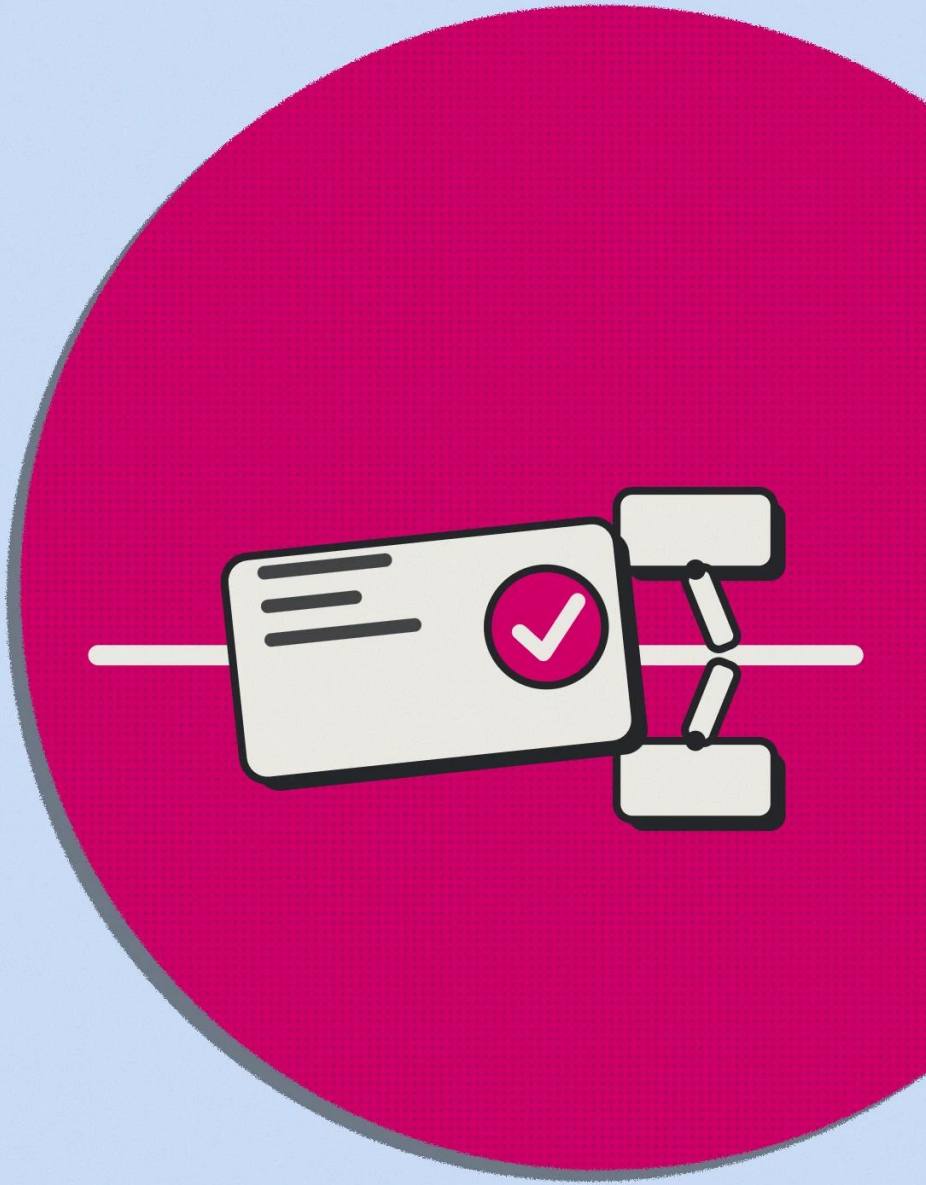
2.11

TEMPS 2 · LE TAMAGOTCHI TROUÉ · SÉANCE 2

Avant de mettre en ligne

Thibault Six · IFOSUP Wavre · 2026-2027



? Un site Laravel en ligne tourne en mode débogage. Qu'est-ce qu'un attaquant peut en tirer ?

Le mode débogage allumé en ligne a exposé la faille d'Ignition, et Androxgh0st le cherche.

novembre 2020

Lexfo trouve une faille dans Ignition

Ignition : en 2020, la page d'erreur détaillée de Laravel

`CVE-2021-3129`

exécuter du code sur un site en mode débogage

janvier 2024

la CISA et le FBI décrivent Androxgh0st

cherche les sites Laravel dont le fichier `.env` est lisible ou qui tournent en mode débogage

pour y prendre des identifiants de messagerie et de cloud

pas de mode débogage en production, et retirer puis révoquer les identifiants de `.env`

Laravel 13 n'installe plus Ignition : sa page de débogage fait partie du framework

La faille venait du code d'Ignition ; le mode débogage permettait de l'atteindre. Le Tamagotchi n'a rien à corriger ici, seulement des réglages.

? Une page plante sur un site où APP_DEBUG vaut true. Qu'est-ce que le visiteur peut lire ?

Avec `APP_DEBUG=true`, une erreur montre au visiteur le code, les requêtes SQL et les cookies.

`.env` | `APP_DEBUG=true` le mode débogage, allumé comme dans le kit

▼ une route provisoire qui lève une exception



La page de Laravel 13 ne montre pas le contenu de `.env`, mais elle en dit assez pour préparer une attaque.

PLUS BAS, LA MÊME PAGE

Exception trace le dossier du projet sur le serveur, le code source

Queries chaque requête SQL avec ses valeurs, dont l'identifiant de la session

Headers les cookies en entier

chapitre 2.06 : l'apostrophe de L'os a ouvert la page de débogage de Laravel

```
Illuminate\Database\Eloquent\Builder->get()
18          if ($search !== '') {
19              $items->whereRaw("name LIKE '%{$search}%'");
20          }
```

En haut, ce que la page affiche d'abord. À droite, ce qu'elle donne plus bas, sans rien masquer : le visiteur n'a qu'à faire défiler.

? Même adresse, avec APP_DEBUG=false dans .env : que reste-t-il au visiteur, et où part le détail ?

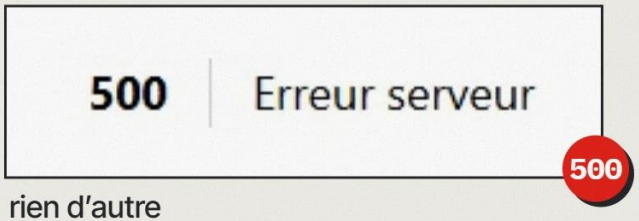
Avec `APP_DEBUG=false`, le visiteur lit « 500 Erreur serveur », et `laravel.log` garde le détail.

`.env` | **`APP_DEBUG=true`** →



le code, les requêtes SQL et les cookies

`.env` | **`APP_DEBUG=false`** →



rien d'autre

- - → **`storage/logs/laravel.log`**

le détail n'est pas perdu : Laravel l'écrit ici, comme au chapitre 1.22

Sur votre copie, remettez `APP_DEBUG=true` et retirez la route provisoire.

Une seule ligne change entre les deux bandes. Le visiteur ne lit plus que « 500 Erreur serveur » ; le détail est dans `storage/logs/laravel.log`.

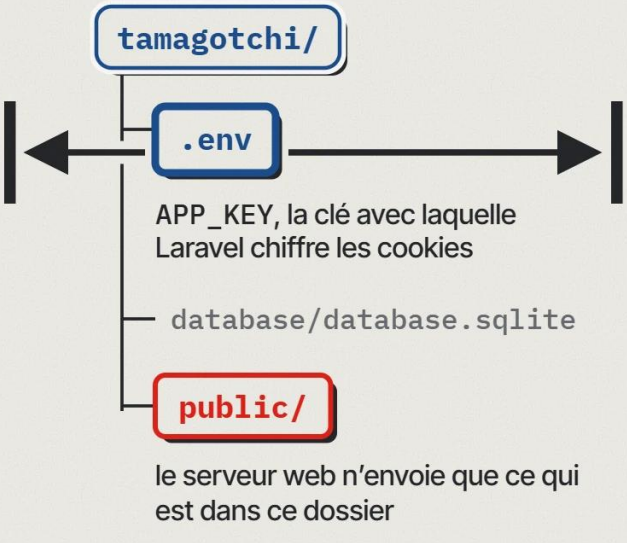
? .env contient APP_KEY et, en ligne, les accès à la base. Par quels deux chemins peut-il sortir ?

.gitignore écarte .env de Git, et le serveur web n'envoie que le contenu du dossier public.

PAR GIT

```
$ git check-ignore -v .env
.gitignore:15:.env    .env
```

```
$ git ls-files | grep env
.env.example
```



PAR LE SERVEUR WEB

```
$ curl -s -i http://tamagotchi.test/.env
HTTP/1.… 404 Not Found
```

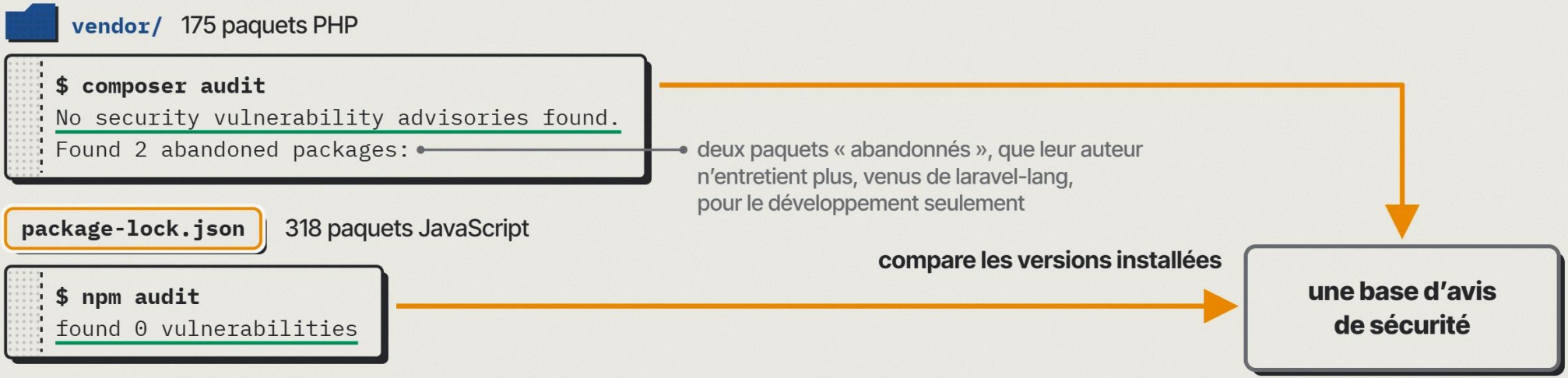
404

un secret qui a déjà été poussé reste publié : il se révoque

À gauche, Git ne suit que `.env.example`. À droite, l'adresse `/.env` n'est qu'une route inconnue de Laravel : 404.

? 175 paquets PHP, 318 paquets JavaScript : comment savoir si l'un d'eux a une faille connue ?

Avant chaque mise en ligne, `composer audit` et `npm audit` comparent vos paquets aux avis publiés.



Un paquet sain ce soir peut recevoir un avis demain : relancez les deux commandes avant chaque mise en ligne.

En vert, la réponse attendue : aucun avis. Les deux paquets abandonnés servent au développement seulement.

? Alice, connectée au Tamagotchi, ouvre la page d'un autre site. Que peut faire cette page ?

Le formulaire piégé arrive sans `Sec-Fetch-Site` ni jeton : Laravel répond 419.

requête forgée : envoyée depuis un autre site au nom d'une personne connectée, en anglais cross-site request forgery, d'où le sigle CSRF



Le formulaire part seul, dès le chargement, sans cookie. Le portillon pose deux questions, dans cet ordre : pas d'en-tête, pas de jeton, donc 419.

? Vos propres formulaires écrivent aussi. Pourquoi ne reçoivent-ils pas 419 ?

Sur `tamagotchi.test`, `Sec-Fetch-Site` manque : le jeton décide, et votre `<Form>` l'envoie.

votre portfolio en https

↓

`Sec-Fetch-Site: same-origin`

l'origine décide d'abord, et le jeton sert de secours

`http://tamagotchi.test`

↓

~~`Sec-Fetch-Site`~~ **absent**

Chrome n'envoie `Sec-Fetch-Site` qu'aux adresses qu'il juge sûres

sans jeton → 419 avec **`X-XSRF-TOKEN`** → ✓ accepté

LE JETON, ENVOYÉ PAR <FORM>

un autre site ne peut pas lire le jeton

Laravel — cookie **`XSRF-TOKEN`** → **`<Form>`** → **`POST /codes-cadeau`**

le jeton chiffré, posé à chaque réponse, lisible par le JavaScript de la page

Inertia lit ce cookie et recopie sa valeur dans l'en-tête `X-XSRF-TOKEN`

`x-xsrf-token: …`

En haut, qui décide selon l'adresse. En bas, le trajet du jeton, du cookie jusqu'à l'en-tête de la requête.

? Le cookie de session a trois attributs à régler. Lequel manque sur votre copie, et pourquoi ?

Le cookie de session est déjà `httponly` et `samesite=lax` ; en ligne, il lui manque `secure`.

config/session.php

```
'secure' => env('SESSION_SECURE_COOKIE'),
'http_only' => env('SESSION_HTTP_ONLY', true),
'same_site' => env('SESSION_SAME_SITE', 'lax'),
```

En ligne, ajoutez **`SESSION_SECURE_COOKIE=true`**.

LA RÉPONSE DE /LOGIN

```
Set-Cookie: tamagotchi-session=…; expires=…; Max-Age=7200; path=/; httponly; samesite=lax
```

`httponly`
interdit au JavaScript de lire le cookie de session

`samesite=lax`
le retient sur les formulaires POST venus d'un autre site, comme le formulaire piégé

`secure`
manque : il ne laisserait partir le cookie qu'en https

les fichiers téléversés : le serveur n'accepte que les formats prévus et les fichiers de 2 Mo au plus, et `store()` choisit le nom du fichier enregistré

Lisez chaque attribut du cookie, puis remontez à sa ligne de configuration. L'attribut `secure` manque en local, et c'est voulu.

? Sur votre portfolio en ligne, quelles lignes de la liste pouvez-vous vérifier dès ce soir ?

Ce soir, vérifiez `APP_DEBUG`, la racine `public` et le cookie de session de votre portfolio.

EN SÉANCE, SUR VOTRE PORTFOLIO EN LIGNE

1 ☐	**APP_DEBUG**	Le .env du serveur contient APP_ENV=production et APP_DEBUG=false, et une erreur affiche bien « 500 Erreur serveur », sans détail.
2 ☐	**la racine public**	Le domaine sert le dossier public, d'après l'interface d'OVH, et https://votre-adresse.be/.env répond 404.
3 ☐	**le cookie de session**	Le cookie de session est httponly, samesite=lax, et secure grâce à SESSION_SECURE_COOKIE=true.

Terminez la liste après la séance, puis passez-la sur le projet en binôme avant sa mise en ligne.

Trois lignes de la liste, à cocher ce soir sur votre portfolio. La liste entière est dans le chapitre ; chaque ligne se vérifie par une commande ou un regard.