Secure Healthcare Software Development

CSP009

PRESENTATION BY: STYLIANOS KARAGIANNIS (PDMFC, PORTUGAL)

Code issue on PACS system

Incomplete String Escaping or Encoding

- **Location:** Found in jquery.mobile.simpledialog2.js at line 137.
- **Issue:** Incomplete string escaping, specifically addressing only the first occurrence of " potentially leaving other instances vulnerable to injection attacks.
- **Tool Used:** CodeQL detected this issue.
- **Recommendation:** Ensure comprehensive string escaping or encoding for all occurrences of special characters to prevent injection attacks like SQL injection or cross-site scripting (XSS).
- **Example:** Demonstrates incomplete escaping where only the first occurrence of " is addressed, leaving subsequent occurrences vulnerable.
- **References:** CWE-20, CWE-80, CWE-116 highlight the weaknesses associated with incomplete string escaping and the importance of thorough sanitization for security.

Unsafe HTML Construction

Code issue on PACS system

Unsafe HTML Construction from Library Input

- **Location:** Identified in jquery.blockui.js at line 253.
- **Issue:** HTML construction relies on potentially unsafe inputs, which can lead to cross-site scripting (XSS) vulnerabilities.
- **Tool Used:** CodeQL flagged this issue.
- Recommendation: Document that the function should only be used with trusted inputs to prevent inadvertent injection of unsafe HTML fragments.
- **Example:** Shows HTML construction dependent on input, potentially exposing the application to XSS attacks if unsafe inputs are used.
- **References:** CWE-79 and CWE-116 highlight the risks associated with unsafe HTML construction and the importance of validating inputs to mitigate XSS vulnerabilities.

Cross-site Scripting Attack

Code issue on Healthcare Information System

Cross-site scripting attacks

- **Location:** Detected in jqModal.js at line 48.
- **Issue:** Dynamic HTML construction without proper sanitization of user input (c.ajaxText), potentially leading to cross-site scripting (XSS) attacks.
- **Tool Used:** CodeQL identified this vulnerability.
- **Recommendation:** Document all inputs susceptible to XSS attacks and implement robust input sanitization measures to mitigate risks.
- **Example:** Demonstrates a safer approach utilizing jQuery's .find() method for HTML manipulation.
- **References:** OWASP and CWE provide guidance on XSS prevention and the importance of secure jQuery plugin development practices.

Thank you

Presenter: Stylianos Karagiannis (PDMFC, Portugal)

Please send all questions to:
stylianos.karagiannis@pdmfc.com