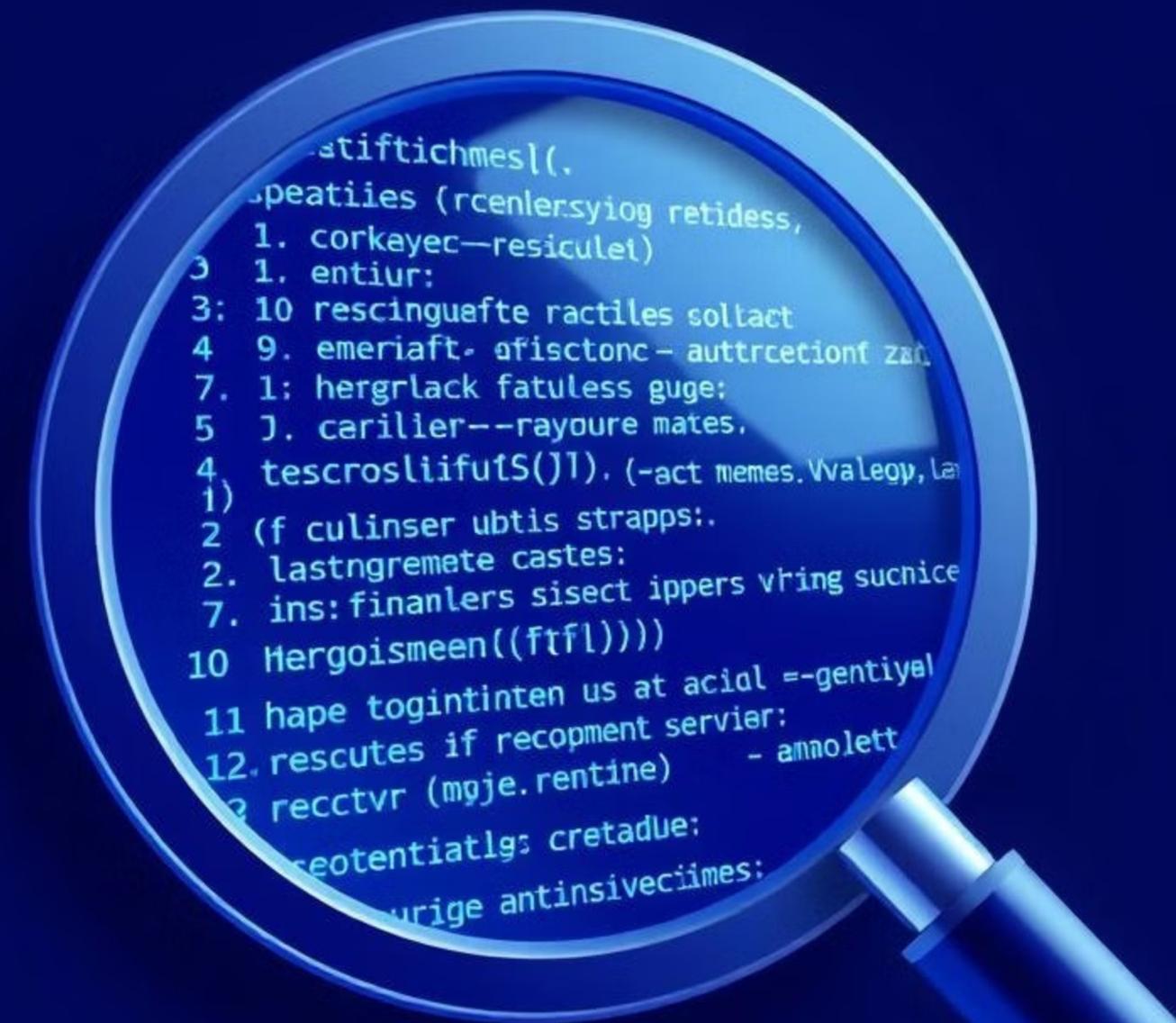


Audit de la sécurité du code-source PHP

Ce rapport constitue les résultats d'un audit de sécurité du code source PHP d'un projet. L'audit a révélé des faiblesses potentielles ainsi que des recommandations visant à améliorer la sécurité globale du code.

Catalina Danila, Philippe Delente et Camille Girard



Contexte et objectifs de l'audit

Contexte

Une audit a été mené avec l'objectif d'évaluer la sûreté du code-source PHP d'un projet.

L'objectif est de chercher des vulnérabilités possibles et des recommandations pour l'adresser.

Objectifs

Les résultats de l'audit révèlent de faiblesses potentielles et des recommandations pour projeter la sûreté de code en général. L'audit valide aussi les meilleures pratiques de sécurité de code de la pratique de code code en tant que sécurité citée.

Analyse des cookies : sécurité et HttpOnly



Sécurité

Les cookies sont sécurisés en utilisant le flag Secure pour les transmettre uniquement via HTTPS.



HttpOnly

Le flag HttpOnly permet d' empêcher le code Javascript d'accéder aux cookies. Cela empêche les attaques XSS.

Sécurité des cookies: état actuel non sécurisé

Absence de configuration

Aucune configuration explicite pour sécuriser les cookies ou les rendre HttpOnly n'a été trouvée.

Manque de protection

Ces cookies sont actuellement exposés à des attaques potentielles telles que le vol de session et le piratage de compte.

Risques de sécurité

Pas de filet de sécurité adéquat pour les cookies, ce qui peut être un risque élevé pour la sécurité de l'application.



Sécurité des Identifiants de Base de Données

Identifiants de base de données utilisés dans les URL difficile à deviner : état actuel sécurisé

Les identifiants de la base de données sont cachés dans le fichier `.env` et ne sont pas exposés directement dans les URL. Toutefois, le fichier `.env` n'est pas dans le fichier `.gitignore`, ce qui pourrait être considéré comme une brèche massive.

Utilisation de JSON Web Token (JWT)

1 Sécurité

Utiliser un algorithme de signature fort pour le JWT.

2 Confidentialité

Éviter de stocker des informations sensibles dans le JWT.

3 Expiration

Définir une durée d'expiration raisonnable pour le JWT.

JSONWebToken (JWT) qui n'expose aucune information sensible : état actuel sans objet

Nous n'avons pas trouvé d'utilisation explicite de JWT dans le projet.



Gestion des variables d'environnement

1

Sécurité

Stocker les informations sensibles dans des variables d'environnement.

2

Confidentialité

Ne jamais exposer les variables d'environnement dans le code.

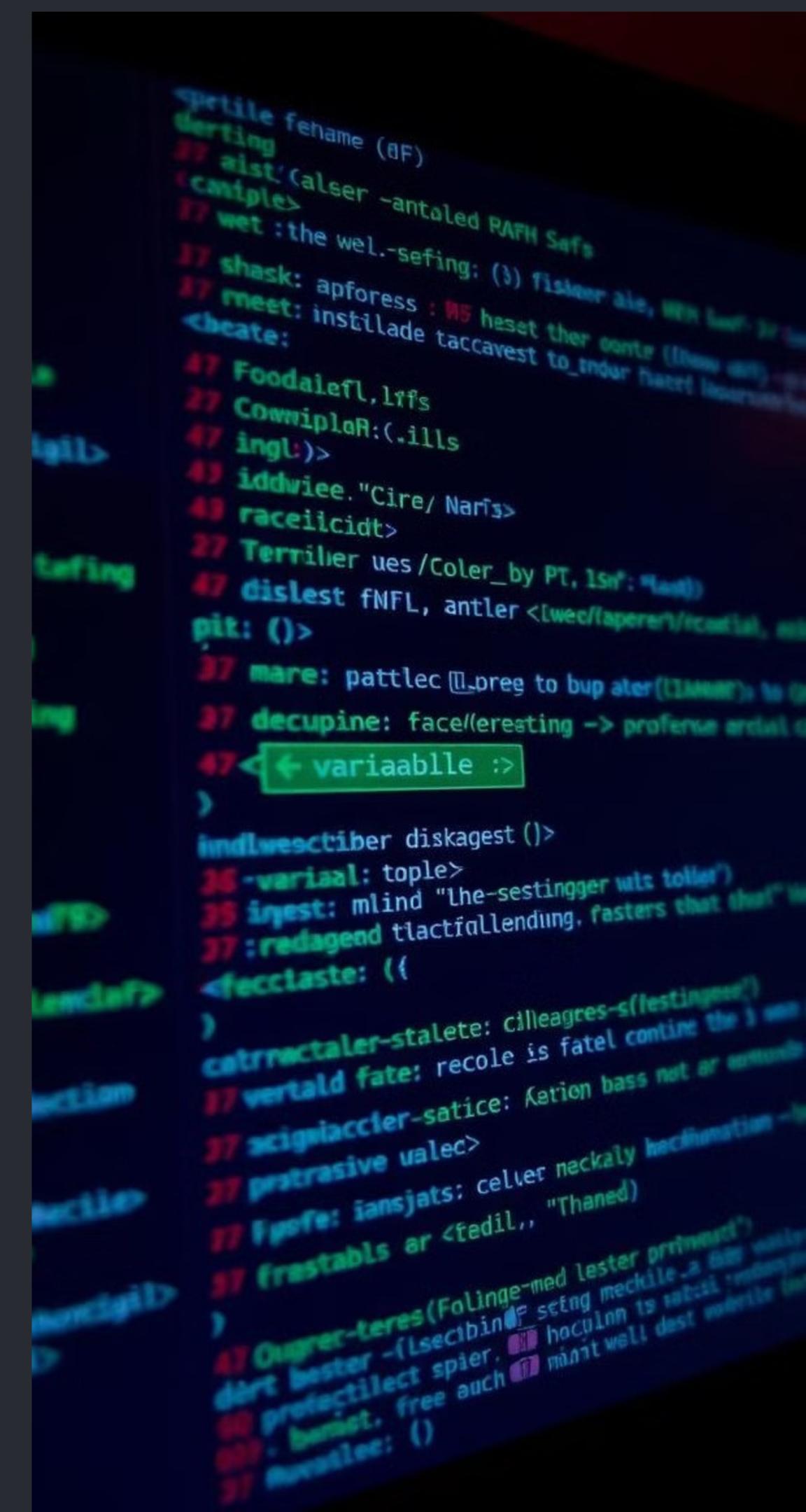
3

Accès

L'accès à ces variables doit être limité aux comptes d'utilisateur avec la permission d'y accéder.

Variables d'environnement non-exposées dans le code-source : état actuel Partiellement sécurisé

Les variables sensibles (comme POSTGRES_USER et POSTGRES_PASSWORD) sont bien placées dans le fichier .env, mais ce fichier doit être protégé par un .gitignore pour éviter une exposition dans un dépôt public, ce qui n'est pas le cas.



```
variable: fename (AF)
variable: ait: (alser -antaled RAPH Sef)
variable: wet : the wel.-sefing: (3) fisior ale, WII sur to
variable: shask: apforess : WII heset ther oante (lhou on
variable: meet: instillade taccavest to_inde hazel hou
variable: <beate:
variable: Foodalefl, lvis
variable: CowiplaR:(.ills
variable: ingl:>
variable: idwiee: "Cire/ Naris>
variable: raceilcidt>
variable: Terrier ues/Coler_by PT, lsm: Nasu)
variable: dislest fNFL, antler <Lwec/laperst/cast)st, m
variable: pit: ()>
variable: mare: pattlec @l_preg to bup ater(lmmer) to
variable: decupine: face/eresting -> profenss ardat c
variable: <+ variaable :>
variable: >
variable: indwesctiber diskagest ()>
variable: -variaal: tople>
variable: ipest: mind "lhe-sestingger wts toller"
variable: :redagend tiactfallendung, fasters that that" w
variable: <feccaste: (
variable: catrrectaler-staete: cilleagues-s(festigem)
variable: vertald fate: recole is fatel contine the
variable: scigniacller-satice: Kation bass not ar unive
variable: prostrative valec>
variable: Fofe: iansjats: celver neckaly hackingus ->
variable: frastabls ar <tedil., "Thaned)
variable: Ouprec-teres(Folinge-med lestar priment)
variable: dert bester -(Lsectbinde setng mechile-a
variable: al protectlect spier. # hoquin is retul se
variable: buriet, free auch # minit well dest mirele
variable: <avatice: ()
```

Opportunités d'amélioration identifiées

Validation

Améliorer la validation des données d'entrée pour éviter les injections SQL et les attaques XSS.

Cryptage

Mettre en place un cryptage des données sensibles, comme les mots de passe.

Secrets utilisés robustes et difficiles à deviner : état actuel à vérifier

Il est nécessaire d'avoir une vérification de la complexité des mots de passe et secrets contenu dans le fichier .env. Au niveau de la capacité de vérifier la robustesse des secrets, aucun mécanisme explicite pour effectuer une telle validation n'a été repéré dans le code.



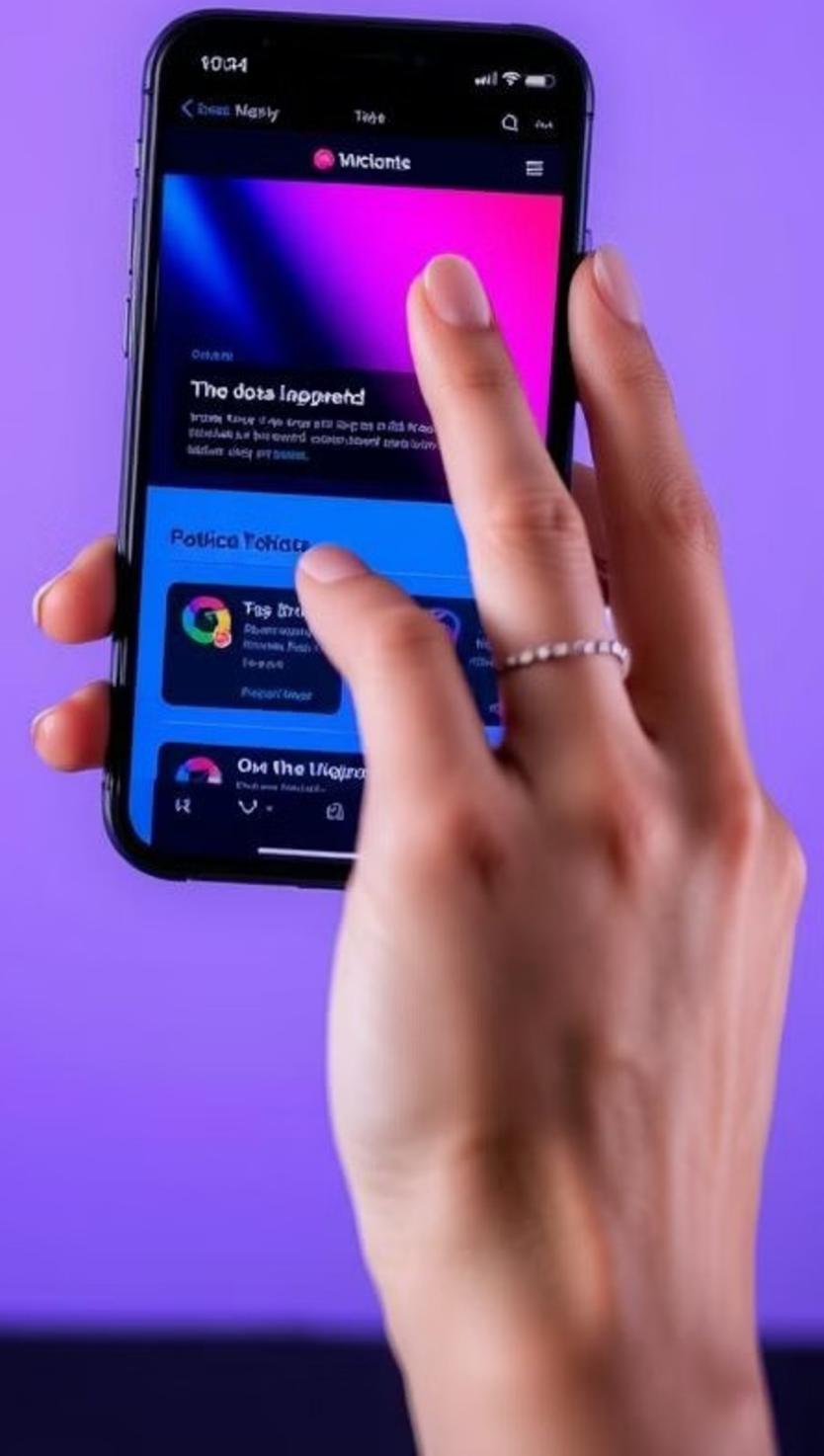
Failles XSS : Risques et Mitigation

Les failles XSS (Cross-Site Scripting) permettent à des hackers d'injecter du code malveillant dans la page web.

Ces injections permettent le vol d'informations confidentielles, la modification du comportement de l'application ou l'exécution d'attaques de type "phishing".

Pas de failles XSS : état actuel non sécurisé

Le projet ne permet pas de prévenir les failles XSS (comme l'utilisation de `htmlspecialchars()` ou des frameworks sécurisés). Il est possible que des données utilisateur soient injectées sans validation dans certaines vues.



Injections SQL : Risques et Mitigation

Les injections SQL exploitent les failles de sécurité dans les applications Web qui permettent aux attaquants d'exécuter des commandes SQL non autorisées.

Ces injections peuvent permettre aux attaquants de contourner les mécanismes de sécurité, de modifier ou supprimer des données, d'obtenir des accès non autorisés, voire de prendre le contrôle du système.

Pas d'injection SQL possibles : état actuel sécurisé

Les requêtes SQL utilisent des mécanismes sécurisés via PDO (`prepare()` et `execute()`). Cela protège efficacement contre les risques d'injection SQL.

En-têtes de sécurité HTTP

Les en-têtes HTTP jouent un rôle crucial dans la sécurité des applications Web.

Ils offrent un moyen efficace de protéger les données sensibles contre les attaques et d'améliorer la confidentialité.

En-têtes de sécurité présents : état actuel non sécurisé

Aucun en-tête de sécurité (comme Content-Security-Policy, Strict-Transport-Security, ou X-Content-Type-Options) n'a été configuré dans le projet.



Autres observations

Le projet utilise des dépendances via `composer` et `npm`. Il faut s'assurer de garder ces dépendances à jour et d'examiner leurs vulnérabilités régulièrement avec des outils comme `npm audit` et `composer audit`.

Pour la sécurité, il faudrait automatiser les mise à jour des `composer` et `npm`.

Recommandations et prochaines étapes

Les recommandations de ce rapport visent à améliorer la sécurité du code-source PHP. Il est essentiel de mettre en œuvre les correctifs et les mesures de sécurité nécessaires pour renforcer la sécurité du projet.

