rappel, Slowloris envoie des paquets incomplets afin de maintenir la connexion ouverte le plus longtemps ouverte ce qui cause la réponse avec les paquets en verts de la machine cible étant une erreur 400 pour une mauvaise requête.

Cependant, cela ne reste pas comme ça :



Au bout d'un certain temps, la machine cible comprend qu'elle est attaquée et met fin à tous les sockets de la machine cible, c'est modsecurity en action !

Hardening de la configuration pour Nginx

Nginx permet de limiter le nombre de connexions simultanées et les requêtes par adresse IP, ce qui peut protéger notre serveur cible d'une attaque SlowLoris.

Nous allons donc ajouter ces lignes dans la section http :

client_body_timeout 5s;
client_header_timeout 5s;

```
  GNU nano 7.2                          /etc/nginx/nginx.conf
http {

        client_body_timeout 5s;
        client_header_timeout 5s;
```

Puis nous pouvons vérifions la syntaxe de notre configuration avec la commande *sudo nginx -t* avant de redémarrer le service :

```
vboxuser@Cible:~$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
vboxuser@Cible:~$ sudo systemctl restart nginx
```

Et nous pouvons donc tester la robustesse de notre serveur en tentant de le ralentir voir stopper avec Slowloris sur la machine attaquante en pleine puissance avec 2000 sockets.

Durant les cinq premières secondes de l'attaque, le serveur réagit comme s'il était attaqué et est indisponible puis réagit en mettant fin au connexions initiées par la machine attaquante.

C'est logique car nous avons défini le temps pendant lequel Nginx attend la réponse du client comme 5 secondes, au bout desquelles le serveur cible met fin à la connexion…

Et pour mieux comprendre, observons le fichier log Nginx /var/log/nginx/access.log :

```
vboxuser@Cible:~$ sudo tail -3 /var/log/nginx/access.log
192.168.1.25 - - [05/Mar/2025:12:12:03 +0400] "GET /?1826 HTTP/1.1" 408 0 "-" "Mozilla/
5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/5
3.0.2785.143 Safari/537.36"
192.168.1.25 - - [05/Mar/2025:12:12:03 +0400] "GET /?1846 HTTP/1.1" 408 0 "-" "Mozilla/
5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/5
3.0.2785.143 Safari/537.36"
192.168.1.25 - - [05/Mar/2025:12:12:03 +0400] "GET /?493 HTTP/1.1" 408 0 "-" "Mozilla/5
.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53
.0.2785.143 Safari/537.36"
```

Le serveur envoie des erreurs de type 408 à toutes les requêtes de Slowloris, le code d'erreur 408 correspondant à un timeout de la requête ce qui signifie que le serveur met fin aux sockets après 5 secondes d'inactivité.

Hardening de la configuration pour Lighttpd

Nous allons faire de même pour Lighttpd en ajoutant la ligne *server.max-read-idle = 5* dans le fichier de configuration /etc/lighttpd/lighttpd.conf :

```
  GNU nano 7.2                    /etc/lighttpd/lighttpd.conf
server.document-root        = "/var/www/html"
server.upload-dirs          = ( "/var/cache/lighttpd/uploads" )
server.errorlog             = "/var/log/lighttpd/error.log"
server.pid-file             = "/run/lighttpd.pid"
server.username             = "www-data"
server.groupname            = "www-data"
server.port                 = 80
server.max-read-idle        = 5
```

Et nous pouvons redémarrer le service puis vérifier si nos configurations protègent le serveur…

Et de la même manière que pour le serveur Nginx, la page est lente pendant les 5 premières secondes avant de fonctionner normalement…

En analysant le fichier de journal des erreurs de Lighttpd, /var/log/lighttpd/error.log, nous remarquons que le serveur met fin à des sockets avant de les autoriser à nouveau en fonction de la limite de connection étant le timeout configuré :

```
vboxuser@Cible:~$ sudo tail -10 /var/log/lighttpd/error.log
2025-03-05 12:29:33: (server.c.1081) [note] sockets enabled again
2025-03-05 12:29:36: (server.c.1089) [note] sockets disabled, connection limit reached
2025-03-05 12:29:49: (server.c.1081) [note] sockets enabled again
2025-03-05 12:29:52: (server.c.1089) [note] sockets disabled, connection limit reached
2025-03-05 12:30:04: (server.c.1081) [note] sockets enabled again
2025-03-05 12:30:07: (server.c.1089) [note] sockets disabled, connection limit reached
2025-03-05 12:30:19: (server.c.1081) [note] sockets enabled again
2025-03-05 12:30:23: (server.c.1089) [note] sockets disabled, connection limit reached
2025-03-05 12:30:35: (server.c.1081) [note] sockets enabled again
2025-03-05 12:30:38: (server.c.1089) [note] sockets disabled, connection limit reached
```

Bien sûr, dans un cas où 5 secondes d'inactivité est trop long, il est possible d'ajuster cette durée de timeout de manière à ce qu'elle ne soit ni trop laxiste ni trop agressive en fonction du besoin…

Au terme de ce TP sur l'attaque Slowloris et la résistance des serveurs web, nous avons pu identifier les vulnérabilités inhérentes à certains logiciels et les solutions permettant d'y remédier. L'expérimentation a mis en lumière les différences notables entre les serveurs Apache2, Nginx, Lighttpd et Caddy face à cette attaque de type DDoS à connexion lente.

L'analyse des résultats a montré que certains serveurs, comme Caddy, sont naturellement plus résistants grâce à leur architecture optimisée, tandis que d'autres, comme Apache2 et Lighttpd, nécessitent des ajustements pour atténuer l'impact de Slowloris. Nginx, bien que plus robuste, a également nécessité une configuration spécifique pour renforcer sa résilience.

L'application de contre-mesures telles que l'installation de ModSecurity pour Apache2, la configuration de timeouts et de limitations de connexions pour Nginx et Lighttpd a permis de renforcer la sécurité des serveurs testés. Ces mesures ont démontré leur efficacité en empêchant l'attaque de saturer les ressources du serveur, assurant ainsi la disponibilité du service pour les utilisateurs légitimes.

Ce TP a également permis d'approfondir l'utilisation d'outils d'analyse comme Wireshark pour observer le comportement du trafic réseau et d'identifier les signatures spécifiques d'une attaque Slowloris. L'analyse des logs des serveurs a confirmé le bon fonctionnement des protections mises en place…