

Compte rendu de travaux pratiques

TP2

Héberger deux sites web et les publier en HTTPS

Nom : PACHOT

Prénom : Dorian

Classe : BTS SIO 2e année

Module : Linux avancé

Date de la séance : 8 septembre 2026

1. Objectif

Dans ce TP, je mets deux sites web en ligne sur le serveur srv-linux monté au TP1. Le premier, `www.bts-sio.lan`, est une vitrine publique en HTTP. Le second, `intranet.bts-sio.lan`, est réservé : il n'est joignable que depuis le réseau `192.168.100.0/24`, il demande un mot de passe, et il est publié en HTTPS avec un certificat que je génère moi-même.

Les deux sites sont sur la même machine et la même adresse IP. C'est tout l'intérêt des hôtes virtuels : c'est l'en-tête Host de la requête qui dit à Apache quel site est demandé.

La séance se termine par trois ajouts. Une page PHP dynamique avec un formulaire, une installation de Nginx sur le port 8080 à côté d'Apache, et l'ouverture du pare-feu sans couper l'accès SSH.

2. Environnement de travail

Machine	Système	Adresse IP	Rôle
poste-hote	Windows 11, VMware Workstation Pro 17.6.1	192.168.100.1 / 24 sur VMnet2	Poste étudiant, navigateur de test
srv-linux	Debian 12 « bookworm », 2 vCPU, 2 Go de RAM	ens32 : NAT en DHCP ; ens33 : 192.168.100.10 / 24 statique	DHCP, DNS, Apache, Nginx
cli-linux	Debian 12, clone de srv-linux	ens33 : 192.168.100.50 par réservation DHCP	Client de test, commandes curl

Les deux machines viennent du TP1 et sont réparties de l'instantané de fin de séance. L'énoncé parle de l'interface `ens33` pour le réseau pédagogique, et c'est bien le nom sur mes machines. Rien à transposer cette fois, contrairement au TP1.

La machine Ubuntu du TP0 reste éteinte. Elle porte aussi `192.168.100.10` sur VMnet2.

Avant de commencer, j'ai pris un instantané VMware des deux VM, nommé TP2 debut.

3. Déroulé

Partie A. Vérifier les prérequis

Étape 1. Contrôler le DNS mis en place au jour 2

Rien de ce TP ne marchera si la résolution de noms est cassée. Je commence donc par la vérifier des deux côtés.

ip -br a ; systemctl is-active named isc-dhcp-server

```
sio@srv-linux:~$ ip -br a
lo                UNKNOWN          127.0.0.1/8 ::1/128
ens32             UP              192.168.88.134/24 fe80::20c:29ff:fec1:907f/64
ens33             UP              192.168.100.10/24 fe80::20c:29ff:fec1:9089/64

sio@srv-linux:~$ systemctl is-active named isc-dhcp-server
active
active
```

Les deux cartes sont montées et les deux services du TP1 tournent toujours. ens33 porte bien son adresse fixe.

dig @localhost www.bts-sio.lan +short

```
sio@srv-linux:~$ dig @localhost www.bts-sio.lan +short
;; communications error to ::1#53: connection refused
;; communications error to ::1#53: connection refused
;; communications error to ::1#53: connection refused
srv-linux.bts-sio.lan.
192.168.100.10

sio@srv-linux:~$ dig @localhost intranet.bts-sio.lan +short
;; communications error to ::1#53: connection refused
;; communications error to ::1#53: connection refused
;; communications error to ::1#53: connection refused
srv-linux.bts-sio.lan.
192.168.100.10
```

Les trois lignes d'erreur ne sont pas une panne, je les explique en section « Difficultés rencontrées ». La réponse arrive bien juste après. Les deux noms mènent à 192.168.100.10, en passant par le même alias.

Je refais la même chose depuis cli-linux, sans préciser de serveur cette fois. C'est le DHCP qui doit avoir poussé le bon résolveur.

```
sio@cli-linux:~$ cat /etc/resolv.conf
domain bts-sio.lan
search bts-sio.lan
nameserver 192.168.100.10

sio@cli-linux:~$ dig www.bts-sio.lan +short
srv-linux.bts-sio.lan.
192.168.100.10

sio@cli-linux:~$ dig intranet.bts-sio.lan +short
srv-linux.bts-sio.lan.
192.168.100.10
```

Le client n'a aucune configuration DNS locale. Les trois lignes de resolv.conf viennent du bail DHCP. Les deux noms se résolvent quand même, et sans message d'erreur : ici j'interroge 192.168.100.10 en IPv4 directement, pas localhost.

ping -c 2 www.bts-sio.lan

```
sio@cli-linux:~$ ping -c 2 www.bts-sio.lan
```

```

PING srv-linux.bts-sio.lan (192.168.100.10) 56(84) bytes of data.
64 bytes from srv-linux.bts-sio.lan (192.168.100.10): icmp_seq=1 ttl=64 time=0.783
ms
64 bytes from srv-linux.bts-sio.lan (192.168.100.10): icmp_seq=2 ttl=64 time=0.585
ms
--- srv-linux.bts-sio.lan ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms

```

Deux paquets envoyés, deux reçus. J'ai tapé `www.bts-sio.lan` et ping affiche `srv-linux.bts-sio.lan` : l'alias a été suivi, puis l'adresse a été retraduite en nom par la zone inverse. Les prérequis sont bons.

Étape 2. Rendre les noms résolubles depuis Windows

Le poste Windows n'utilise pas mon DNS mais celui de l'établissement. Il ne connaît donc pas `bts-sio.lan`. Je le vérifie avant de toucher à quoi que ce soit.

```

PS C:\Users\doria> nslookup www.bts-sio.lan
Serveur : dns.google
Address: 8.8.8.8
*** dns.google ne parvient pas a trouver www.bts-sio.lan : Non-existent domain

PS C:\Users\doria> ping 192.168.100.10
Reponse de 192.168.100.10 : octets=32 temps<1ms TTL=64
Paquets : envoyes = 4, recus = 4, perdus = 0 (perte 0%),

```

Le résultat est net. Le nom échoue, l'adresse répond. Ce n'est donc pas un problème de réseau mais de résolution de noms. Mon poste interroge 8.8.8.8, un serveur public, qui ne connaîtra jamais un domaine en .lan.

J'ajoute donc les trois noms dans le fichier `hosts` de Windows, qui se trouve dans `C:\Windows\System32\drivers\etc\`. Il demande les droits administrateur.

```

192.168.100.10    srv-linux.bts-sio.lan    srv-linux
192.168.100.10    www.bts-sio.lan
192.168.100.10    intranet.bts-sio.lan

```

Je vide ensuite le cache DNS de Windows, sinon les échecs précédents restent en mémoire, puis je teste :

```

PS C:\Users\doria> Get-Content C:\Windows\System32\drivers\etc\hosts | Select-String bts-sio
192.168.100.10    srv-linux.bts-sio.lan    srv-linux
192.168.100.10    www.bts-sio.lan
192.168.100.10    intranet.bts-sio.lan

PS C:\Users\doria> ipconfig /flushdns
Cache de resolution DNS vide.

PS C:\Users\doria> ping www.bts-sio.lan
Envoi d'une requete 'ping' sur www.bts-sio.lan [192.168.100.10] avec 32 octets de donnees :
Reponse de 192.168.100.10 : octets=32 temps<1ms TTL=64
Paquets : envoyes = 4, recus = 4, perdus = 0 (perte 0%),

PS C:\Users\doria> ping intranet.bts-sio.lan
Envoi d'une requete 'ping' sur intranet.bts-sio.lan [192.168.100.10] avec 32 octets de donnees :
Reponse de 192.168.100.10 : octets=32 temps<1ms TTL=64
Paquets : envoyes = 4, recus = 4, perdus = 0 (perte 0%),

```

Les deux noms répondent maintenant depuis Windows. Le poste peut donc atteindre mes deux sites par leur nom, ce qui est nécessaire pour tester les hôtes virtuels au navigateur.

Un détail m'a surpris. Après la modification, nslookup continue d'échouer alors que ping marche. C'est normal, et c'est expliqué en réponse à la question 2 : les deux commandes ne prennent pas le même chemin.

Partie B. Installer Apache et découvrir son arborescence

Étape 3. Installer le paquet et vérifier le service

sudo apt install -y apache2 apache2-utils

L'installation tire onze paquets et se termine seule. La fin de la sortie est la partie intéressante : Debian ne se contente pas de poser des fichiers, elle active tout de suite une configuration de départ.

```
Enabling module mpm_event.  
Enabling module authz_core.  
Enabling module auth_basic.  
[...]  
Enabling conf security.  
Enabling site 000-default.  
Created symlink /etc/systemd/system/multi-user.target.wants/apache2.service  
→ /lib/systemd/system/apache2.service.
```

Trois choses à retenir de ces lignes. Le MPM activé est mpm_event, le plus efficace, qui est le défaut sur Debian 12. Il changera tout seul en partie G quand j'installerai PHP. Le site 000-default est activé d'office : c'est lui qui sert la page Debian par défaut, et je le désactiverai en partie C. Enfin le lien symbolique créé par systemd veut dire que le service est déjà persistant, sans que j'aie à taper enable.

```
sio@srv-linux:~$ systemctl status apache2 --no-pager  
● apache2.service - The Apache HTTP Server  
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; preset: enabled)  
   Active: active (running) since Tue 2026-09-08 15:40:55 CEST; 1s ago  
     Main PID: 2024 (apache2)  
       CGroup: /system.slice/apache2.service  
              └─2024 /usr/sbin/apache2 -k start  
                └─2025 /usr/sbin/apache2 -k start  
                  └─2026 /usr/sbin/apache2 -k start  
  
sio@srv-linux:~$ systemctl is-enabled apache2  
enabled
```

Le service est actif et persistant. C'est une différence nette avec isc-dhcp-server du TP1, qui échouait juste après son installation. Apache, lui, a une configuration par défaut qui fonctionne toute seule.

sudo ss -tulpn | grep :80

```
sio@srv-linux:~$ sudo ss -tulpn | grep :80  
tcp LISTEN 0 511 *:80 *.* users:(("apache2",pid=2026,fd=4),  
                                ("apache2",pid=2025,fd=4),  
                                ("apache2",pid=2024,fd=4))
```

L'étoile devant le port est le point important, je la commente en réponse à la question 4. Les trois processus partagent le même descripteur de fichier fd=4 : c'est le processus maître qui a ouvert le port et l'a transmis à ses enfants.

J'ai voulu tester avec curl, qui n'était pas installé :

```
sio@srv-linux:~$ curl -I http://localhost/  
-bash: curl : commande introuvable
```

Debian 12 en installation minimale ne livre pas curl. Je l'installe, et le test passe :

```
sio@srv-linux:~$ curl -I http://localhost/  
HTTP/1.1 200 OK
```

```
Date: Tue, 08 Sep 2026 13:45:00 GMT
Server: Apache/2.4.68 (Debian)
Last-Modified: Tue, 08 Sep 2026 13:40:54 GMT
Content-Length: 10701
Content-Type: text/html
```

Le code 200 dit que la page a été trouvée et renvoyée. L'en-tête Server annonce Apache 2.4.68 sur Debian, ce qu'on masque en production pour ne pas indiquer sa version exacte à un attaquant. L'option -l n'envoie qu'un HEAD : je reçois les en-têtes sans les 10 701 octets de la page.

Installer curl sur cli-linux a demandé un détour, raconté en section « Difficultés rencontrées ».

Étape 4. Explorer /etc/apache2 et identifier l'utilisateur du service

```
sio@srv-linux:~$ ls -l /etc/apache2/
-rw-r--r-- 1 root root 7178 apache2.conf
drwxr-xr-x 2 root root 4096 conf-available
drwxr-xr-x 2 root root 4096 conf-enabled
-rw-r--r-- 1 root root 1782 envvars
drwxr-xr-x 2 root root 12288 mods-available
drwxr-xr-x 2 root root 4096 mods-enabled
-rw-r--r-- 1 root root 274 ports.conf
drwxr-xr-x 2 root root 4096 sites-available
drwxr-xr-x 2 root root 4096 sites-enabled
```

Voici à quoi sert chaque chemin, d'après mes observations :

Chemin	À quoi il sert
/etc/apache2/apache2.conf	Fichier maître : réglages globaux et inclusion de tous les autres
/etc/apache2/ports.conf	Les directives Listen, donc les ports d'écoute (80, et 443 si mod_ssl est chargé)
/etc/apache2/sites-available/	Tous les hôtes virtuels écrits, actifs ou non. Un fichier .conf par site
/etc/apache2/sites-enabled/	Liens symboliques vers les sites réellement actifs
/etc/apache2/mods-enabled/	Liens symboliques vers les modules chargés au démarrage
/etc/apache2/envvars	Les variables du service : compte, groupe, dossier des journaux
/var/www/html/	Racine documentaire du site par défaut de Debian
/var/log/apache2/	access.log, error.log et les journaux propres à chaque hôte virtuel

```
ls -l /etc/apache2/sites-available/ /etc/apache2/sites-enabled/
```

```
/etc/apache2/sites-available/:
-rw-r--r-- 1 root root 1286 000-default.conf
-rw-r--r-- 1 root root 6195 default-ssl.conf

/etc/apache2/sites-enabled/:
lrwxrwxrwx 1 root root 35 000-default.conf -> ../sites-available/000-default.conf
```

La différence saute aux yeux et fait l'objet de la question 5. Deux fichiers écrits d'un côté, un seul lien de l'autre. Le site default-ssl est donc présent sur la machine mais pas actif.

```
grep -E "^export" /etc/apache2/envvars
```

```
export APACHE_RUN_USER=www-data
export APACHE_RUN_GROUP=www-data
export APACHE_PID_FILE=/var/run/apache2$SUFFIX/apache2.pid
export APACHE_RUN_DIR=/var/run/apache2$SUFFIX
export APACHE_LOCK_DIR=/var/lock/apache2$SUFFIX
export APACHE_LOG_DIR=/var/log/apache2$SUFFIX
```

```
ps aux | grep [a]pache2
```

```
root      2024  0.0  0.2   6584  4748 ?    Ss   15:40   /usr/sbin/apache2 -k start
www-data  2025  0.0  0.3 1212588  6884 ?    Sl   15:40   /usr/sbin/apache2 -k start
www-data  2026  0.0  0.3 1212588  6884 ?    Sl   15:40   /usr/sbin/apache2 -k start
www-data  2115  0.0  0.0   3424   164 ?    Ss   15:40   /usr/bin/htcacheclean -d 120
```

Un processus root et deux processus www-data, plus un htcachelean qui n'est pas un travailleur Apache mais l'utilitaire de nettoyage du cache disque. Je détaille ce partage des rôles en réponse à la question 6.

```
ls -l /var/www/html/  
-rw-r--r-- 1 root root 10701 index.html
```

Un seul fichier : la page Debian par défaut, celle que renvoyait le curl de l'étape 3. Les 10 701 octets correspondent bien au Content-Length lu dans les en-têtes.

Partie C. Créer le site public

Étape 5. Créer l'arborescence et la page d'accueil

```
sudo mkdir -p /var/www/www.bts-sio.lan/public_html
```

Je range chaque site dans son propre dossier, avec un sous-dossier public_html. Tout ce qui est en dehors de public_html ne sera jamais servi par le web, même si quelqu'un devine son nom. C'est là qu'on met les mots de passe et les scripts.

J'écris ensuite la page d'accueil, puis je pose le propriétaire et les droits :

```
sio@srv-linux:~$ sudo chown -R root:www-data /var/www/www.bts-sio.lan  
sio@srv-linux:~$ sudo chmod 755 /var/www/www.bts-sio.lan  
sio@srv-linux:~$ sudo chmod 755 /var/www/www.bts-sio.lan/public_html  
sio@srv-linux:~$ sudo chmod 644 /var/www/www.bts-sio.lan/public_html/index.html  
  
sio@srv-linux:~$ ls -lR /var/www/www.bts-sio.lan  
/var/www/www.bts-sio.lan:  
drwxr-xr-x 2 root www-data 4096  8 sept. 15:47 public_html  
  
/var/www/www.bts-sio.lan/public_html:  
-rw-r--r-- 1 root www-data 363  8 sept. 15:47 index.html
```

Le propriétaire est root et le groupe www-data. Apache peut donc lire, mais pas écrire. C'est voulu : une faille dans le serveur web ne doit pas permettre de modifier mes pages. Le détail des droits 755 et 644 est en réponse à la question 8.

Étape 6. Écrire et activer l'hôte virtuel

Le fichier est recopié en section 5. L'extension .conf est obligatoire : Debian n'inclut que sites-enabled/*.conf, et un fichier mal nommé serait ignoré sans le moindre message.

```
sio@srv-linux:~$ sudo a2ensite www.bts-sio.lan.conf  
Enabling site www.bts-sio.lan.  
To activate the new configuration, you need to run:  
    systemctl reload apache2  
  
sio@srv-linux:~$ sudo a2dissite 000-default.conf  
Site 000-default disabled.  
  
sio@srv-linux:~$ sudo apache2ctl configtest  
Syntax OK  
sio@srv-linux:~$ sudo systemctl reload apache2
```

Les deux commandes disent elles-mêmes qu'elles n'ont rien rechargé. Elles ne font que créer ou supprimer un lien symbolique, et me rappellent de faire le reload. Je contrôle toujours la syntaxe avant : une faute empêcherait Apache de repartir et couperait tous les sites de la machine.

```
sudo apache2ctl -S
```

```
VirtualHost configuration:  
*:80    www.bts-sio.lan (/etc/apache2/sites-enabled/www.bts-sio.lan.conf:1)  
ServerRoot: "/etc/apache2"
```

```
Main DocumentRoot: "/var/www/html"
Main ErrorLog: "/var/log/apache2/error.log"
PidFile: "/var/run/apache2/apache2.pid"
User: name="www-data" id=33
Group: name="www-data" id=33
```

Mon site est chargé, avec son fichier et sa ligne. Il est seul sur le port 80 pour l'instant, ce qui explique que la sortie ne parle ni de NameVirtualHost ni de default server. Ces mentions apparaîtront en partie D quand j'ajouterai l'intranet, et je compare les deux sorties en réponse à la question 10.

```
ls -l /etc/apache2/sites-enabled/
```

```
lrwxrwxrwx 1 root root 39 www.bts-sio.lan.conf ->
../sites-available/www.bts-sio.lan.conf
```

Un seul lien. Celui de 000-default a disparu, mais son fichier est toujours dans sites-available. Désactiver n'est pas supprimer.

Étape 7. Tester depuis le client, puis depuis Windows

Je fais les trois tests dans un ordre précis, du plus indépendant au plus complet.

```
curl -s -H "Host: www.bts-sio.lan" http://192.168.100.10/ | head -5
```

```
<!DOCTYPE html>
<html lang="fr">
<head><meta charset="UTF-8"><title>BTS SIO - site vitrine</title></head>
<body>
<h1>Bienvenue sur www.bts-sio.lan</h1>
```

C'est bien ma page, et pas celle de Debian. Cette commande contacte l'adresse IP directement et fabrique l'en-tête Host à la main : elle ne fait aucune résolution de nom. Je l'explique en réponse à la question 11.

```
curl -I http://www.bts-sio.lan/
```

```
HTTP/1.1 200 OK
Date: Tue, 08 Sep 2026 13:49:23 GMT
Server: Apache/2.4.68 (Debian)
Last-Modified: Tue, 08 Sep 2026 13:47:01 GMT
Content-Length: 363
Content-Type: text/html
```

Cette fois je passe par le nom, donc par le DNS. Le code 200 et le Content-Length de 363 octets, qui est la taille exacte de mon index.html, confirment que c'est ma page. Les deux étapes marchent : le DNS amène le client jusqu'à la machine, l'en-tête Host l'amène jusqu'au bon site.

Le troisième curl, sans -I, renvoie le même contenu que le premier.

Enfin le test au navigateur depuis le poste Windows.



Figure 1 : le site vitrine dans le navigateur Windows, à l'adresse `http://www.bts-sio.lan/`

La page s'affiche avec son titre et ses trois paragraphes. Le navigateur indique « Non sécurisé » à gauche de l'adresse : c'est normal en HTTP, tout circule en clair. L'intranet, lui, passera en HTTPS en partie E.

Ce que montre le journal des accès

```
sudo tail -5 /var/log/apache2/www.bts-sio.lan-access.log
```

```
192.168.100.50 - - [08/Sep/2026:15:49:23 +0200] "HEAD / HTTP/1.1" 200 252
```

```

"-" "curl/7.88.1"
192.168.100.50 - - [08/Sep/2026:15:49:30 +0200] "GET / HTTP/1.1" 200 615
"-" "curl/7.88.1"
192.168.100.1 - - [08/Sep/2026:15:49:55 +0200] "GET / HTTP/1.1" 200 612
"-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/152.0.0.0 Safari/537.36"
192.168.100.1 - - [08/Sep/2026:15:49:55 +0200] "GET /favicon.ico HTTP/1.1" 404 533
"http://www.bts-sio.lan/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) [...]"
192.168.100.1 - - [08/Sep/2026:15:51:22 +0200] "GET / HTTP/1.1" 200 612
"-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) [...]"

```

Les deux origines sont visibles : 192.168.100.50 pour cli-linux, 192.168.100.1 pour le poste Windows. Je décompose ces lignes champ par champ en réponse à la question 12.

Une ligne que je n'avais pas provoquée mérite le détour. Le navigateur a demandé /favicon.ico tout seul, juste après la page, et a reçu un 404. C'est la petite icône affichée dans l'onglet. Je ne l'ai jamais créée, donc le fichier n'existe pas. Ça n'empêche rien de marcher, mais ça montre deux choses. D'abord qu'un navigateur redemande des ressources en plus de la page, alors que curl ne demande que ce qu'on lui dit. Ensuite qu'un 404 dans un journal n'est pas toujours un problème : il faut regarder ce qui a été demandé avant de s'inquiéter.

Partie D. Créer l'intranet et le restreindre

Étape 8. Second site et restriction par adresse IP

Je crée le second site sur le même modèle que la vitrine : un dossier, un public_html, une page d'accueil, et les mêmes droits. La différence tient en une ligne dans l'hôte virtuel. Au lieu de Require all granted, j'écris Require ip 192.168.100.0/24.

Le fichier est recopié en section 5.

```

sio@srv-linux:~$ sudo a2ensite intranet.bts-sio.lan.conf
Enabling site intranet.bts-sio.lan.
sio@srv-linux:~$ sudo apache2ctl configtest && sudo systemctl reload apache2
Syntax OK

```

sudo apache2ctl -S

```

VirtualHost configuration:
*:80 is a NameVirtualHost
    default server intranet.bts-sio.lan (.../intranet.bts-sio.lan.conf:1)
    port 80 namevhost intranet.bts-sio.lan (.../intranet.bts-sio.lan.conf:1)
    port 80 namevhost www.bts-sio.lan (.../www.bts-sio.lan.conf:1)
    alias bts-sio.lan

```

La sortie a changé depuis l'étape 6. Les mentions NameVirtualHost et default server sont apparues, parce qu'il y a maintenant deux sites sur le même port. Deux choses sont à relever.

D'abord l'alias bts-sio.lan figure bien sous la vitrine : c'est mon ServerAlias, et il est pris en compte.

Ensuite le site attrape-tout est intranet.bts-sio.lan, pas la vitrine. Apache prend le premier fichier dans l'ordre alphabétique de sites-enabled, et i vient avant w. Donc une requête sur http://192.168.100.10/ ou avec un Host inconnu tombe sur l'intranet. Ça n'est pas gênant ici puisque l'intranet est protégé, mais il faut le savoir : c'est le site le plus sensible qui répond aux requêtes anonymes. Je reviens sur ce point en réponse à la question 10.

curl -I http://intranet.bts-sio.lan/

```

sio@cli-linux:~$ curl -I http://intranet.bts-sio.lan/
HTTP/1.1 200 OK
Content-Length: 272

```

Le client est en 192.168.100.50, donc dans le réseau autorisé. Il passe. Le Content-Length de 272 octets confirme que c'est bien la page de l'intranet et pas celle de la vitrine, qui fait 363 octets. Ce détail m'a servi plus tard pour repérer une panne.

Tester le refus

Je n'ai pas de machine hors du réseau. Je simule donc le refus en remplaçant la règle par un réseau auquel je n'appartiens pas :

```
Require ip 10.0.0.0/8
sio@cli-linux:~$ curl -I http://intranet.bts-sio.lan/
HTTP/1.1 403 Forbidden
Server: Apache/2.4.68 (Debian)
Content-Type: text/html; charset=iso-8859-1
```

Le code 403 est le bon. Je vais lire ce que le serveur en dit de son côté :

```
sudo grep -i denied /var/log/apache2/intranet-error.log
[Wed Sep 09 08:54:01.176166 2026] [authz_core:error] [pid 1031:tid 1054]
  [client 192.168.100.50:50730] AH01630: client denied by server configuration:
  /var/www/intranet.bts-sio.lan/public_html/
[Wed Sep 09 09:01:52.793331 2026] [authz_core:error] [pid 1108:tid 1111]
  [client 192.168.100.10:41650] AH01630: client denied by server configuration:
  /var/www/intranet.bts-sio.lan/public_html/
```

La ligne donne tout : le module qui refuse (authz_core), l'adresse du client, le code interne AH01630 et le chemin concerné. Je détaille la différence avec un 404 en réponse à la question 13.

Il y a deux lignes parce que la seconde vient d'une faute de ma part, racontée en section « Difficultés rencontrées ». Je remets ensuite la bonne valeur, et le client repasse en 200.

Étape 9. Ajouter l'authentification par mot de passe

Je crée deux comptes dans un fichier placé hors du DocumentRoot, pour qu'il ne soit jamais servi par le web.

```
sio@srv-linux:~$ sudo mkdir -p /etc/apache2/passwd
sio@srv-linux:~$ sudo htpasswd -c /etc/apache2/passwd/.htpasswd-intranet sio
sio@srv-linux:~$ sudo htpasswd /etc/apache2/passwd/.htpasswd-intranet alice

sio@srv-linux:~$ sudo cat /etc/apache2/passwd/.htpasswd-intranet
sio:$apr1$yy6zHTpA$Av5wG7ZkkyPHyzCr0XdC.
alice:$apr1$1irYsDAa$ou4nN0gmSJLtreYz8Wnc0
```

Une ligne par compte. Le mot de passe n'y figure pas : ce qui est stocké est une empreinte, reconnaissable au préfixe \$apr1\$ qui désigne l'algorithme MD5 d'Apache. Les deux empreintes sont différentes alors que les deux comptes ont le même mot de passe, parce que chacune contient un sel tiré au hasard. Sans ça, deux comptes au même mot de passe se repéreraient d'un coup d'oeil.

Le -c n'est présent que sur la première commande. Je détaille pourquoi en réponse à la question 16.

```
sio@srv-linux:~$ sudo chown root:www-data /etc/apache2/passwd/.htpasswd-intranet
sio@srv-linux:~$ sudo chmod 640 /etc/apache2/passwd/.htpasswd-intranet
sio@srv-linux:~$ ls -la /etc/apache2/passwd/
-rw-r----- 1 root www-data 86  9 sept. 09:06 .htpasswd-intranet
```

Le fichier est en 640 : root écrit, www-data lit, les autres n'ont rien. C'est le minimum nécessaire pour qu'Apache puisse vérifier les mots de passe. Il faut le ls avec l'option -a pour le voir, puisque son nom commence par un point.

Je complète ensuite le bloc Directory de l'intranet, avec un bloc RequireAll autour des deux conditions. Sans lui, plusieurs Require sont en OU logique et l'adresse IP suffirait à entrer sans mot de passe. Le fichier est en section 5.

Les trois tests demandés, dans l'ordre :

```
sio@cli-linux:~$ curl -I http://intranet.bts-sio.lan/
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Intranet BTS SIO - acces reserve"

sio@cli-linux:~$ curl -I -u sio:'<mot de passe>' http://intranet.bts-sio.lan/
HTTP/1.1 200 OK
Content-Length: 272

sio@cli-linux:~$ curl -I -u sio:'MauvaisMotDePasse' http://intranet.bts-sio.lan/
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Intranet BTS SIO - acces reserve"
```

401 sans identifiants, 200 avec les bons, 401 de nouveau avec un mauvais mot de passe. Le serveur ne fait pas la différence entre « tu n'as rien envoyé » et « ce que tu as envoyé est faux » : il renvoie le même code et le même en-tête. C'est voulu, ça évite de dire à un attaquant que le compte existe.

Test au navigateur ensuite, en navigation privée pour ne pas réutiliser d'identifiants mémorisés.

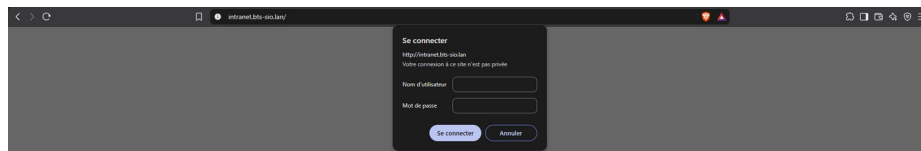


Figure 2 : la boîte de dialogue d'authentification sur <http://intranet.bts-sio.lan/>

La boîte apparaît bien, mais mon AuthName ne s'affiche pas. Ce n'est pas une erreur de configuration : le curl ci-dessus prouve que le serveur envoie bien l'en-tête WWW-Authenticate avec mon texte. C'est le navigateur qui ne l'affiche plus. Les navigateurs à base de Chromium ont retiré cet affichage il y a quelques versions, parce qu'un serveur malveillant pouvait y écrire n'importe quel message pour tromper l'utilisateur. Firefox l'affiche encore.

Je note aussi la ligne « Votre connexion à ce site n'est pas privée » sous l'adresse. Le navigateur prévient qu'on va taper un mot de passe sur une connexion en clair. Il a raison, et c'est exactement ce que la partie suivante corrige.

Partie E. Passer l'intranet en HTTPS

Étape 10. Générer le certificat auto-signé

Voici la commande complétée. Les valeurs manquantes étaient la durée, la taille de clé, le CN et le SAN.

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:4096 \
  -keyout /etc/ssl/private/intranet.bts-sio.lan.key \
  -out /etc/ssl/certs/intranet.bts-sio.lan.crt \
  -subj "/C=FR/ST=France/L=Ville/O=BTS SIO/CN=intranet.bts-sio.lan" \
  -addext "subjectAltName=DNS:intranet.bts-sio.lan"
```

Le CN et le SAN valent exactement le nom que tapera le navigateur. Le SAN est le champ que les navigateurs récents lisent vraiment : sans lui, le certificat serait refusé même avec un CN correct.

```
sio@srv-linux:~$ sudo ls -l /etc/ssl/private/intranet.bts-sio.lan.key
-rw----- 1 root root 3272  9 sept. 09:16 intranet.bts-sio.lan.key

sio@srv-linux:~$ sudo openssl x509 -in /etc/ssl/certs/intranet.bts-sio.lan.crt \
  -noout -subject -dates -ext subjectAltName
```

```
subject=C = FR, ST = France, L = Ville, O = BTS SIO, CN = intranet.bts-sio.lan
notBefore=Sep  9 07:16:31 2026 GMT
notAfter=Sep  9 07:16:31 2027 GMT
X509v3 Subject Alternative Name:
    DNS:intranet.bts-sio.lan
```

Le sujet est correct, le SAN est présent, et l'écart entre les deux dates fait exactement un an. Les heures sont en GMT, deux heures de moins que ma montre, comme le fichier de baux DHCP du TP1. La clé est en 600 et appartient à root : je justifie ce choix en réponse à la question 19.

Étape 11. Activer mod_ssl et écrire l'hôte virtuel 443

```
sio@srv-linux:~$ sudo a2enmod ssl
Considering dependency socache_shmcb for ssl:
Enabling module socache_shmcb.
Enabling module ssl.

sio@srv-linux:~$ grep -n Listen /etc/apache2/ports.conf
5:Listen 80
8:      Listen 443
12:     Listen 443
```

Le module tire une dépendance, socache_shmcb, qui sert de cache pour les sessions TLS. Et le grep montre que le port 443 apparaît deux fois dans ports.conf, à l'intérieur de blocs conditionnels : le port n'est ouvert que si mod_ssl est chargé. C'est pour ça que a2enmod ssl est un préalable obligatoire, et pas une option.

J'écris ensuite l'hôte virtuel du port 443. Il reprend exactement les mêmes restrictions que la partie D : passer en HTTPS ne doit rien enlever à la protection. Le fichier est en section 5.

Puis je remplace tout le contenu de l'hôte virtuel HTTP de l'intranet par une simple redirection. Le port 80 ne sert plus la page, il renvoie vers le HTTPS.

```
<VirtualHost *:80>
    ServerName intranet.bts-sio.lan
    Redirect permanent / https://intranet.bts-sio.lan/
    ErrorLog ${APACHE_LOG_DIR}/intranet-redirect-error.log
</VirtualHost>
```

Cette redirection ne doit surtout pas être écrite aussi dans le bloc 443, sinon le site se redirigerait vers lui-même sans fin.

sudo apache2ctl -S ; sudo ss -tulpn | grep -E ":(80|443)"

```
VirtualHost configuration:
*:443 intranet.bts-sio.lan (.../intranet-ssl.bts-sio.lan.conf:1)
*:80  is a NameVirtualHost
      default server intranet.bts-sio.lan (.../intranet.bts-sio.lan.conf:1)
      port 80 namevhost intranet.bts-sio.lan (.../intranet.bts-sio.lan.conf:1)
      port 80 namevhost www.bts-sio.lan (.../www.bts-sio.lan.conf:1)
Mutex ssl-cache: using_defaults
Mutex ssl-stapling: using_defaults

tcp LISTEN 0 511 *:80  *: * users:(("apache2",pid=1613,fd=4),[...])
tcp LISTEN 0 511 *:443 *: * users:(("apache2",pid=1613,fd=6),[...])
```

Trois hôtes virtuels et deux ports. Les mutex ssl-cache et ssl-stapling n'existaient pas avant : c'est la preuve que mod_ssl est chargé. Le même processus tient les deux ports, avec deux descripteurs différents, fd=4 pour le 80 et fd=6 pour le 443.

Étape 12. Tester en HTTPS depuis le client puis depuis Windows

```
sio@cli-linux:~$ curl -I http://intranet.bts-sio.lan/
HTTP/1.1 301 Moved Permanently
Location: https://intranet.bts-sio.lan/
```

```
sio@cli-linux:~$ curl -k -I https://intranet.bts-sio.lan/
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Intranet BTS SIO - acces reserve"

sio@cli-linux:~$ curl -k -I -u sio:'<mot de passe>' https://intranet.bts-sio.lan/
HTTP/1.1 200 OK
Content-Length: 272

sio@cli-linux:~$ curl -I https://intranet.bts-sio.lan/
curl: (60) SSL certificate problem: self-signed certificate
```

Les quatre résultats sont ceux attendus. Le 301 avec son en-tête Location, le 401 qui prouve que le site répond en HTTPS et exige toujours un mot de passe, le 200 avec les bons identifiants, et l'échec du quatrième faute de l'option -k. Je détaille ce dernier en réponse à la question 20.

Je vérifie aussi que la vitrine n'a pas bougé :

```
sio@cli-linux:~$ curl -I http://www.bts-sio.lan/
HTTP/1.1 200 OK
Content-Length: 363
```

200 direct, sans redirection. Ma redirection ne concerne bien que l'hôte virtuel de l'intranet, pas tout le port 80.

Au navigateur, l'ouverture de `http://intranet.bts-sio.lan/` bascule d'abord en `https://` puis affiche un avertissement.

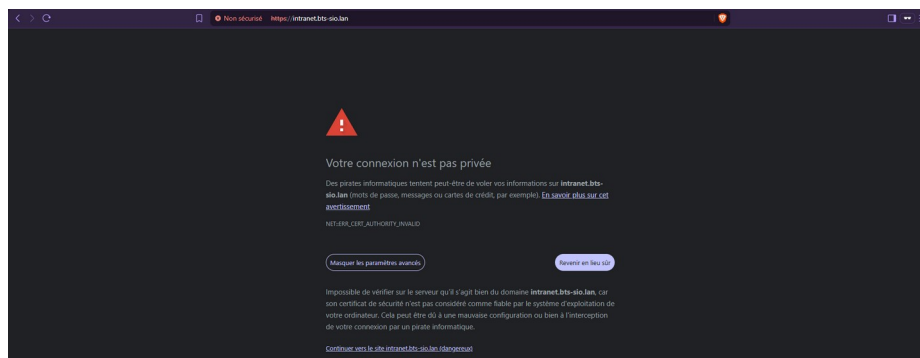


Figure 3 : l'avertissement de sécurité du navigateur, paramètres avancés dépliés

Le texte est précis et vaut d'être lu. Le navigateur dit qu'il ne peut pas vérifier qu'il s'agit bien du domaine `intranet.bts-sio.lan`, parce que le certificat n'est pas considéré comme fiable par le système. Le code `NET::ERR_CERT_AUTHORITY_INVALID` nomme la cause : c'est l'autorité qui pose problème, pas le chiffrement. Je développe en réponse à la question 21.

Après avoir accepté l'exception, la boîte de mot de passe s'affiche.

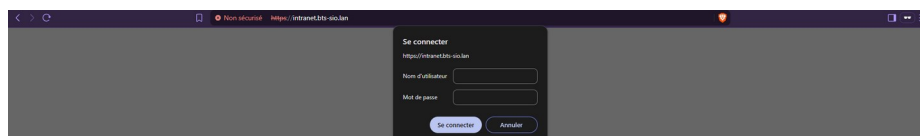


Figure 4 : la demande d'identifiants, cette fois en HTTPS

Une ligne a disparu par rapport à la figure 2 : le navigateur ne prévient plus que la connexion n'est pas privée. Les identifiants sont maintenant chiffrés. Les deux mesures, mot de passe et TLS, n'ont de sens qu'ensemble.

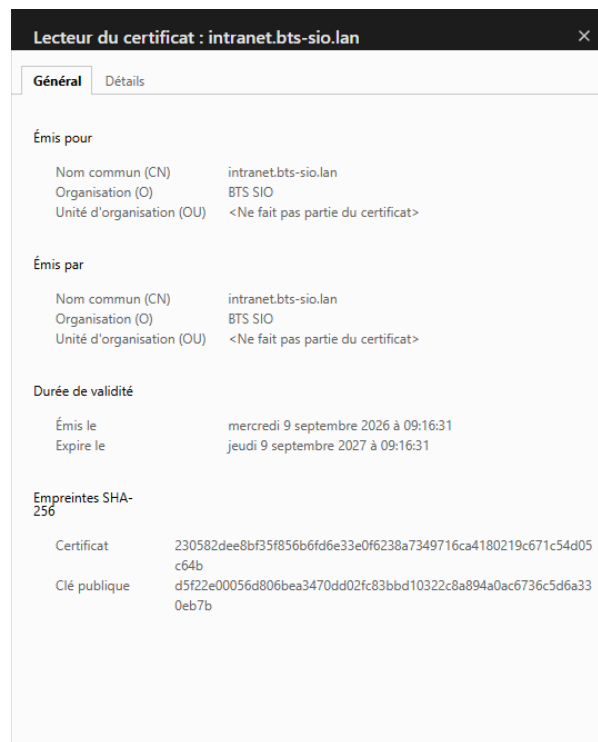


Figure 5 : le détail du certificat, sujet, émetteur et dates de validité

Cette fenêtre montre l'auto-signature d'un coup d'oeil. « Émis pour » et « Émis par » contiennent la même identité : CN intranet.bts-sio.lan, organisation BTS SIO. Le serveur est sa propre autorité. Un certificat normal aurait ici le nom d'une autorité connue, comme Let's Encrypt ou DigiCert, et c'est cette différence, et elle seule, qui déclenche l'avertissement. Les dates confirment la validité d'un an, du 9 septembre 2026 au 9 septembre 2027.

Partie F. Certbot en démonstration

Étape 13. Analyser ce qui manque pour Let's Encrypt

Cette partie ne se met pas en oeuvre, elle s'analyse. Let's Encrypt est une autorité gratuite et automatisée qui délivre des certificats reconnus par tous les navigateurs, valables 90 jours et renouvelés tout seuls. La séquence type est courte :

```
sudo apt install -y certbot python3-certbot-apache
sudo certbot --apache -d intranet.example.fr -m admin@example.fr --agree-tos
sudo certbot certificates
sudo certbot renew --dry-run
systemctl list-timers | grep certbot
```

La commande certbot --apache fait tout d'un coup : elle obtient le certificat, modifie l'hôte virtuel, crée le bloc 443 et met en place la redirection. Le renouvellement est confié à un minuteur systemd installé par le paquet, qui se déclenche deux fois par jour et ne renouvelle que ce qui approche de l'expiration. La commande renew --dry-run simule l'opération sans consommer les quotas de l'autorité.

Rien de tout ça ne peut fonctionner sur ma maquette. J'explique les trois raisons en réponse à la question 23.

Partie G. Déployer une petite application PHP

Étape 14. Installer PHP et publier une page dynamique

```
sio@srv-linux:~$ sudo apt install -y php libapache2-mod-php
sio@srv-linux:~$ php -v
PHP 8.2.33 (cli) (built: Aug 10 2026 16:44:03) (NTS)
Zend Engine v4.2.33, with Zend OPcache v8.2.33

sio@srv-linux:~$ sudo systemctl restart apache2
sio@srv-linux:~$ sudo apache2ctl -V | grep MPM
Server MPM:      prefork
sio@srv-linux:~$ sudo apache2ctl -M | grep php
php_module (shared)
```

Le MPM a changé tout seul. À l'installation d'Apache, en partie B, la sortie annonçait « Enabling module mpm_event ». Il est maintenant en prefork. C'est libapache2-mod-php qui a fait la bascule, parce que l'interpréteur PHP embarqué dans Apache n'est pas prévu pour être partagé entre plusieurs fils d'exécution. Prefork lance un processus par connexion, sans threads, donc le problème ne se pose pas. On y perd en mémoire, c'est le prix à payer pour mod_php.

Deux paquets, deux rôles. php fournit l'interpréteur en ligne de commande, celui que montre php -v. libapache2-mod-php fournit le module qui exécute le PHP dans le serveur web. Sans le second, Apache renverrait le fichier .php en téléchargement.

J'écris ensuite la page infos.php dans le site vitrine, avec les droits habituels, puis je la teste avec et sans paramètre :

```
sio@cli-linux:~$ curl -s http://www.bts-sio.lan/infos.php | grep -i adresse
<li>Votre adresse IP : <strong>192.168.100.50</strong></li>

sio@cli-linux:~$ curl -s 'http://www.bts-sio.lan/infos.php?prenom=Marie' | grep
Bonjour
<p>Bonjour <strong>Marie</strong>, page generee le 09/09/2026 07:37:43.</p>
```

curl reçoit du HTML, jamais de code PHP. C'est le signe que le module fait son travail : le fichier est exécuté sur le serveur, et seul son résultat part sur le réseau. Le paramètre prenom passé dans l'adresse est bien récupéré et réaffiché.



Figure 6 : la page infos.php après envoi du formulaire depuis le poste Windows

Le nom d'hôte affiché est srv-linux, l'heure change à chaque rechargement, et le formulaire renvoie bien le prénom saisi. L'adresse dans la barre contient ?prenom=Dorian : le formulaire est en méthode GET, donc les données passent par l'URL.

L'adresse IP affichée vaut 192.168.100.1 ici, contre 192.168.100.50 dans le curl précédent. C'est la même page, deux clients différents. Je l'explique en réponse à la question 24.

Le test de phpinfo, fait puis défait

```
sio@srv-linux:/var/www/www.bts-sio.lan/public_html$ echo '<?php phpinfo(); ?>' |
sudo tee info.php
<?php phpinfo(); ?>
sio@srv-linux:/var/www/www.bts-sio.lan/public_html$ sudo rm info.php
sio@srv-linux:/var/www/www.bts-sio.lan/public_html$ ls -l
-rw-r--r-- 1 root www-data 363  8 sept. 15:47 index.html
-rw-r--r-- 1 root www-data 836  9 sept. 09:36 infos.php
```

J'ai consulté la page dans le navigateur avant de la supprimer. Elle affiche des pages entières d'informations sur la machine. Je cite trois de ces informations et j'explique le danger en réponse à la question 25. Le ls final montre qu'il ne reste que mes deux fichiers.

Partie H. Introduction à Nginx

Étape 15. Installer Nginx sans casser Apache

J'installe le paquet en sachant que le démarrage va échouer : Apache occupe déjà le port 80, et deux processus ne peuvent pas écouter le même couple adresse et port.

```
sio@srv-linux:~$ systemctl status nginx --no-pager | head -5
× nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; preset: enabled)
   Active: failed (Result: exit-code) since Wed 2026-09-09 09:42:09 CEST; 3s ago
   Process: 8234 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process
on;
          (code=exited, status=0/SUCCESS)
```

sudo journalctl -u nginx -n 10 --no-pager

```
nginx[8235]: nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in
use)
nginx[8235]: nginx: [emerg] bind() to [::]:80 failed (98: Address already in use)
nginx[8235]: nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in
use)
nginx[8235]: nginx: [emerg] bind() to [::]:80 failed (98: Address already in use)
nginx[8235]: nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in
use)
nginx[8235]: nginx: [emerg] bind() to [::]:80 failed (98: Address already in use)
nginx[8235]: nginx: [emerg] still could not bind()
systemd[1]: nginx.service: Control process exited, code=exited, status=1/FAILURE
systemd[1]: Failed to start nginx.service
```

Le journal est très parlant. Nginx a essayé six fois avant d'abandonner, trois tentatives en IPv4 sur 0.0.0.0:80 et trois en IPv6 sur [::]:80. Il réessaie donc avant de renoncer, et il tente les deux familles d'adresses. La ligne « still could not bind() » marque l'abandon.

Une chose est à noter au passage : c'est Nginx qui tombe, pas Apache. Le premier arrivé garde le port, et il n'est même pas prévenu que quelqu'un a essayé. Je détaille en réponse à la question 26.

Je crée le contenu à servir, puis le site sur le port 8080. Le fichier demo.conf est recopié en section 5.

```
sio@srv-linux:~$ sudo rm /etc/nginx/sites-enabled/default
sio@srv-linux:~$ sudo ln -s /etc/nginx/sites-available/demo.conf \
                        /etc/nginx/sites-enabled/demo.conf
sio@srv-linux:~$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
sio@srv-linux:~$ sudo systemctl restart nginx
```

Il n'existe pas de commande `n2ensite` chez Nginx : le lien symbolique se crée à la main. Il faut aussi supprimer le lien du site default, qui écoute sur le port 80 et ferait échouer le démarrage même après ma modification.

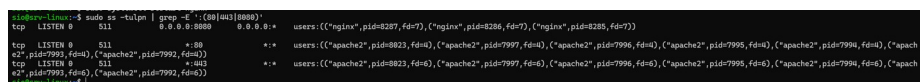


Figure 7 : Apache sur les ports 80 et 443, Nginx sur le port 8080

Les deux serveurs tournent en même temps sur la même machine. Une nuance dans l'affichage : Apache montre `*:80` alors que Nginx montre `0.0.0.0:8080`. Les deux veulent dire « toutes les adresses », mais l'étoile d'Apache couvre IPv4 et IPv6 d'un coup, alors que 0.0.0.0 ne désigne que l'IPv4.

```
sio@cli-linux:~$ curl -s http://srv-linux.bts-sio.lan:8080/
<h1>Nginx sur le port 8080</h1><p>srv-linux - TP2</p>

sio@cli-linux:~$ curl -s http://srv-linux.bts-sio.lan:8080/etat
Nginx operationnel

sio@cli-linux:~$ curl -I http://www.bts-sio.lan/
HTTP/1.1 200 OK
Server: Apache/2.4.68 (Debian)
```

Les trois répondent. La page servie par Nginx, le texte de la route /etat, et le site vitrine toujours servi par Apache. Le port 8080 doit être écrit dans l'adresse, contrairement au 80 qui est le port par défaut du HTTP.

La route /etat est intéressante : elle ne correspond à aucun fichier sur le disque. Nginx fabrique la réponse lui-même avec return 200. C'est le genre de point de contrôle qu'on met en place pour qu'un outil de supervision vérifie que le serveur répond, sans charger de vraie page.

Étape 16. Comparer les deux configurations

Rôle	Directive Apache	Directive Nginx
Déclarer un site	<VirtualHost *:80>	server { listen 8080; }
Nom(s) du site	ServerName et ServerAlias	server_name a b c;
Racine des fichiers	DocumentRoot	root
Page servie pour un répertoire	DirectoryIndex	index
Journal des accès	CustomLog	access_log
Restreindre par adresse IP	Require ip 192.168.100.0/24	allow 192.168.100.0/24; deny all;
Authentification par mot de passe	AuthType Basic et AuthUserFile	auth_basic et auth_basic_user_file
Activer un site	a2ensite	In -s à la main, pas d'outil dédié
Tester la syntaxe	apache2ctl configtest	nginx -t
Configuration par répertoire	.htaccess	n'existe pas

Deux remarques en remplissant ce tableau. Les directives se correspondent presque une pour une, seuls les noms changent : passer d'un serveur à l'autre demande surtout du vocabulaire. Et les deux dernières lignes sont les vraies différences. Je les commente en réponse à la question 27.

Une différence de forme aussi : chez Nginx, chaque directive se termine par un point-virgule. Un oubli est l'erreur la plus fréquente, et nginx -t donne la ligne exacte.

Partie I. Ouvrir le pare-feu

Étape 17. Autoriser SSH, puis 80, 443 et 8080

C'est l'étape qui peut couper l'accès à la machine. J'ai ouvert une console VMware sur srv-linux avant de commencer, et je l'ai laissée ouverte pendant toute l'opération. Si ma session SSH avait sauté, c'était ma seule façon de revenir.

La règle sur le port 22 passe en premier, avant tout le reste et avant l'activation.

```
sio@srv-linux:~$ sudo ufw allow 22/tcp
Rules updated
Rules updated (v6)
```

La commande de l'énoncé pour Apache n'a pas fonctionné :

```
sio@srv-linux:~$ sudo ufw allow 'Apache Full'
ERROR: Could not find a profile matching 'Apache Full'
```

J'ai cherché pourquoi, et l'explication est en section « Difficultés rencontrées ». J'ai ouvert les deux ports directement, ce qui donne le même résultat et se lit plus facilement dans l'état du pare-feu.

```
sio@srv-linux:~$ sudo ufw allow 80/tcp
sio@srv-linux:~$ sudo ufw allow 443/tcp
sio@srv-linux:~$ sudo ufw allow 8080/tcp
sio@srv-linux:~$ sudo ufw allow 53
sio@srv-linux:~$ sudo ufw allow 67/udp
sio@srv-linux:~$ sudo ufw default deny incoming
Default incoming policy changed to 'deny'
sio@srv-linux:~$ sudo ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
```

Les règles 53 et 67 ne concernent pas ce TP mais le précédent. Elles gardent le DNS et le DHCP du jour 2 en état. Sans elles, cli-linux perdrait son adresse et sa résolution de noms au prochain renouvellement de bail, et je passerais la fin de la séance à chercher une panne de serveur web qui n'existerait pas.

Ma session SSH est restée ouverte pendant toute l'opération, parce que le port 22 était autorisé avant.

```
sio@srv-linux:~$ sudo ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To Action From
---
22/tcp ALLOW IN Anywhere
80/tcp ALLOW IN Anywhere
443/tcp ALLOW IN Anywhere
8080/tcp ALLOW IN Anywhere
53 ALLOW IN Anywhere
67/udp ALLOW IN Anywhere
22/tcp (v6) ALLOW IN Anywhere (v6)
80/tcp (v6) ALLOW IN Anywhere (v6)
443/tcp (v6) ALLOW IN Anywhere (v6)
8080/tcp (v6) ALLOW IN Anywhere (v6)
53 (v6) ALLOW IN Anywhere (v6)
67/udp (v6) ALLOW IN Anywhere (v6)
```

Figure 8 : l'état du pare-feu, politique par défaut et six règles

La politique par défaut est deny en entrée et allow en sortie. Chaque règle apparaît deux fois, une version IPv4 et une version (v6). ufw écrit les deux d'office, même si mon montage n'utilise pas l'IPv6.

Vérifier depuis le client

C'est le seul test qui compte vraiment. Un service qui écoute et un pare-feu qui semble bon ne prouvent rien tant que le client n'a pas répondu.

```
sio@cli-linux:~$ curl -I http://www.bts-sio.lan/
HTTP/1.1 200 OK

sio@cli-linux:~$ curl -k -I -u sio:'<mot de passe>' https://intranet.bts-sio.lan/
HTTP/1.1 200 OK

sio@cli-linux:~$ curl -s http://srv-linux.bts-sio.lan:8080/etat
Nginx operationnel

sio@cli-linux:~$ nc -zv 192.168.100.10 80 443 8080
srv-linux.bts-sio.lan [192.168.100.10] 80 (http) open
srv-linux.bts-sio.lan [192.168.100.10] 443 (https) open
srv-linux.bts-sio.lan [192.168.100.10] 8080 (http-alt) open

sio@cli-linux:~$ nc -zv -w 3 192.168.100.10 3306
srv-linux.bts-sio.lan [192.168.100.10] 3306 (mysql) : Connection timed out
```

Les trois ports ouverts répondent tout de suite. Le port 3306 met trois secondes à abandonner, et le message n'est pas le même. Cette différence est la réponse à la question 28.

Dernière vérification, le DHCP du jour 2 :

```
sio@cli-linux:~$ sudo dhclient -r ens33 && sudo dhclient ens33
sio@cli-linux:~$ ip -br a
lo                UNKNOWN          127.0.0.1/8 ::1/128
ens32             DOWN
ens33            UP          192.168.100.50/24 fe80::20c:29ff:feb1:8d39/64
```

Le client a rendu son bail puis en a redemandé un, et il a retrouvé son adresse réservée. La règle 67/udp fait bien son travail.

4. Réponses aux questions

Partie A. Vérifier les prérequis

Q1. Les deux noms *www* et *intranet* sont des enregistrements CNAME. Expliquez ce qu'est un CNAME, pourquoi la sortie de *dig ... +short* contient deux lignes au lieu d'une, et quel serait l'inconvénient d'utiliser deux enregistrements A à la place.

Un CNAME est un alias. Il associe un nom à un autre nom, et jamais à une adresse. Dans ma zone du TP1, *www* et *intranet* pointent tous les deux vers *srv-linux.bts-sio.lan*, qui lui porte un enregistrement A vers 192.168.100.10.

La sortie contient deux lignes parce que le serveur suit l'alias tout seul :

```
sio@cli-linux:~$ dig www.bts-sio.lan +short
srv-linux.bts-sio.lan.
192.168.100.10
```

La première ligne est la cible de l'alias, la seconde son adresse. Répondre seulement le nom cible ne servirait à rien au client, il devrait reposer une question. Le serveur lui donne donc les deux enregistrements d'un coup. L'option *+short* enlève l'habillage et ne garde que les valeurs, d'où ces deux lignes nues.

Avec deux enregistrements A à la place, tout marcherait aussi, mais l'adresse serait écrite trois fois dans le fichier de zone : une fois pour *srv-linux*, une fois pour *www*, une fois pour *intranet*. Le jour où la machine change d'adresse, il faut penser aux trois. Si on en oublie une, un des noms continue de pointer vers le vide, et rien ne le signale. Avec des CNAME, une seule ligne à changer, celle du A de *srv-linux*, et les deux alias suivent automatiquement.

Q2. À quoi sert le fichier *hosts* sous Windows comme sous Linux (*/etc/hosts*) ? Est-il consulté avant ou après le serveur DNS, et où ce comportement est-il défini sous Linux ?

Le fichier *hosts* fait le même travail qu'un serveur DNS mais en local : il associe des adresses IP à des noms, une par ligne. Il servait à ça avant l'invention du DNS, quand tout Internet tenait dans un seul fichier partagé. Aujourd'hui il sert aux cas particuliers : une machine que le DNS ne connaît pas, un test avant mise en production, ou comme ici un poste qui n'utilise pas mon serveur.

Il est consulté avant le serveur DNS, et sa réponse gagne. Sous Linux, ce comportement est écrit dans */etc/nsswitch.conf*, à la ligne « *hosts: files dns* ». L'ordre des deux mots est l'ordre de consultation : *files* pour */etc/hosts*, *dns* ensuite. Inverser les deux mots inverserait la priorité.

J'ai eu la démonstration sous les yeux. Après avoir rempli le fichier *hosts* de Windows, *ping* fonctionne mais *nslookup* échoue toujours :

```
PS C:\Users\doria> ping www.bts-sio.lan
Envoi d'une requete 'ping' sur www.bts-sio.lan [192.168.100.10]
```

```
Reponse de 192.168.100.10 : octets=32 temps<1ms TTL=64
```

```
PS C:\Users\doria> nslookup www.bts-sio.lan
Serveur : dns.google
*** dns.google ne parvient pas a trouver www.bts-sio.lan : Non-existent domain
```

Les deux commandes ne prennent pas le même chemin. ping demande au système de traduire le nom, donc passe par le résolveur, qui lit le fichier hosts en premier. nslookup est un outil de diagnostic du DNS : il parle directement au serveur et ne lit jamais le fichier hosts. C'est la même leçon qu'au TP1, où dig et ping ne racontaient pas la même chose. Quand deux outils ne sont pas d'accord sur un nom, il faut regarder le fichier hosts.

Q3. Citez deux inconvénients concrets d'utiliser le fichier hosts plutôt qu'un serveur DNS pour un parc de 30 postes.

Premier inconvénient : chaque modification doit être faite trente fois. Ajouter un serveur, changer une adresse, retirer une machine, c'est trente fichiers à éditer, chacun avec les droits administrateur. En pratique, personne ne le fait vraiment, et les fichiers finissent différents d'un poste à l'autre. On obtient alors la pire panne qui soit : ça marche sur certains postes et pas sur d'autres, sans raison visible.

Deuxième inconvénient : aucune durée de vie. Une réponse DNS porte un TTL, donc elle finit par expirer et être redemandée. Une ligne de fichier hosts reste là pour toujours. Une adresse périmée continue d'être utilisée des mois après le changement, et elle passe avant le DNS qui, lui, a la bonne valeur. Le fichier hosts devient alors la cause de la panne, et c'est le dernier endroit où on pense à regarder.

Un serveur DNS règle les deux d'un coup : une seule modification, sur une seule machine, et tous les postes la voient au bout du TTL.

Partie B. Installer Apache

Q4. Dans la sortie de `ss -tulpn`, que signifie exactement `*:80` ? Qu'aurait signifié `127.0.0.1:80`, et quelle conséquence cela aurait-il eu pour vos tests depuis `cli-linux` ?

L'étoile désigne toutes les adresses de la machine. Apache accepte donc les connexions sur le port 80 quelle que soit la carte par laquelle elles arrivent : la boucle locale, `ens33` en `192.168.100.10`, et même `ens32` côté NAT.

`127.0.0.1:80` aurait voulu dire tout autre chose : n'écouter que sur la boucle locale. Le service n'aurait alors été joignable que depuis la machine elle-même. Mon `curl -I http://localhost/` sur `srv-linux` aurait marché exactement pareil, ce qui est le piège. Mais depuis `cli-linux` j'aurais eu un « Connection refused » immédiat, et aucun de mes deux sites n'aurait été accessible.

C'est un cas de panne classique et déroutant : le serveur répond parfaitement quand on teste dessus, et il est injoignable de partout ailleurs. On soupçonne le réseau ou le pare-feu, alors que le problème est dans l'adresse d'écoute. La commande qui tranche est justement `ss -tulpn` : il faut lire ce qu'il y a avant les deux points, pas seulement le numéro de port.

Écouter seulement en local a un usage réel. C'est ce qu'on fait pour un service applicatif placé derrière un reverse proxy : il ne doit être joignable que par le proxy, pas depuis le réseau.

Q5. Comparez `ls -l` sur `sites-available/` et sur `sites-enabled/`. Quelle différence de nature observez-vous entre les entrées des deux répertoires, et quel est l'intérêt de ce mécanisme par rapport à une simple copie de fichier ?

```
/etc/apache2/sites-available/:
-rw-r--r-- 1 root root 1286 000-default.conf
-rw-r--r-- 1 root root 6195 default-ssl.conf
```

```
/etc/apache2/sites-enabled/:  
lrwxrwxrwx 1 root root 35 000-default.conf -> ../sites-available/000-default.conf
```

La différence est visible à la première lettre. Dans available, le tiret indique un fichier ordinaire. Dans enabled, le l indique un lien symbolique, et la flèche montre vers quoi il pointe. Il n'y a donc qu'un seul fichier réel par site, dans available. Le dossier enabled ne contient que des renvois.

L'intérêt par rapport à une copie tient en trois points.

Il n'y a qu'une seule version du fichier. Avec deux copies, on modifie forcément la mauvaise un jour, et on cherche pendant vingt minutes pourquoi la modification n'a aucun effet.

Désactiver ne détruit rien. a2dissite supprime le lien, le fichier reste dans available avec toute sa configuration. Je l'ai fait pour 000-default : il est inactif mais intact, et le réactiver serait une seule commande.

L'état du serveur se lit d'un coup d'oeil. Un ls sur enabled donne la liste exacte des sites actifs. Avec des copies, il faudrait comparer les fichiers deux à deux pour savoir lesquels sont vraiment en service.

Q6. Dans la sortie de `ps aux | grep apache2`, combien de processus tournent sous root et combien sous www-data ? Expliquez pourquoi ce n'est pas la même chose, et à quoi sert précisément le processus root.

```
root      2024  /usr/sbin/apache2 -k start  
www-data  2025  /usr/sbin/apache2 -k start  
www-data  2026  /usr/sbin/apache2 -k start  
www-data  2115  /usr/bin/htcacheclean -d 120
```

Un seul processus tourne sous root, et deux sous www-data. Le quatrième, htcacheclean, tourne aussi sous www-data mais n'est pas un serveur : c'est l'utilitaire qui nettoie le cache disque.

Le processus root ne traite aucune requête. Il fait deux choses au démarrage. Il ouvre les ports 80 et 443, ce que seul root peut faire puisque les ports en dessous de 1024 sont réservés. Puis il crée les processus enfants et leur transmet les ports déjà ouverts, avant que ces enfants abandonnent leurs privilèges pour devenir www-data. Ensuite le processus root se contente de surveiller ses enfants et d'en relancer si besoin.

On voit ce partage dans `ss -tulpn` : les trois processus partagent le même descripteur de fichier, `fd=4`. C'est le même port ouvert une seule fois, par root, et hérité par les autres.

Ce n'est pas la même chose parce que ce sont les processus www-data qui reçoivent les données venant du réseau, donc ceux qui sont exposés. Une faille dans le traitement d'une requête ne donne que les droits de www-data, un compte sans mot de passe et sans shell de connexion, qui peut lire les pages mais pas les modifier. Si tout tournait en root, la même faille donnerait la machine entière.

Q7. Relevez les lignes `APACHE_RUN_USER`, `APACHE_RUN_GROUP` et `APACHE_LOG_DIR` de envvars. Où retrouverez-vous la variable `${APACHE_LOG_DIR}` par la suite, et pourquoi Debian passe-t-elle par une variable plutôt que d'écrire le chemin en dur ?

```
export APACHE_RUN_USER=www-data  
export APACHE_RUN_GROUP=www-data  
export APACHE_LOG_DIR=/var/log/apache2$SUFFIX
```

Je retrouve `${APACHE_LOG_DIR}` dans chacun de mes hôtes virtuels, sur les lignes `ErrorLog` et `CustomLog`. Par exemple `ErrorLog ${APACHE_LOG_DIR}/intranet-error.log`, qui désigne en réalité `/var/log/apache2/intranet-error.log`.

Debian passe par une variable pour deux raisons.

La première se lit dans la valeur elle-même : `/var/log/apache2$SUFFIX`. Il y a un second niveau de variable. Ce suffixe permet de faire tourner plusieurs instances d'Apache sur la même machine, chacune avec ses propres journaux, sans réécrire un seul fichier de site. Les mêmes fichiers de configuration servent alors pour toutes les instances.

La seconde est plus terre à terre. Le chemin des journaux n'est pas le même sur toutes les distributions. Un fichier de site écrit avec la variable se recopie d'une machine à l'autre sans modification. Un chemin en dur devrait être corrigé partout, dans chaque fichier de chaque site, le jour où il change.

Partie C. Le site public

Q8. Pourquoi les répertoires sont-ils en 755 et le fichier en 644 ? Que se passerait-il si le répertoire `public_html` était en 644, et pourquoi, alors que le fichier `index.html`, lui, resterait parfaitement lisible ?

Les deux valeurs ne diffèrent que par le bit `x`, et ce bit ne veut pas dire la même chose selon qu'on parle d'un fichier ou d'un dossier.

Sur un fichier, `x` veut dire « exécutable ». Une page HTML n'est pas un programme, elle n'a aucune raison de l'être. D'où 644 : root écrit, tout le monde lit.

Sur un dossier, `x` veut dire « traverser ». C'est le droit d'entrer dedans pour atteindre ce qu'il contient. Le droit `r`, lui, sert seulement à lister les noms des fichiers. Ce sont deux choses différentes, et c'est toute la clé de la question.

Si `public_html` était en 644, il perdrait le bit `x`. Apache pourrait lire la liste des fichiers, donc savoir que `index.html` existe, mais ne pourrait pas entrer dans le dossier pour l'ouvrir. Le fichier resterait pourtant en 644, parfaitement lisible : ses propres droits ne changent pas. Simplement, personne ne peut plus l'atteindre. C'est comme un livre en libre accès dans une pièce fermée à clé.

Le résultat serait un 403 Forbidden sur une page qui existe, avec des droits corrects, ce qui est très déroutant. La commande qui le montre est `namei -l`, qui affiche les permissions de chaque niveau du chemin. C'est aussi pour ça que 755 s'applique à tous les dossiers parents : il suffit qu'un seul du chemin perde son `x` pour bloquer tout ce qu'il y a en dessous.

Q9. Que fait précisément `a2ensite` ? Donnez la commande `ln -s` exacte qui produirait le même résultat, et vérifiez-la avec `ls -l /etc/apache2/sites-enabled/`.

`a2ensite` crée un lien symbolique dans `sites-enabled` qui pointe vers le fichier de `sites-available`. Il ne copie rien, ne modifie aucune configuration et ne recharge pas Apache. Il le dit lui-même, en rappelant qu'il faut faire le reload à la main.

La commande équivalente est :

```
sudo ln -s /etc/apache2/sites-available/www.bts-sio.lan.conf \
/etc/apache2/sites-enabled/www.bts-sio.lan.conf
```

La vérification confirme ce qu'a fait la commande :

```
sio@srv-linux:~$ ls -l /etc/apache2/sites-enabled/
lrwxrwxrwx 1 root root 39 www.bts-sio.lan.conf ->
../sites-available/www.bts-sio.lan.conf
```

Une nuance sur le chemin. `a2ensite` écrit un lien relatif, `../sites-available/...`, et non un chemin absolu. C'est plus robuste : si tout l'arbre `/etc/apache2` était déplacé ou copié ailleurs, le lien continuerait de pointer au bon endroit. J'ai d'ailleurs créé le lien à la main pour Nginx en partie H, où aucune commande équivalente n'existe.

a2ensite apporte quand même deux choses par rapport au `ln -s`. Il vérifie que le fichier existe et refuse avec un message clair sinon, alors que `ln -s` créerait sans broncher un lien vers un fichier absent. Et il accepte le nom avec ou sans l'extension `.conf`.

Q10. Pourquoi désactive-t-on `000-default.conf` ? Que se passerait-il si on le laissait actif et que votre `ServerName` comportait une faute de frappe ? Aidez-vous de la ligne `default server` dans la sortie de `apache2ctl -S`.

Quand Apache reçoit une requête dont l'en-tête `Host` ne correspond à aucun `ServerName` ni `ServerAlias`, il ne renvoie pas d'erreur. Il sert le premier hôte virtuel du port, dans l'ordre alphabétique des fichiers de `sites-enabled`. C'est le site attrape-tout, et c'est lui que la sortie `apache2ctl -S` appelle `default server`.

Le fichier de Debian s'appelle `000-default.conf` pour cette raison : ses trois zéros le placent en tête de l'ordre alphabétique, donc il capte tout ce qui n'est reconnu par personne.

Avec une faute de frappe dans mon `ServerName`, disons `www.bts-sio.lan`, mon site ne correspondrait plus à rien. La requête tomberait sur `000-default` et j'obtiendrais la page Apache par défaut de Debian. Le vrai problème est que rien n'indique la cause : pas d'erreur, pas de ligne dans `error.log`, juste une page qui n'est pas la mienne. On soupçonne le DNS, le reload oublié, le `DocumentRoot`, et on cherche longtemps.

En le désactivant, cette erreur silencieuse devient visible : la requête tombe sur mon propre site, ou sur l'autre, et l'incohérence saute aux yeux. La commande qui tranche reste `apache2ctl -S`, qui montre exactement ce qu'Apache a compris.

Ma propre sortie illustre bien le mécanisme :

```
*:80 is a NameVirtualHost
default server intranet.bts-sio.lan (.../intranet.bts-sio.lan.conf:1)
port 80 namevhost intranet.bts-sio.lan (.../intranet.bts-sio.lan.conf:1)
port 80 namevhost www.bts-sio.lan (.../www.bts-sio.lan.conf:1)
```

Après désactivation de `000-default`, c'est `intranet.bts-sio.lan` qui est devenu le site par défaut, parce que `i` vient avant `w`. Une requête sur `http://192.168.100.10/` tombe donc sur mon site le plus sensible. Ici il est protégé par mot de passe, donc le risque est nul. Mais ça montre que le site attrape-tout n'est pas un choix : c'est une conséquence de l'ordre alphabétique. En production, on garde souvent un attrape-tout volontaire qui répond une page neutre, pour qu'un balayage d'adresses IP ne révèle aucun site réel.

Q11. Pourquoi la première commande fonctionne-t-elle même si le DNS est en panne ? Quelle information l'option `-H "Host: ..."` fournit-elle au serveur, et quelle étape du processus normal remplace-t-elle ?

Parce qu'elle ne demande aucune traduction de nom. L'adresse dans l'URL est déjà une adresse IP :

```
curl -s -H "Host: www.bts-sio.lan" http://192.168.100.10/
```

`curl` ouvre directement une connexion TCP vers `192.168.100.10` sur le port 80. Le DNS n'est jamais interrogé, donc son état n'a aucune importance.

L'option `-H` fournit l'en-tête `Host`, c'est à dire le nom du site demandé. C'est cet en-tête, et lui seul, qui permet à Apache de choisir entre mes deux sites, puisqu'ils partagent la même adresse et le même port. Sans lui, le serveur reçoit une connexion mais ne sait pas quel site est voulu.

Elle remplace deux étapes du processus normal. D'abord la résolution du nom en adresse, que fait ici l'utilisateur en écrivant l'adresse. Ensuite la fabrication automatique de l'en-tête `Host` par le client, qui d'habitude y recopie le nom tapé dans l'URL. Ici on écrit l'en-tête à la main, ce qui permet de mentir au serveur et de demander n'importe quel site sur n'importe quelle adresse.

C'est la commande la plus utile de la journée pour déboguer. Le DNS amène le client jusqu'à la machine, l'en-tête Host l'amène jusqu'au site. En forçant la seconde étape, on teste Apache seul. Si ce curl marche et que celui par le nom échoue, le problème est dans le DNS. S'il échoue aussi, il est dans Apache. Le problème est coupé en deux.

Q12. Prenez une ligne d'`access.log` produite par le navigateur Windows et décomposez-la champ par champ (au moins sept champs). Quelle est la différence entre les deux lignes relevées, et à quoi voyez-vous laquelle vient de curl ?

La ligne du navigateur Windows :

```
192.168.100.1 - - [08/Sep/2026:15:49:55 +0200] "GET / HTTP/1.1" 200 612
 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/152.0.0.0 Safari/537.36"
```

Champ	Valeur	Signification
%h	192.168.100.1	Adresse IP du client, ici le poste Windows
%l	-	Identité identd, plus jamais utilisée, toujours un tiret
%u	-	Utilisateur authentifié, un tiret car la vitrine est publique
%t	[08/Sep/2026:15:49:55 +0200]	Date, heure et décalage horaire
%"r"	"GET / HTTP/1.1"	La ligne de requête : méthode, cible, version
%>s	200	Code de statut renvoyé
%O	612	Taille de la réponse en octets, en-têtes compris
%"[Referer]i"	"-"	Page d'où venait le lien suivi, vide ici
%"[User-Agent]i"	"Mozilla/5.0 ... Chrome/152"	Le logiciel client

Deux différences entre les deux lignes relevées. La plus visible est le dernier champ : curl/7.88.1 d'un côté, la longue chaîne Mozilla de l'autre. C'est le moyen le plus simple de savoir laquelle vient de curl.

La seconde est la méthode. curl -I envoie un HEAD, donc la ligne de requête porte HEAD et pas GET. Sans l'option -I, curl fait un GET comme le navigateur. La taille change aussi : 252 octets pour le HEAD, qui ne renvoie que les en-têtes, contre 612 pour un GET complet.

Le navigateur a aussi produit une ligne que je n'avais pas demandée :

```
192.168.100.1 - - [08/Sep/2026:15:49:55 +0200] "GET /favicon.ico HTTP/1.1" 404 533
"http://www.bts-sio.lan/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) [...]"
```

C'est la petite icône de l'onglet, que le navigateur va chercher tout seul après la page. Je ne l'ai jamais créée, d'où le 404. Cette ligne est la seule dont le champ Referer n'est pas vide : elle porte l'adresse de la page depuis laquelle la demande a été faite. Ça montre deux choses. Un navigateur demande bien plus qu'une page, alors que curl ne demande que ce qu'on lui dit. Et un 404 dans un journal n'est pas toujours un problème : il faut d'abord regarder ce qui a été demandé.

Partie D. Restreindre l'intranet

Q13. Quel code HTTP obtenez-vous quand l'adresse est refusée ? Relevez la ligne correspondante dans `error.log` et expliquez pourquoi ce n'est pas un 404.

J'obtiens un 403 Forbidden :

```
sio@cli-linux:~$ curl -I http://intranet.bts-sio.lan/
HTTP/1.1 403 Forbidden
```

La ligne correspondante dans le journal :

```
[Wed Sep 09 08:54:01.176166 2026] [authz_core:error] [pid 1031:tid 1054]
```

```
[client 192.168.100.50:50730] AH01630: client denied by server configuration:
/var/www/intranet.bts-sio.lan/public_html/
```

Ce n'est pas un 404 parce que les deux codes ne disent pas la même chose. 404 veut dire « ce que vous demandez n'existe pas ». 403 veut dire « ça existe, mais je refuse de vous le donner ». Ici la page existe, Apache a trouvé le bon hôte virtuel, il a trouvé le fichier, et il a même identifié le répertoire concerné, comme le montre la fin du message. Il refuse ensuite, à cause d'une règle que j'ai écrite. Répondre 404 serait un mensonge.

Le message le dit précisément : le refus vient de la configuration du serveur, et le module qui le prononce est `authz_core`, celui qui traite les directives `Require`. Ce n'est ni un problème de fichier, ni de permissions Unix.

À noter qu'il existe un raisonnement inverse, utilisé en sécurité : répondre 404 au lieu de 403 pour ne pas révéler l'existence d'une ressource. Un 403 confirme qu'il y a quelque chose derrière, ce qui est déjà une information pour quelqu'un qui cherche.

Q14. *Le filtrage `Require ip` suffit-il à sécuriser un intranet ? Citez deux situations où il est contourné ou inopérant.*

Non, il ne suffit pas. C'est une barrière utile mais faible, parce qu'elle repose sur l'adresse source d'un paquet, une information que le serveur ne peut pas vraiment vérifier.

Première situation, un reverse proxy placé devant le serveur. Toutes les requêtes semblent alors venir du proxy, et le filtrage devient inopérant : soit il laisse tout passer, soit il bloque tout le monde. Il faut travailler sur l'en-tête `X-Forwarded-For`, avec prudence puisque cet en-tête est fourni par le client et donc falsifiable.

Deuxième situation, une machine du réseau autorisé qui est compromise, ou simplement un poste étudiant sur lequel n'importe qui s'assoit. La règle autorise tout le réseau 192.168.100.0/24 sans distinguer qui utilise la machine. Un VPN ou un accès Wi-Fi mal configuré donnant une adresse de ce réseau contourne la protection de la même façon.

C'est justement pour ça que le TP ajoute un mot de passe juste après, dans un bloc `RequireAll` : l'adresse dit d'où vient la demande, le mot de passe dit qui la fait. Il faut les deux. Et comme le mot de passe circule en clair en HTTP, il faut ensuite le chiffrage, ce que fait la partie E. Les trois mesures se complètent, aucune ne suffit seule.

Q15. *Que renvoie le serveur sans identifiants ? Relevez l'en-tête complet `WWW-Authenticate` de la réponse et expliquez à quoi il sert pour le navigateur.*

Le serveur renvoie un 401 Unauthorized, accompagné d'un en-tête :

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="Intranet BTS SIO - acces reserve"
```

Ce 401 n'est pas un refus définitif, c'est une invitation. Le serveur dit « il me faut des identifiants », et l'en-tête explique lesquels et comment les envoyer.

Il porte deux informations. Le mot `Basic` donne la méthode d'authentification à utiliser, ici HTTP Basic. Le `realm` est un libellé, celui que j'ai écrit dans la directive `AuthName`. Il sert à deux choses pour le navigateur : l'afficher dans la boîte de dialogue pour que l'utilisateur sache où il s'authentifie, et regrouper les pages protégées. Toutes les adresses partageant le même `realm` sur le même serveur réutilisent les mêmes identifiants, ce qui évite de redemander le mot de passe à chaque image et à chaque feuille de style.

Sans cet en-tête, le navigateur recevrait un 401 sans savoir quoi faire et afficherait une page d'erreur. C'est lui qui déclenche la boîte de dialogue.

Un point relevé pendant les tests : mon navigateur n'affiche plus le realm dans la boîte, alors que le curl ci-dessus prouve que le serveur l'envoie bien. Les navigateurs à base de Chromium ont retiré cet affichage, parce qu'un serveur malveillant pouvait y écrire un message trompeur du genre « entrez votre mot de passe Google ». Firefox l'affiche encore.

Q16. Vous avez lancé `htpasswd -c` une seule fois, pour sio. Que se serait-il passé si vous aviez remis `-c` pour alice ? Justifiez avec le contenu du fichier.

L'option `-c` veut dire create. Elle ne crée pas un compte, elle crée le fichier, et elle écrase celui qui existe déjà sans aucun avertissement.

Avec `-c` sur la seconde commande, le fichier aurait été recréé de zéro et ne contiendrait plus qu'une ligne, celle d'alice. Le compte sio aurait disparu. Mon fichier contient bien deux lignes, une par compte :

```
sio:$apr1$yy6zHTpA$Av5wG7ZkkyPHyzCr0XdC.  
alice:$apr1$1irYsDAa$ouZ4nN0gmSJLtreYz8Wnc0
```

Le plus gênant est que rien ne signalerait la perte. La commande se serait terminée normalement, alice fonctionnerait parfaitement, et le problème n'apparaîtrait qu'au moment où sio essaierait de se connecter, peut-être des jours plus tard. Sur un intranet avec vingt comptes ajoutés au fil du temps, un `-c` de trop les efface tous d'un coup.

La bonne habitude est donc de contrôler le fichier après chaque ajout, avec un `cat` ou un `wc -l`. Une ligne par utilisateur, toujours.

Le contenu montre aussi autre chose. Les deux comptes ont le même mot de passe, et pourtant les deux empreintes sont totalement différentes. C'est le sel, la partie qui suit `$apr1$`, tiré au hasard pour chaque compte. Sans lui, deux comptes au même mot de passe se repéreraient d'un coup d'oeil.

Q17. L'en-tête envoyé par le client est `Authorization: Basic c2lvOINpbzlwMjYh`. Décodez-le avec `echo 'c2lvOINpbzlwMjYh' | base64 -d` et concluez sur le niveau de protection réel de cette authentification tant que le site reste en HTTP.

La commande renvoie immédiatement l'identifiant et le mot de passe en clair, séparés par deux points. Je ne les recopie pas ici, mais l'opération prend une fraction de seconde et ne demande aucun outil particulier.

C'est tout le problème. Le base64 n'est pas un chiffrement : c'est une transcription, publique, réversible et sans clé. Elle sert à faire passer n'importe quels caractères dans un en-tête HTTP, pas à cacher quoi que ce soit. La commande `base64 -d` est livrée avec tous les systèmes.

Tant que le site reste en HTTP, la protection réelle est donc nulle contre quelqu'un qui écoute le réseau. Et c'est pire que ça pour deux raisons.

L'en-tête est renvoyé à chaque requête, pas seulement à la connexion. Chaque image, chaque feuille de style, chaque rechargement le transporte de nouveau. Il y a donc des dizaines d'occasions de le capturer plutôt qu'une seule.

Et rien n'avertit l'utilisateur ni l'administrateur. Le site fonctionne, la boîte de mot de passe donne un sentiment de sécurité, et le mot de passe traverse le réseau en clair en boucle. N'importe qui sur le même segment, avec `tcpdump` ou `Wireshark`, le récupère en quelques secondes.

D'où la règle : pas d'authentification sans chiffrement. Mettre un mot de passe sur un site en HTTP ne protège pas, ça donne juste l'illusion de protéger. C'est exactement pour ça que la partie E fait passer l'intranet en HTTPS juste après l'avoir protégé. Les deux mesures n'ont de sens qu'ensemble.

Partie E. Le HTTPS

Q18. À quoi sert l'option `-nodes` ? Que se passerait-il concrètement, au prochain redémarrage de `srv-linux`, si vous l'aviez omise ?

L'option `-nodes` veut dire « no DES ». Elle demande à `openssl` de ne pas chiffrer la clé privée avec une phrase de passe. La clé est donc écrite en clair sur le disque.

Sans elle, `openssl` aurait demandé une phrase de passe au moment de la génération, et la clé aurait été chiffrée avec. Apache aurait alors besoin de cette phrase à chaque démarrage pour pouvoir lire la clé.

Concrètement, au prochain redémarrage de `srv-linux`, Apache ne repartirait pas. Le service resterait bloqué à attendre une phrase de passe que personne ne taperait, puisque le démarrage se fait tout seul, sans humain devant l'écran. La machine redémarrerait après une coupure de courant ou une mise à jour du noyau, et les deux sites resteraient hors service jusqu'à ce que quelqu'un ouvre une console et tape la phrase à la main. Pour un serveur, c'est inacceptable.

La contrepartie est que la clé est en clair sur le disque. C'est justement pour ça qu'elle est en 600 et qu'elle appartient à `root` : puisqu'elle n'est plus protégée par une phrase de passe, elle doit l'être par les permissions du système de fichiers.

Q19. Pourquoi la clé privée est-elle en 600 et le certificat en 644 ? Apache tourne sous `www-data`, qui ne peut donc pas lire la clé : pourquoi le service démarre-t-il quand même ?

Les deux fichiers n'ont pas du tout le même statut.

La clé privée est le secret du serveur. C'est elle qui prouve qu'il est bien celui qu'il prétend être. Quelqu'un qui la vole peut se faire passer pour mon intranet et déchiffrer les échanges. Elle ne quitte jamais la machine et n'est jamais envoyée à personne. D'où le 600 : seul `root` la lit, et `root` seul.

Le certificat, lui, est public par nature. Il est envoyé à tous les clients qui se connectent, c'est le principe même du TLS. Le mettre en 644 ne révèle rien, il est déjà donné à qui le demande.

Le service démarre quand même à cause du partage des rôles vu en question 6. Le processus maître d'Apache démarre en `root`, et c'est lui qui lit la clé, au tout début, en même temps qu'il ouvre les ports privilégiés. Il charge la clé en mémoire. Ensuite seulement il crée les processus enfants, qui abandonnent leurs privilèges et deviennent `www-data`.

Les processus `www-data` n'ont donc jamais besoin d'ouvrir le fichier : ils travaillent avec une clé déjà chargée. C'est la même mécanique que pour le port 80, que `root` ouvre avant de céder la place. La lecture du fichier se fait une fois, en `root`, avant la perte de privilèges. Une faille dans le traitement d'une requête, qui donnerait les droits de `www-data`, ne permettrait pas de lire le fichier de la clé sur le disque.

Q20. Pourquoi le quatrième `curl`, sans `-k`, échoue-t-il ? Recopiez le message exact et expliquez ce que `-k` demande à `curl` de ne pas faire.

```
sio@cli-linux:~$ curl -I https://intranet.bts-sio.lan/
curl: (60) SSL certificate problem: self-signed certificate
More details here: https://curl.se/docs/sslcerts.html
```

```
curl failed to verify the legitimacy of the server and therefore could not
establish a secure connection to it.
```

Il échoue parce que `curl` fait par défaut ce que fait un navigateur : il vérifie le certificat présenté par le serveur. Il regarde qui l'a signé, et remonte la chaîne jusqu'à une autorité qu'il connaît, à partir du magasin livré avec le système, le paquet `ca-certificates` sur Debian.

Ici la chaîne s'arrête tout de suite. Mon certificat est signé par lui-même, donc l'autorité au bout de la chaîne est mon propre serveur, qui ne figure dans le magasin d'aucune machine. curl ne trouve aucun point d'ancrage de confiance et refuse la connexion. Le message le dit clairement : il n'a pas pu vérifier la légitimité du serveur.

L'option -k, pour insecure, demande à curl de ne pas faire cette vérification d'identité. Attention à ce qu'elle ne fait pas : elle ne désactive pas le chiffrement. La connexion reste chiffrée exactement comme avant, avec les mêmes algorithmes. Ce que -k abandonne, c'est la troisième garantie du TLS, l'authentification du serveur. On accepte de parler à quelqu'un sans avoir vérifié qui il est.

C'est acceptable en test sur un réseau fermé, où je sais que le serveur en face est le mien. Ça ne l'est pas ailleurs : sans vérification d'identité, on peut établir un tunnel parfaitement chiffré avec un attaquant qui s'est interposé, et ne rien voir.

Q21. *Le navigateur affiche un avertissement de sécurité. Le chiffrement est-il pour autant plus faible qu'avec un certificat payant ? Expliquez précisément ce que le navigateur ne parvient pas à vérifier, et pourquoi ce comportement est correct.*

Non, le chiffrement est exactement le même. Ma clé RSA fait 4096 bits, et mon hôte virtuel n'accepte que TLS 1.2 et TLS 1.3. Un certificat payant ne changerait pas une ligne de tout ça. Les algorithmes de chiffrement sont négociés entre le client et le serveur, et n'ont aucun rapport avec l'identité de celui qui a signé le certificat.

Ce que le navigateur ne parvient pas à vérifier, c'est l'identité du serveur. Il le dit dans le texte de l'avertissement : il ne peut pas vérifier qu'il s'agit bien du domaine intranet.bts-sio.lan, parce que le certificat n'est pas considéré comme fiable par le système. Le code d'erreur nomme la cause, NET::ERR_CERT_AUTHORITY_INVALID : c'est l'autorité qui pose problème.

Un certificat est une identité signée par un tiers de confiance. Le navigateur remonte la chaîne des signatures jusqu'à une autorité racine pré-installée dans son magasin. Ici la chaîne fait un seul maillon et se referme sur elle-même : je suis ma propre autorité. La figure 5 le montre bien, « Émis pour » et « Émis par » contiennent la même identité.

Ce comportement est correct, et c'est le point important. Le navigateur ne peut pas faire la différence entre mon certificat, qui est légitime, et celui qu'un attaquant fabriquerait en trente secondes pour se faire passer pour mon serveur. Les deux sont auto-signés, les deux portent le bon nom, rien ne les distingue techniquement. La seule chose qui prouve qu'un certificat appartient à qui il prétend, c'est la signature d'une autorité qui a vérifié. Sans elle, il n'y a plus rien à vérifier, et se taire serait la mauvaise réponse.

Un cadenas ne dit donc jamais « ce site est honnête ». Il dit « la connexion est chiffrée et le nom correspond au certificat présenté ». Un site d'hameçonnage peut parfaitement avoir un cadenas valide.

Q22. *Vous obtenez un 301 sur le port 80. Que contient l'en-tête Location de cette réponse ? Pourquoi vaut-il mieux tester une redirection avec curl -I qu'avec le navigateur ?*

```
sio@cli-linux:~$ curl -I http://intranet.bts-sio.lan/
HTTP/1.1 301 Moved Permanently
Location: https://intranet.bts-sio.lan/
```

L'en-tête Location contient l'adresse de destination, ici la même page en https. Un 301 sans Location n'aurait aucun sens : le code dit « ce n'est plus ici », l'en-tête dit où c'est. Les deux vont toujours ensemble.

curl -I vaut mieux que le navigateur pour deux raisons.

La première est le cache. Un 301 est une redirection permanente, et les navigateurs la retiennent, parfois des mois. Si je me trompe de cible, corriger la configuration ne suffira pas : le navigateur n'interrogera même plus le serveur, il ira directement à la mauvaise adresse. curl ne met rien en cache, il repose la question à chaque fois.

La seconde est la visibilité. Le navigateur suit la redirection tout seul et affiche la page finale. On voit le résultat, pas le mécanisme : impossible de savoir si c'était un 301 ou un 302, ni ce que contenait Location. curl -I s'arrête à la première réponse et montre l'étape. Avec l'option -L, il suivrait la chaîne, mais on peut alors voir chaque saut.

C'est aussi pour ça qu'on conseille de tester d'abord en 302, qui n'est pas mis en cache, et de passer en 301 une fois sûr de la cible.

Partie F. Certbot

Q23. *Citez les trois conditions qui manquent pour utiliser Let's Encrypt sur intranet.bts-sio.lan dans cette maquette. Pour chacune, dites précisément à quelle étape du défi HTTP-01 le processus échouerait. Indiquez enfin quel type de certificat est adapté ici, et quelle solution serait retenue en entreprise pour un service interne afin d'éviter l'avertissement du navigateur sur tous les postes.*

Le défi HTTP-01 se déroule en six temps. certbot demande un certificat, Let's Encrypt renvoie un jeton aléatoire, certbot dépose ce jeton dans le site sous /.well-known/acme-challenge/, les serveurs de l'autorité résolvent le nom dans le DNS public et viennent lire le jeton depuis Internet, et si le jeton est le bon la preuve est faite et le certificat délivré. Toute la logique repose sur une vérification faite depuis l'extérieur.

Il manque trois choses, et chacune bloque à une étape différente.

Premièrement, un nom de domaine public. Le suffixe .lan est un domaine privé qui n'existe dans aucun registre. L'échec est immédiat, à la première étape : Let's Encrypt refuse la demande sans même émettre de jeton, puisqu'il ne peut délivrer de certificat que pour un nom enregistré.

Deuxièmement, un enregistrement DNS public pointant vers le serveur. Ma zone bts-sio.lan n'est servie que par srv-linux sur VMnet2, et n'est visible de nulle part ailleurs. L'échec se produit à la quatrième étape, au moment où les serveurs de l'autorité essaient de résoudre le nom depuis Internet : ils obtiennent une réponse négative et ne savent pas à quelle adresse aller lire le jeton.

Troisièmement, le port 80 joignable depuis Internet. VMnet2 est en host-only, donc isolé, et 192.168.100.10 est une adresse privée non routable. Même en supposant les deux premiers points résolus, l'échec surviendrait à la quatrième étape aussi, mais un cran plus loin : le nom serait résolu, la connexion vers le port 80 n'aboutirait jamais et le jeton ne serait jamais lu.

Le type de certificat adapté ici est donc l'auto-signé, celui du TP. Il ne dépend de rien d'extérieur et fonctionne en réseau fermé. Son seul défaut est l'avertissement du navigateur.

En entreprise, pour un service interne, on ne se contente pas de demander aux utilisateurs de cliquer sur « accepter le risque ». On monte sa propre autorité de certification interne, on signe les certificats des services internes avec elle, et on distribue son certificat racine sur tous les postes, par GPO dans un domaine Windows. Les postes de l'entreprise reconnaissent alors cette autorité comme n'importe quelle autre, la chaîne de confiance est complète, et l'avertissement disparaît partout sans avoir rien acheté.

C'est aussi la bonne pratique pour une autre raison. Apprendre aux utilisateurs à passer outre un avertissement de sécurité est une mauvaise habitude : le jour où l'avertissement est justifié, ils cliqueront pareil.

Partie G. PHP

Q24. L'adresse IP affichée diffère selon que vous appelez la page depuis cli-linux ou depuis le poste Windows. D'où vient cette valeur, et que vaudrait-elle si un reverse proxy était placé devant le serveur ?

J'ai relevé les deux valeurs :

Client	Adresse affichée	Origine
cli-linux	192.168.100.50	Adresse fixée par réservation DHCP
poste Windows	192.168.100.1	Adresse de l'hôte sur VMnet2

Cette valeur vient de la variable `$_SERVER['REMOTE_ADDR']`. Ce n'est pas PHP qui la devine : elle est fournie par Apache, qui la tient de la connexion TCP elle-même. C'est l'adresse source des paquets reçus, celle avec laquelle le serveur communique. Elle est donc fiable, puisque sans elle le serveur ne saurait pas à qui répondre.

Avec un reverse proxy devant le serveur, elle vaudrait l'adresse du proxy, et pas celle du vrai client. Toutes les requêtes lui arriveraient de la même adresse, souvent 127.0.0.1 si le proxy est sur la même machine. Ce n'est pas un bug : du point de vue d'Apache, le client est bien le proxy, c'est lui qui a ouvert la connexion TCP. Le vrai client est plus loin et n'a jamais parlé directement à Apache.

Les conséquences sont concrètes. Toutes les lignes d'`access.log` porteraient la même adresse, ce qui rendrait les journaux inutiles. Et un filtrage `Require ip` deviendrait inopérant, comme je l'ai dit en question 14.

La solution est que le proxy ajoute l'adresse d'origine dans un en-tête, `X-Forwarded-For` ou `X-Real-IP`. Côté Apache, on installe `libapache2-mod-remoteip` et on lui dit de lire cet en-tête, pour que `REMOTE_ADDR` et les journaux retrouvent la vraie adresse. Avec une réserve importante : un en-tête est fourni par le client, donc falsifiable. Il ne faut faire confiance à un `X-Forwarded-For` que s'il vient d'un proxy qu'on maîtrise, jamais dans le cas général.

Q25. Créez temporairement `info.php` contenant `phpinfo()`, consultez-la, puis supprimez-la. Citez trois informations qu'elle expose et expliquez pourquoi la laisser en production est une faute. Indiquez aussi ce qui se passerait si `libapache2-mod-php` n'était pas installé et que vous demandiez `infos.php`.

J'ai créé la page, je l'ai consultée dans le navigateur et je l'ai supprimée tout de suite après. Trois informations parmi celles qu'elle affiche :

La version exacte de PHP et celle d'Apache. Un attaquant n'a plus qu'à chercher les failles connues de ces versions précises, au lieu de tester à l'aveugle.

La liste complète des extensions chargées. Elle indique ce que le serveur sait faire, donc à quoi il sert : une extension de base de données révèle qu'il y en a une derrière, et laquelle.

Les chemins absolus des fichiers de configuration et de la racine du site, plus les variables d'environnement. C'est le plan de l'installation, utile pour construire une attaque visant un fichier précis, et les variables contiennent parfois des identifiants de connexion à une base.

La laisser en production est une faute parce qu'elle offre gratuitement tout ce qu'un attaquant devrait mettre du temps à deviner. C'est une carte détaillée de la machine, publique et accessible sans aucune authentification. Les robots la cherchent en permanence : les noms `info.php`, `phpinfo.php` et `test.php` sont dans toutes les listes de balayage. Un `info.php` oublié est un classique des rapports d'audit.

Sans `libapache2-mod-php`, Apache ne saurait pas quoi faire d'un fichier `.php`. Il le traiterait comme un fichier ordinaire et l'enverrait tel quel. Le navigateur afficherait le code source, ou proposerait de

le télécharger. C'est bien plus grave qu'une page qui ne marche pas : tout le contenu du fichier devient lisible, y compris les mots de passe de base de données et les clés d'API qu'il contient souvent. C'est d'ailleurs un symptôme classique après une mise à jour ratée, et il faut réagir vite.

Partie H. Nginx

Q26. Pourquoi Nginx refusait-il de démarrer avant votre modification ? Recopiez le message d'erreur et expliquez-le. Citez trois façons différentes de faire cohabiter Apache et Nginx sur une même machine.

```
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
nginx: [emerg] bind() to [::]:80 failed (98: Address already in use)
nginx: [emerg] still could not bind()
```

Nginx refusait de démarrer parce que son fichier default écoutait sur le port 80, déjà occupé par Apache. Deux processus ne peuvent pas écouter le même couple adresse et port : le système attribue le port au premier qui le demande, et refuse aux suivants.

Le message décrit exactement ça. `bind()` est l'appel système qui réserve un port, et il échoue avec le code 98, « Address already in use ». Les tentatives apparaissent en double parce que Nginx essaie l'IPv4 sur 0.0.0.0:80 et l'IPv6 sur [::]:80. Et l'ensemble se répète trois fois : Nginx réessaie avant de renoncer, d'où la ligne finale « still could not bind() ».

Un point mérite d'être noté : c'est Nginx qui tombe, pas Apache. Le premier arrivé garde le port et n'est même pas prévenu. Installer un second serveur web ne casse donc pas le premier, ce qui est plutôt rassurant.

Trois façons de les faire cohabiter.

Des ports différents, c'est la solution du TP. Apache garde 80 et 443, Nginx passe en 8080. Simple, mais le port doit être écrit dans l'URL, ce qui ne convient qu'à un usage de test ou d'administration.

N'en garder qu'un seul, en désactivant l'autre avec `systemctl disable --now apache2`. C'est le choix à faire si on a migré d'un serveur à l'autre et qu'on ne veut pas laisser traîner un service inutile.

Nginx en reverse proxy devant Apache, qui est l'architecture professionnelle. Nginx prend les ports 80 et 443, gère le TLS et sert les fichiers statiques. Apache est déplacé sur 127.0.0.1:8080, donc joignable seulement en local, et traite le PHP. Les deux ne se marchent plus dessus et chacun fait ce qu'il fait le mieux. Il faut alors transmettre les en-têtes Host, X-Real-IP et X-Forwarded-For, sinon Apache croit que toutes les requêtes viennent de 127.0.0.1.

Q27. Quelle différence fondamentale y a-t-il entre un bloc `<Directory>` d'Apache et un bloc `location` de Nginx ? Expliquez aussi pourquoi Nginx n'a pas d'équivalent du `.htaccess` et en quoi c'est un choix de conception, pas un manque.

La différence tient à ce sur quoi chacun raisonne. Un bloc `Directory` d'Apache désigne un chemin sur le disque : `<Directory /var/www/intranet.bts-sio.lan/public_html>`. Un bloc `location` de Nginx désigne un morceau d'URL : `location /etat`.

Ce n'est pas juste une question d'écriture. Une même URL peut correspondre à plusieurs chemins, et un même chemin peut être atteint par plusieurs URL, par exemple à travers un alias ou un lien symbolique. Apache protège le fichier, Nginx protège l'adresse. Ma route `/etat` le montre bien : elle ne correspond à aucun fichier sur le disque, Nginx fabrique la réponse lui-même. Un bloc `Directory` ne saurait pas quoi en faire, il n'y a rien à désigner.

Nginx n'a pas d'équivalent du `.htaccess`, et c'est délibéré. Le `.htaccess` est un fragment de configuration posé dans un répertoire du site, relu par le serveur à chaque requête. Pour servir

/a/b/c/page.html, Apache doit chercher un .htaccess dans chacun des quatre niveaux, à chaque requête, multiplié par tout le trafic.

Nginx charge toute sa configuration une fois au démarrage et ne va plus jamais lire le disque pour ça. C'est cohérent avec son modèle événementiel, où quelques processus gèrent des milliers de connexions : un accès disque imprévu bloquerait tout le monde.

C'est un choix de conception parce que le .htaccess a trois autres défauts, indépendants de la performance. La configuration réelle se retrouve éparpillée dans l'arborescence, illisible d'un coup d'oeil. Une erreur dedans n'est pas vue par configtest et se manifeste par un 500 découvert par l'utilisateur. Et toute personne pouvant écrire dans le répertoire modifie la configuration du serveur, ce qui est une faille en soi.

Le .htaccess garde un usage précis : l'hébergement mutualisé, où le client n'a pas accès à /etc/apache2 et n'a que ça pour configurer son espace. Sur un serveur qu'on administre soi-même, la bonne pratique est AllowOverride None et tout dans le fichier .conf, ce que j'ai fait dans mes trois hôtes virtuels. On perd de la souplesse, on gagne en lisibilité et en testabilité.

Partie I. Le pare-feu

Q28. *Quelle différence observez-vous, du côté client, entre un port fermé par le pare-feu et un port sur lequel aucun service n'écoute ? Appuyez-vous sur les deux dernières commandes nc. Expliquez pourquoi ce comportement est délibéré du point de vue de la sécurité, et donnez la commande ufw qui n'autoriserait le port 8080 qu'au réseau 192.168.100.0/24.*

```
sio@cli-linux:~$ nc -zv 192.168.100.10 80 443 8080
srv-linux.bts-sio.lan [192.168.100.10] 80 (http) open
srv-linux.bts-sio.lan [192.168.100.10] 443 (https) open
srv-linux.bts-sio.lan [192.168.100.10] 8080 (http-alt) open

sio@cli-linux:~$ nc -zv -w 3 192.168.100.10 3306
srv-linux.bts-sio.lan [192.168.100.10] 3306 (mysql) : Connection timed out
```

La différence est le temps de réponse et le message. Un port fermé par le pare-feu ne répond rien du tout : le client attend, puis abandonne au bout du délai que je lui ai fixé avec -w 3, d'où « Connection timed out ». Sans ce délai, l'attente aurait duré bien plus longtemps.

Un port sans service mais non filtré se comporte autrement. Le système répond tout de suite par un paquet TCP de refus, et le client affiche « Connection refused » immédiatement. La machine dit « je suis là, mais il n'y a personne sur ce port ».

Ce silence est délibéré, pour deux raisons.

Il ne donne aucune information. Un refus immédiat confirme que la machine existe et qu'elle répond. Le silence ne dit rien : celui qui balaie ne peut pas distinguer un port filtré d'une machine éteinte ou d'une adresse inutilisée.

Il coûte cher à l'attaquant. Balayer mille ports en recevant mille refus instantanés prend une seconde. En recevant mille silences, il faut attendre le délai d'expiration pour chacun, ce qui prend des heures. Le balayage devient lent, donc visible, donc détectable.

C'est la différence entre les cibles DROP et REJECT dans un pare-feu. ufw utilise DROP par défaut, et c'est le bon choix pour un serveur exposé.

Pour n'autoriser le port 8080 qu'au réseau pédagogique :

```
sudo ufw allow from 192.168.100.0/24 to any port 8080 proto tcp
```

Cette règle aurait un sens réel ici. Le port 8080 ne sert qu'à ma démonstration Nginx : il n'a aucune raison d'être ouvert à tout le monde, contrairement à 80 et 443. Il faudrait alors supprimer

l'ancienne règle avec `ufw status numbered` puis `ufw delete`, sinon la première trouvée s'applique et la restriction ne servirait à rien.

5. Fichiers de configuration

[/etc/apache2/sites-available/www.bts-sio.lan.conf](#)

```
<VirtualHost *:80>
    ServerName www.bts-sio.lan
    ServerAlias bts-sio.lan
    ServerAdmin sio@bts-sio.lan
    DocumentRoot /var/www/www.bts-sio.lan/public_html

    ErrorLog ${APACHE_LOG_DIR}/www.bts-sio.lan-error.log
    CustomLog ${APACHE_LOG_DIR}/www.bts-sio.lan-access.log combined

    <Directory /var/www/www.bts-sio.lan/public_html>
        Options -Indexes +FollowSymLinks
        AllowOverride None
        Require all granted
    </Directory>
</VirtualHost>
```

[/etc/apache2/sites-available/intranet.bts-sio.lan.conf](#)

Ce fichier a servi deux fois. En partie D il portait le site en HTTP, avec la restriction par adresse puis l'authentification. En partie E, son contenu a été entièrement remplacé par une redirection : le port 80 ne sert plus de page, il renvoie vers le HTTPS. Voici la version en place à la fin du TP :

```
<VirtualHost *:80>
    ServerName intranet.bts-sio.lan
    Redirect permanent / https://intranet.bts-sio.lan/
    ErrorLog ${APACHE_LOG_DIR}/intranet-redirect-error.log
</VirtualHost>
```

Et la version de la partie D, avant le passage en HTTPS, dont le bloc `Directory` a été repris tel quel dans l'hôte virtuel 443 :

```
<VirtualHost *:80>
    ServerName intranet.bts-sio.lan
    ServerAdmin sio@bts-sio.lan
    DocumentRoot /var/www/intranet.bts-sio.lan/public_html

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

    <Directory /var/www/intranet.bts-sio.lan/public_html>
        Options -Indexes +FollowSymLinks
        AllowOverride None

        AuthType Basic
        AuthName "Intranet BTS SIO - acces reserve"
        AuthUserFile /etc/apache2/passwd/.htpasswd-intranet

        <RequireAll>
            Require ip 192.168.100.0/24
            Require valid-user
        </RequireAll>
    </Directory>
</VirtualHost>
```

[/etc/apache2/sites-available/intranet-ssl.bts-sio.lan.conf](#)

```
<VirtualHost *:443>
```

```

ServerName intranet.bts-sio.lan
DocumentRoot /var/www/intranet.bts-sio.lan/public_html

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

SSLEngine on
SSLCertificateFile /etc/ssl/certs/intranet.bts-sio.lan.crt
SSLCertificateKeyFile /etc/ssl/private/intranet.bts-sio.lan.key

# Desactive tout, puis reactive uniquement TLS 1.2 et 1.3.
SSLProtocol -all +TLSv1.2 +TLSv1.3

<Directory /var/www/intranet.bts-sio.lan/public_html>
    Options -Indexes +FollowSymLinks
    AllowOverride None

    AuthType Basic
    AuthName "Intranet BTS SIO - acces reserve"
    AuthUserFile /etc/apache2/passwd/.htpasswd-intranet

    # RequireAll : les deux conditions sont exigees.
    # Sans ce bloc, plusieurs Require sont en OU logique.
    <RequireAll>
        Require ip 192.168.100.0/24
        Require valid-user
    </RequireAll>
</Directory>
</VirtualHost>

```

[/etc/nginx/sites-available/demo.conf](#)

```

server {
    # 8080 pour ne pas entrer en conflit avec Apache sur le 80.
    # default_server : le site attrape-tout de ce port.
    listen 8080 default_server;
    server_name srv-linux.bts-sio.lan srv-linux;

    root /var/www/demo-nginx; # equivalent de DocumentRoot
    index index.html;        # equivalent de DirectoryIndex

    access_log /var/log/nginx/demo-access.log;
    error_log /var/log/nginx/demo-error.log;

    location / {
        # Essaie le fichier, puis le repertoire, sinon 404.
        try_files $uri $uri/ =404;
    }

    # Point de controle : aucun fichier sur le disque,
    # la reponse est fabriquee par Nginx.
    location = /etat {
        default_type text/plain;
        return 200 "Nginx operationnel\n";
    }
}

```

[/etc/apache2/passwd/.htpasswd-intranet](#)

```

sio:$apr1$yy6zHTpA$Av5wG7ZkkyPHyzCr0XdC.
alice:$apr1$1irYsDAa$ouz4nN0gmSJLtreYz8Wnc0

```

Ce fichier ne contient aucun mot de passe, seulement des empreintes. Il est en 640, propriété de root:www-data, et placé hors du DocumentRoot.

Le certificat et la clé

Le certificat est dans /etc/ssl/certs/intranet.bts-sio.lan.crt, en 644. La clé privée est dans /etc/ssl/private/intranet.bts-sio.lan.key, en 600 et propriété de root. Elle n'est pas recopiée ici, et elle ne figure pas dans l'archive rendue.

6. Difficultés rencontrées

Un fichier hosts modifié qui n'était pas le bon

À l'étape 2, j'ai ajouté mes trois lignes dans le fichier hosts de Windows avec le Bloc-notes, et j'ai enregistré. Le fichier affiché à l'écran contenait bien mes lignes. Pourtant ping www.bts-sio.lan continuait d'échouer après un ipconfig /flushdns.

J'ai vérifié ce que Windows lisait vraiment :

```
PS C:\Users\doria> Get-Content C:\Windows\System32\drivers\etc\hosts |  
                    Select-String bts-sio  
PS C:\Users\doria>
```

Aucun résultat. Le fichier lu par le système ne contenait pas mes lignes. Le Bloc-notes avait enregistré ailleurs, parce qu'il n'était pas lancé en administrateur : il ne peut pas écrire dans System32 et propose alors un enregistrement ailleurs, souvent avec l'extension .txt ajoutée.

J'ai contourné le problème en écrivant directement depuis PowerShell lancé en administrateur, avec Add-Content. Le Get-Content a ensuite affiché les trois lignes, et le ping a répondu.

Ce que j'en retiens : après avoir modifié un fichier, vérifier ce que le système lit, pas ce que l'éditeur affiche. Les deux ne sont pas toujours la même chose.

curl absent, et un client volontairement coupé d'Internet

Debian 12 en installation minimale ne livre pas curl. Sur srv-linux, un apt install a suffi. Sur cli-linux, la même commande a échoué :

```
sio@cli-linux:~$ sudo apt install -y curl  
Err:1 http://deb.debian.org/debian bookworm/main amd64 libcurl4  
  Impossible d'initialiser la connexion a deb.debian.org: 80  
  (2a04:4e42::644). - connect (101: Le reseau n'est pas accessible)  
  Connexion a debian.map.fastlydns.net: 80 (151.101.130.132) impossible,  
  delai de connexion depasse
```

Ce n'est pas une panne, c'est une conséquence de mon montage du TP1. J'avais laissé la carte NAT de cli-linux commentée dans /etc/network/interfaces, pour que le client ne reçoive sa passerelle et son DNS que de mon serveur. Il n'a donc aucun accès à Internet, et aucun dépôt n'est joignable.

J'ai écarté la solution facile, qui aurait été de rallumer la carte NAT : ça aurait cassé le montage validé au TP1. J'ai téléchargé les paquets sur srv-linux et je les ai transférés par SSH, en utilisant la clé mise en place au TP1 :

```
sio@srv-linux:~$ cd /tmp && apt-get download curl libcurl4  
  
sio@cli-linux:~$ scp 'sio@192.168.100.10:/tmp/*curl*.deb' /tmp/  
sio@cli-linux:~$ sudo dpkg -i /tmp/libcurl4_*.deb /tmp/curl_*.deb  
sio@cli-linux:~$ curl -I http://192.168.100.10/  
HTTP/1.1 200 OK
```

L'ordre compte : libcurl4 avant curl, puisque c'est sa dépendance. dpkg, contrairement à apt, ne va pas chercher les dépendances tout seul.

Ce que j'en retiens : une machine volontairement isolée reste administrable, à condition d'avoir gardé un chemin vers elle. Ici c'était SSH, monté au TP1 justement pour ça.

Un configtest qui répond Syntax OK sur un site qui n'existe plus

C'est la difficulté la plus instructive du TP. En modifiant le bloc Directory de l'intranet pour y ajouter l'authentification, j'ai effacé une ligne de trop dans nano. Le fichier ne contenait plus que le bloc Directory, sans le VirtualHost autour.

Les trois tests d'authentification ont alors donné trois fois 200, sans jamais demander de mot de passe. J'ai d'abord cru que mon RequireAll était mal écrit.

Le détail qui m'a mis sur la piste est le Content-Length :

```
HTTP/1.1 200 OK
Content-Length: 363
```

363 octets, c'est la taille de la page de la vitrine. Celle de l'intranet fait 272. Je n'étais donc pas du tout sur le bon site : mon hôte virtuel intranet avait disparu, et la requête tombait sur le site attrape-tout, qui est en Require all granted.

Le diagnostic a été immédiat avec la bonne commande :

```
sio@srv-linux:~$ sudo apache2ctl configtest
Syntax OK

sio@srv-linux:~$ sudo apache2ctl -S
VirtualHost configuration:
*:80 www.bts-sio.lan (/etc/apache2/sites-enabled/www.bts-sio.lan.conf:1)
```

Le lien symbolique existait, le fichier était inclus, et configtest répondait Syntax OK. Mais apache2ctl -S ne listait plus qu'un seul site. Un bloc Directory seul, sans VirtualHost autour, est syntaxiquement valide : Apache le lit comme une directive globale. La syntaxe était correcte, le sens ne l'était pas.

J'ai réécrit le fichier en entier, en comptant les trois fermetures : RequireAll, Directory, VirtualHost. Le site est réapparu et les trois tests ont donné 401, 200, 401.

Ce que j'en retiens, et c'est ma leçon principale de ce TP : configtest vérifie la grammaire, pas le sens. Après toute modification d'un hôte virtuel, la commande qui compte est apache2ctl -S, parce qu'elle montre ce qu'Apache a réellement compris. J'ai eu la même chose au TP1 avec un auto ens33 écrit deux fois : une configuration fautive ne produit pas toujours une erreur.

Cette faute s'est reproduite une fois de plus, quand j'ai remplacé le contenu de l'hôte virtuel HTTP par la redirection. nano n'avait pas vidé le fichier, et l'ancien contenu est resté sous le nouveau. Cette fois configtest a bien vu le problème, et il a donné le fichier et la ligne :

```
apache2: Syntax error on line 23 of
/etc/apache2/sites-enabled/intranet.bts-sio.lan.conf:
</VirtualHost> without matching <VirtualHost> section
Action 'configtest' failed.
```

La comparaison est parlante : la même mauvaise manipulation a donné une fois une erreur claire et une fois rien du tout. Après ces deux épisodes, j'ai arrêté d'utiliser nano pour remplacer un fichier entier et je suis passé à une commande tee, qui écrase vraiment tout.

Une règle Require ip mal recopiée, validée par configtest

En remettant la bonne valeur après le test du refus, j'ai retapé une adresse incorrecte. configtest a répondu Syntax OK et le reload est passé sans rien dire. Le curl renvoyait toujours 403 alors que le client était bien dans le réseau autorisé.

Un grep sur le fichier a montré la faute tout de suite. Même leçon que ci-dessus, sous une autre forme : une adresse mal écrite reste une adresse valide pour l'analyseur. Le journal error.log, lui, donnait la bonne information en nommant le client refusé.

Le profil ufw Apache Full qui n'existe pas

La commande donnée par l'énoncé a échoué :

```
sio@srv-linux:~$ sudo ufw allow 'Apache Full'
ERROR: Could not find a profile matching 'Apache Full'
```

Les profils d'ufw sont des fichiers déposés par les paquets dans /etc/ufw/applications.d/. J'ai listé ce qui était réellement présent :

```
sio@srv-linux:~$ ls /etc/ufw/applications.d/
bind9  nginx  openssh-server  ufw-bittorrent  ufw-chat
ufw-directoryserver  ufw-dnsserver  ufw-fileserver  ufw-loginserver
ufw-mailserver  ufw-printserver  ufw-proxyserver  ufw-webserver
```

Il n'y a pas de fichier apache2. Le paquet Apache de Debian 12 ne dépose pas de profil ufw, alors que ceux de Nginx et de bind9 le font. C'est visible dans la liste des profils disponibles, où Nginx Full existe mais pas Apache Full.

J'ai ouvert les deux ports directement avec `ufw allow 80/tcp` et `ufw allow 443/tcp`. Le résultat est identique, et l'état du pare-feu est même plus lisible : on voit les numéros de port plutôt qu'un nom de profil qu'il faut aller décoder.

Ce que j'en retiens : un profil applicatif dépend du paquet installé et de la distribution. Avant d'utiliser un nom trouvé dans une documentation, un `ufw app list` évite de chercher longtemps.

7. Conclusion

Tout ce qui était demandé fonctionne. Les deux sites sont en ligne sur la même machine et la même adresse IP, chacun avec son hôte virtuel. La vitrine `www.bts-sio.lan` est publique en HTTP et répond depuis `cli-linux` comme depuis le poste Windows. L'intranet est restreint au réseau `192.168.100.0/24`, protégé par mot de passe, publié en HTTPS avec un certificat que j'ai généré, et le port 80 renvoie vers le HTTPS par une redirection permanente.

Le reste suit. Une page PHP dynamique traite un formulaire et affiche l'adresse du client. Nginx tourne sur le port 8080 sans avoir gêné Apache. Le pare-feu est actif avec une politique de refus par défaut, les ports du jour 3 et ceux du jour 2 sont ouverts, et ma session SSH n'a jamais été coupée.

Rien ne reste en suspens. La page `info.php` a été supprimée après son test, la clé privée est en 600 et ne figure pas dans l'archive, et aucun mot de passe n'est recopié dans ce document. La partie F sur Certbot était une analyse et n'a pas été mise en oeuvre, comme le prévoyait l'énoncé.

Ce que je retiens de ce TP tient en trois points.

Le premier est qu'une configuration valide n'est pas une configuration correcte. `apache2ctl configtest` m'a répondu Syntax OK sur un fichier qui ne définissait plus aucun site, et sur une règle d'adresse que j'avais mal recopiée. La commande qui montre la vérité est `apache2ctl -S`, parce qu'elle affiche ce que le serveur a compris et pas ce que j'ai écrit. C'est la suite directe de ce que j'avais vu au TP1 avec le serial non incrémenté.

Le deuxième est qu'une panne de site web se coupe en deux. Le DNS amène le client jusqu'à la machine, l'en-tête Host l'amène jusqu'au bon site. La commande `curl -H "Host: ..." contre l'adresse IP` teste la seconde moitié sans la première. C'est ce qui m'a permis, plusieurs fois, de savoir de quel côté chercher.

Le troisième est que les mesures de sécurité ne valent qu'ensemble. Le filtrage par adresse dit d'où vient la demande, mais se contourne. Le mot de passe dit qui la fait, mais circule en clair en HTTP, et le décodage base64 de la question 17 l'a montré en une commande. Le chiffrement protège le mot de passe, mais un certificat auto-signé ne prouve aucune identité. Chacune bouche un trou que les autres laissent ouvert.

Les deux machines restent en état pour la suite. Elles gardent leur DHCP, leur DNS, leurs deux sites web et leur pare-feu. Le jour 4 s'appuiera dessus pour NFS et fail2ban.