Powerzio Audit

Bank Application Security Analysis

Analysis

The bank application, written in Python and interfacing with a database, is small but has several critical vulnerabilities that expose it to significant security risks. These vulnerabilities include:

1. **SQL Injection Vulnerabilities**: All API routes are susceptible to SQL Injection (SQLi), allowing unauthorized database manipulation.
2. **File Inclusion Vulnerability in /partials/<file>**: This route allows unrestricted file inclusion without verification, potentially exposing sensitive files.
3. **Lack of Access Control for Admin Features**: The /partials/<file> route does not verify if the user is an admin, exposing privileged content to regular users.
4. **Unrestricted Access to Backups**: The route /bkps/<filename> allows anyone to download backup files, risking sensitive data leakage.
5. **Command Injection in Admin Panel**: The admin panel allows command injection, which can lead to arbitrary command execution on the server.
6. **Lack of Input Validation**: Many API routes do not verify the types or structure of inputs, leading to possible injection attacks and data inconsistencies.

Remediation

To mitigate these vulnerabilities, we implemented a reverse proxy using **Nginx** in front of the bank application. We then configured **ModSecurity** as a Web Application Firewall (WAF) with the **OWASP Core Rule Set (CRS)** to monitor and validate all incoming traffic. This configuration helps block harmful requests by enforcing security rules and patterns.

- **Restricted Access to Sensitive Routes**: Nginx is configured to deny access to /partials/admin.html and /bkps/<filename> routes. These routes either expose sensitive functions or lack access control, making them high-risk.
- **Input and Command Filtering**: ModSecurity provides an additional layer of security by blocking suspicious commands and filtering inputs for SQLi, XSS, and command injection attempts.

Recommendations

For future security improvements:

1. **Use Prepared SQL Statements**: Implement parameterized queries to prevent SQL Injection attacks.
2. **Input Validation and Type Checking**: Validate all incoming parameters (URL search parameters, POST body data) for type and structure to ensure they conform to expected formats.
3. **Access Control Implementation**: Implement role-based access control, especially for sensitive routes like /partials/admin.html, to restrict access based on user roles.

This layered security approach reduces the risk of unauthorized access and mitigates SQLi, XSS, and command injection vulnerabilities.

Social Wall Security Analysis

Analysis

The Social Wall application allows users to upload and view images, but it has several critical security vulnerabilities in its image upload functionality:

1. **Command Injection through Filename**: Users can inject commands within the filename, potentially executing unauthorized commands on the server.
2. **Lack of File Type Validation**: There is no restriction on file types uploaded, allowing users to upload non-image files. This could lead to various risks, including site defacement or malicious file uploads.

Remediation

As modifications to the Social Wall code are permitted, we made the following security improvements to mitigate these risks:

- **Safe Filename Generation**: We generate unique filenames using UUIDs, preventing user-controlled filenames from containing malicious commands.
- **Switching to Process.run**: By using Process.run instead of Shell.run, we avoid shell interpretation of commands, which reduces the risk of command injection.
- **File Type and MIME Type Validation**: We added validation for the file extension, allowing only PNG and JPEG formats. Additionally, we verify the MIME type to prevent non-image files from being uploaded.

We also incorporated a **Web Application Firewall (WAF)** in front of the Social

Wall, utilizing **Nginx** and **ModSecurity** with the **OWASP Core Rule Set (CRS)**. This setup helps monitor and block malicious requests before they reach the application, further safeguarding against SQLi, XSS, command injection, and other common web vulnerabilities.

Recommendations

To maintain and enhance the security of the Social Wall:

1. **Additional File Validation**: Periodically review and tighten validation checks on uploaded content types and sizes.
2. **Monitoring and Logging**: Implement logging and regular monitoring of upload activities to quickly detect and respond to any unusual patterns.
3. **Apply Security Patches**: Ensure the application and dependencies are consistently updated to mitigate potential vulnerabilities.

Kevin GOUYET