Inventaire — Étape 6 : sécuriser l'API avec un token

Étape 6 du projet inventaire : protéger l'API avec un token partagé et comprendre les limites d'un tel système.

5TQ Synthèse

Jusqu'ici, n'importe qui peut envoyer un rapport au nom de n'importe quel PC. Cette dernière étape ajoute une protection minimale par **token partagé** et discute honnêtement de ses limites.

🎯 Pourquoi ce chapitre

Aujourd'hui, **n'importe qui** sur le réseau de l'école (ou pire, sur Internet si l'API est exposée) peut :

- envoyer un faux rapport au nom de `PC-DU-PROF`,
- saturer ton disque en envoyant des milliers de POST,
- piéger un autre élève en injectant des données fausses dans son module.

C'est **inacceptable** pour une API en production. Mais on a volontairement attendu cette dernière étape pour que tu **vives** ce problème avant de le résoudre. C'est comme ça qu'on retient à quoi sert la sécurité.

Partie 1 — La démonstration (à faire en classe)

Sans rien protéger, ouvre Postman et envoie ce JSON :

```
{
  "hostname": "PC-DE-LUDO",
  "timestamp": "2026-05-13T08:00:00",
  "system": { "os": "Windows 95 LOL" },
  "memory": { "totalGo": 0.5 },
  "disks": []
}
```

Recharge le dashboard. Le PC du prof affiche **Windows 95** et 512 Mo de RAM. **Catastrophe.** Tu viens de comprendre pourquoi une API sans authentification n'est pas une API.

Partie 2 — Le mécanisme du token (concept)

Un **token** est un secret partagé entre :

- le client (ton script PowerShell),
- le serveur (ton API PHP).

Le client envoie ce secret dans **chaque requête**. Le serveur vérifie qu'il correspond avant de traiter quoi que ce soit.

C'est **basique** mais déjà beaucoup mieux que rien.

> 💡 Ce n'est **PAS** un mot de passe : un mot de passe identifie une personne, un token identifie une **application autorisée**. Tu ne demanderais pas à ton script de taper un mot de passe à chaque exécution. Le token, lui, peut vivre dans un fichier de configuration.

Partie 3 — Où mettre le token ?

Pas dans l'URL :

```
❌ https://serveur/api/report?token=ABC123
```

Pourquoi ? Parce que les URLs sont :

- visibles dans les logs du serveur,
- visibles dans l'historique du navigateur,
- visibles dans les outils réseau,
- **journalisées partout**.

Dans un en-tête HTTP :

```
✅ X-API-Key: ABC123
```

Les en-têtes ne sont pas affichés dans les logs standards et ne sont pas journalisés par le navigateur.

Partie 4 — Côté serveur : vérifier le token

Crée `src/auth.php` (qui était vide jusqu'ici) :

```php
<?php
declare(strict_types=1);

const TOKEN_FICHIER = __DIR__ . '/../data/.api-token';

function token_attendu(): string
{
    if (!is_file(TOKEN_FICHIER)) {
        // Aucun token défini : on génère, on enregistre, on l'affiche une seule fois
        $nouveau = bin2hex(random_bytes(24));
        file_put_contents(TOKEN_FICHIER, $nouveau);
        // Pour le TP : afficher dans la réponse pour que l'élève le voie une fois
        return $nouveau;
    }
    return trim(file_get_contents(TOKEN_FICHIER));
}

function exiger_token(): void
{
    $envoye = $_SERVER['HTTP_X_API_KEY'] ?? '';
    $attendu = token_attendu();

    // hash_equals : comparaison "à temps constant" résistante aux attaques par chronométrage
    if (!is_string($envoye) || !hash_equals($attendu, $envoye)) {
        http_response_code(401); // Unauthorized
        header('Content-Type: application/json; charset=utf-8');
        echo json_encode(['error' => 'Token manquant ou invalide.']);
        exit;
    }
}
```

> ⚠️ **POURQUOI `hash_equals` ET PAS `===`** ? Parce qu'un `===` peut être détecté par chronométrage très fin : il s'arrête au premier caractère faux. `hash_equals` compare en temps constant, peu importe où est la différence. C'est une **bonne habitude** dès qu'on compare des secrets.

Partie 5 — Activer le token dans l'endpoint

Dans `public/api/report.php`, ajoute **au tout début** (juste après les headers) :

```php
require __DIR__ . '/../../src/auth.php';
```

```
exiger_token();
```

Et c'est tout. Toute requête sans `X-API-Key` valide reçoit un **401**.

Partie 6 — Côté PowerShell : envoyer le token

```powershell
$token = "ABC123ABC123..."  # le contenu de data/.api-token

Invoke-RestMethod -Uri $ApiUrl `
                  -Method Post `
                  -ContentType "application/json; charset=utf-8" `
                  -Headers @{ "X-API-Key" = $token } `
                  -Body $json
```

> 💡 **NE METS JAMAIS LE TOKEN EN DUR DANS LE SCRIPT PUBLIÉ SUR GITHUB.** Stocke-le dans une variable d'environnement Windows et lis-la :
>
> ```powershell
> $token = $env:INVENTAIRE_TOKEN
> if (-not $token) { throw "Token non défini." }
> ```

Partie 7 — Tester le bon comportement

Cas	Attendu
POST sans `X-API-Key`	**401**
POST avec un `X-API-Key` faux	**401**
POST avec le bon `X-API-Key`	**201**
GET sur le dashboard	**200** (lecture libre, on peut décider de protéger plus tard)

🤖 Travailler avec l'IA — bon réflexe en sécurité

Quand tu travailles sur du code de sécurité, **double-vérifie chaque suggestion**. L'IA peut :

- proposer une comparaison `===` au lieu de `hash_equals` ;

- suggérer de loguer le token quelque part ;
- te conseiller de mettre le token dans une URL.

Bon prompt :

> Voici mon code de vérification de token. [colle le code] Liste-moi 5 erreurs de sécurité **CLASSIQUES** que je pourrais commettre ici, et indique laquelle est présente dans mon code.

C'est une **technique très utile** : demander à l'IA de chercher des problèmes plutôt que de produire une solution. Tu transformes l'IA en relecteur.

Partie 8 — Limites de ce token (très important à comprendre)

Cette protection est **simple** mais **fragile**. Pour vraiment sécuriser une API en production, on ajouterait :

- **HTTPS obligatoire** (sinon le token transite en clair sur le réseau) ;
- **Un token par PC**, pas un token unique partagé (révocation individuelle) ;
- **Un système de signature** (HMAC) qui prouve que le PC connaît le secret sans envoyer le secret lui-même ;
- **Une rotation régulière** des tokens (changer tous les 90 jours par ex.) ;
- **Un rate limiting** (max N requêtes par minute) pour bloquer les abus.

Tout ça est hors-scope pour le TP, mais **tu dois pouvoir l'expliquer à l'oral**. Si l'examinateur demande « ton API est-elle sécurisée pour la prod ? », la bonne réponse est :

> Non, c'est une sécurité minimale par token partagé. Pour la production, il faudrait HTTPS, un token par client, du HMAC, de la rotation et du rate limiting.

🤖 Travailler avec l'IA — comprendre le « pourquoi »

Pour chacune des 5 améliorations ci-dessus, demande à l'IA :

> Explique-moi en 4 phrases **POURQUOI** un token unique partagé est moins sûr qu'un token par client. Donne-moi un scénario d'attaque concret où la différence est critique.

Tu apprendras plus en posant **un pourquoi** qu'en demandant **un comment**.

✅ Critères de réussite

- [ ] L'endpoint refuse les requêtes sans token (401).
- [ ] L'endpoint refuse les requêtes avec un mauvais token (401).
- [ ] Ton script PowerShell envoie correctement le token.
- [ ] Le token n'est **jamais** affiché dans une URL ou un log.
- [ ] Tu peux expliquer pourquoi `hash_equals` plutôt que `===`.
- [ ] Tu peux citer **3 limites** de ce système.

⚠️ Pièges fréquents

Piège	Solution
Token affiché en clair sur le dashboard	Jamais ! Le supprimer du HTML/log
Token commité dans Git	Mettre `.api-token` dans `.gitignore`
Le token change à chaque redémarrage	Le sauvegarder dans un fichier (ce qu'on fait)
401 incompréhensible côté PowerShell	`try/catch` qui affiche `$_.Exception.Response.StatusCode`

✍️ Bilan général du projet

À ce stade, tu as :

- un **client PowerShell** qui scanne ton PC,
- une **API PHP** qui reçoit, valide, stocke et protège,
- un **dashboard** et des **fiches PC** avec graphes,
- un **module spécifique** qui t'identifie dans la classe,
- une **authentification** basique mais propre.

C'est **un vrai mini-produit**. Pas un exercice. Pas un jouet.

📋 Évaluation finale

Tu présentes en 8 minutes :

1. La démo (1 min)
2. L'architecture (2 min) — schéma + choix
3. Ton module spécifique (2 min)
4. Une faille corrigée pendant le projet (1 min)
5. Une limite connue de ton système (1 min)
6. Une réflexion d'écoresponsabilité (1 min)

> 💡 **CONSEIL FINAL IA** : utilise l'IA pour **te préparer aux questions du jury**, pas pour préparer ton speech. Prompt-type : *"Voici mon projet [résumé]. Quelles sont les 5 questions les plus difficiles qu'un jury de prof d'informatique pourrait me poser ?"* Si tu sais répondre aux 5, tu passes.

Bon courage !

Pour aller plus loin

Va jeter un œil au mockup UI qui te montre à quoi pourrait ressembler le dashboard final, avec sa direction artistique et ses composants visuels.