App information	
Name	OpenEMR
Developer	Open-emr.org
Category	EHR and medical practice management solution
Supported platforms	Windows, Linux, Mac OS X
Version	7.0.0
Last update	11/12/2023
Compatible idioms	More than thirty
Source	Open source
License	GNU General Public License
Main language	PHP + MySQL
Actors	
Auditor	Carlos M. Mejía-Granda
RE team	Carlos M. Mejía-Granda José L Fernández-Alemán Juan Manuel Carrillo-de-Gea José A. García-Berná

Audit summary

Carried out on December 27, 2023.

OpenEMR was installed on Ubuntu 20.04.6 LTS (Focal Fossa), Apache/2.4.41, MariaDB 10.3.38, and PHP 7.4.3.

The objective of this audit is to evaluate the fulfillment of the application to the requirements of SEC-CAT concerning security.

The requirements focus on the following ten characteristics:

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and outdated components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery

The audit was carried out using the SEC-AM.

The application is generally classified as "Secure", with a score of 69.97%. It incorporates mechanisms to mitigate vulnerabilities such as Canonical Representation, Race Conditions, Memory Use After Free, Buffer Overflow Attacks, SQL injection attacks, privilege elevation, and unauthorized access to data processing. The software follows a formal user registration and deregistration process, enabling users to be assigned to workgroups and granting access to records based on these groups. It generates an individual authenticator before using a group authenticator, employs a robust FIPS-validated cryptographic infrastructure to secure electronic health information, and enforces effective restrictions on the types of files users can upload. Additionally, it implements an information output validation mechanism. However, a few deficiencies have been identified:

Deficiencies:

1. The application doesn't implement and apply dual access control before deleting users, medical records.
2. The application doesn't implement data entry validation measures before storing them to guarantee the integrity and security of the information processed, stored, or accessible through end-user point devices.
3. The application doesn't implement protection measures against command injections.
4. The application doesn't implement protection measures against Cross-Site Scripting (XSS) vulnerabilities to prevent malicious code injection attacks.
5. The application doesn't show a system of use of the system and privacy notification before granting access.
6. The application doesn't physically or logically separate the functionality of the user, including user interface services and system management functionality.
7. The application contains authentication data incorporated into its code files.
8. The application is housed on a general-purpose server.
9. The application doesn't implement an adequate limitation of access routes to prevent malicious route manipulation, enabling the unauthorized revelation of files or directories.
10. The application doesn't define, document, and implement necessary security configurations in the most restrictive way consistent with operational requirements.
11. The application doesn't define and implement due security settings to avoid DDoS attacks.
12. The application reveals technical details about architecture in error events to users.
13. The application doesn't configure session cookies with the following properties:
    - Establish the HTTPOnly flag in session cookies.
    - Ensure that the safe flag (Secure) is activated in session cookies.
14. The application has components or libraries outdated.
15. The application keeps libraries or software components that do not have support or maintenance.
16. The application doesn't provide notifications or alerts when product updates and safety-related patches are available.
17. The application doesn't implement safe login procedures that protect against login attempts by brute force.
18. The application doesn't require the re-authentication of users and devices when information systems' authenticators, roles, or security categories change.
19. The application doesn't allow a temporary password for the session, with an immediate change to a permanent password.
20. The application doesn't show the following information when a successful login is completed:
    - Date and time of the last successful login.
    - Details of any login and login attempts since the last successful session.
21. The application doesn't ensure that changing the password requires changing at least eight characters.

22. The application doesn't limit the number of concurrent sessions for each account to three.
23. The application doesn't restrict the duration of connections to application services to provide additional security in access to those applications.
24. The application doesn't automatically deactivate the user accounts after an inactivity period of 35 days.
25. The application doesn't acquire components only from official sources through safe links, preferring signed packages that reduce the possibility of including a modified or malicious component.
26. The application doesn't incorporate a "software supply chain safety tool" to verify that the components do not contain known vulnerabilities.
27. The application doesn't have a process for reviewing code and configuration changes to minimize the possibility of malicious code or configuration introduction in the software pipeline.
28. The application doesn't ensure that its continuous integration channel/continuous deployment (CI/CD) has adequate segregation, safe configuration, and access control to guarantee the integrity of the code that flows through construction and implementation processes.
29. The application cannot show the previous content of a record at any point in the past, as well as the associated details of who entered, agreed, or modified the data and at what time.
30. The application cannot ensure that audit records contain information that establishes the following:

    - The function performed by the accessing user;
    - network addresses and protocols
    - destination IP addresses
    - device identity or location, if possible, and system identifier;
    - In the case of access override to blocked or masked records or portions of records, a reason for the override, as chosen by the user making the access
    - In the case of changes to consent directives made by a substitute decision-maker, the identity of the decision-maker.
31. The application doesn't back up the audit records at least every seven days in a system or component different from the one being audited.
32. The application doesn't maintain audit records for a specific time, as determined by regulations and medical standards. This retention must cover from 7 to 75 years, depending on the information and legal requirements of the State.
33. The application doesn't transfer audit records to a system or medium different from the audited system and is not configured to write application records in a centralized repository.
34. The application doesn't generate audit records when a concurrent session is produced from different workstations.
35. The application is not configured to send an alarm to the Information Security Officer (ISSO) and the System administration (SA) when the audit system fails or is experiencing failure.

Recommendations:

1. Implement a dual access control mechanism before allowing the deletion of users and medical records. It can require two levels of authorization, such as approval from a supervisor or a secondary authentication step, before allowing the deletion of users or medical records.
2. Implement data entry validation measures before storing information to guarantee integrity and security. Developers can use input validation functions to ensure that user-entered data is of the correct type, length, and format, preventing potential SQL injection or other data manipulation attacks.
3. Implement protection measures against command injections by utilizing parameterized queries or prepared statements in database interactions to prevent malicious input from being interpreted as executable code.
4. Implement protection measures against Cross-Site Scripting (XSS) vulnerabilities by sanitizing user input and encoding output to prevent the execution of malicious scripts. It is recommended to use security libraries or frameworks that automatically handle input sanitization.
5. Implement a mechanism for notifying users of system usage and privacy conditions before granting access. It could be done by displaying a pop-up or notification window during login, outlining terms of use, privacy policies, and any other relevant information that users must acknowledge before accessing the system.
6. Physically or logically separate user functionality, including user interface services and system management functionality. Organizations can utilize role-based access control (RBAC) to restrict access to system management functionality to authorized personnel while providing standard user access to relevant user interface services.
7. Remove authentication data from code files and adopt secure authentication mechanisms. The system can use environment variables or configuration files external to the codebase to store sensitive authentication data. Employ secure authentication protocols like OAuth or JWT.
8. Deploy the application on a dedicated server with proper security configurations. Utilize a web application firewall (WAF) to filter and monitor HTTP traffic, implement intrusion detection and prevention systems (IDPS), and follow server hardening best practices.
9. Implement access controls to prevent malicious route manipulation and unauthorized access to files or directories. Developers can use access control lists (ACLs) or role-based access control (RBAC) to restrict access to specific routes. Regularly audit and validate access control.
10. Define, document, and implement necessary security configurations in the most restrictive way consistent with operational requirements. Follow security baselines such as CIS benchmarks, turn off unnecessary services, apply the principle of least privilege, and regularly update and patch the system.
11. Implement security settings to mitigate DDoS attacks by employing rate limiting, traffic analysis, and content delivery network (CDN) services to distribute and absorb traffic during a DDoS attack.
12. Avoid revealing technical details in error events to users by customizing error messages to provide generic information to users while logging detailed error messages for internal review. Use a centralized logging system.
13. Configure session cookies with security flags. Set the HTTPOnly flag to prevent client-side access to cookies via JavaScript. Activate the Secure flag to ensure cookies are transmitted over only secure (HTTPS) connections.

14. **14.** Regularly update and replace outdated components and libraries. Use package managers to keep dependencies up to date, monitor vulnerability databases, and perform regular security audits.
15. **15.** Replace unsupported components or libraries with actively maintained alternatives. Identify alternative libraries or components that fulfill the same functionality and update the application accordingly. Remove or replace components that are no longer supported.
16. **16.** Implement a notification system to alert users about product updates and safety-related patches. Provide in-app notifications or send email alerts to users whenever updates or patches are available, encouraging them to apply the latest security measures.
17. **17.** Implement secure login procedures to protect against brute-force attacks. Introduce account lockout mechanisms after a specified number of failed login attempts and use CAPTCHA challenges to prevent automated brute force attacks.
18. **18.** Require re-authentication when there are changes in authenticators, roles, or security categories. The system can prompt users to re-enter their credentials or perform an additional authentication step when there are changes to their access permissions or security roles.
19. **19.** Allow the use of temporary passwords with an immediate change to a permanent one. The system can send users a temporary password via email, and upon login with the temporary password, it prompts them to set a new permanent password.
20. **20.** Display relevant information after a successful login, such as the date and time of the last successful login and details of login attempts. Show a user-friendly dashboard upon login with details about the last login, recent login attempts, and any relevant security information.
21. **21.** Ensure that changing the password requires a strong password with a minimum length. Enforce a policy that mandates a minimum password length of eight characters and includes a mix of uppercase, lowercase, numbers, and special characters.
22. **22.** Limit the number of concurrent sessions for each account. The system could allow only three simultaneous logins for a single user account to prevent unauthorized access.
23. **23.** Implement restrictions on the duration of connections to application services. Set session timeout limits to log out users automatically after a defined period of inactivity.
24. **24.** Automatically deactivate user accounts after a specified inactivity period. Disable accounts that have been inactive for 35 days to reduce the risk of unauthorized access.
25. **25.** Acquire software components only from official and secure sources. Establish a process to verify and obtain components from trusted repositories, ensuring they are signed and unmodified.
26. **26.** Integrate a software supply chain safety tool into the development process to verify components for known vulnerabilities. Implement a tool like OWASP Dependency-Check to automatically scan and identify third-party libraries' and components' vulnerabilities.
27. **27.** Establish a formal process for reviewing code and configuration changes to minimize the introduction of malicious code. Implement code review practices using tools like GitLab CI/CD pipelines with mandatory code reviews before merging changes into the main codebase.
28. **28.** Ensure the CI/CD channel has adequate segregation, safe configuration, and access control to guarantee code integrity. Use separate CI/CD environments for development, testing, and production, with strict access controls and secure configurations at each stage.
29. **29.** Enable the application to show the previous content of a record and associated details at any point in the past. Implement version control or record history functionality that

captures changes, including who made the change, when, and the nature of the modification.

30. Include detailed information in audit records such as the function performed, network addresses, destination IP addresses, device identity or location, reasons for access overrides, and changes to consent directives.

31. Set up automated processes to regularly back-up audit records to a secure and isolated storage system.

32. Configure the system to retain audit records for the required period, ensuring compliance with relevant laws and regulations.

33. Use secure protocols to transfer audit records to a centralized repository isolated from the audited system, providing an additional layer of security.

34. Implement mechanisms to detect and record concurrent sessions, capturing details of each session for auditing purposes.

35. Configure the application to send alerts to the ISSO and SA in case of audit system failures.