A defined systems/software development life cycle process that includes:

Friday, August 16, 2024 2:40 PM

Stmt 21: Defined Systems/Software Development Life Cycle Process

Checklist:

1. **Documentation Review:**
   - Verify that there is a documented systems/software development life cycle (SDLC) process.
   - Ensure that the documentation outlines all phases of the SDLC, including planning, design, development, testing, deployment, and maintenance.
2. **Compliance Check:**
   - Confirm that the documented SDLC process complies with relevant industry standards and regulations.

Stmt 21.1: Documented Secure Application Development Process

Checklist:

1. **Process Documentation:**
   - Review the documented secure application development process.
   - Ensure that it includes security requirements, coding standards, and secure design principles.
2. **Approval and Updates:**
   - Verify that the secure application development process is approved by appropriate stakeholders.
   - Check for regular updates and reviews of the process.
3. **Integration with SDLC:**
   - Ensure that the secure application development process is integrated into the overall SDLC.

Stmt 21.2: Secure Code Repository and Controls

Checklist:

1. **Repository Review:**
   - Confirm that a secure code repository is in place.
   - Verify that the repository supports version control, access control, and audit logging.
2. **Access Controls:**
   - Review controls for checking code in and out, ensuring that only authorized personnel have access.
   - Verify that code changes are logged and reviewed.
3. **Security Measures:**
   - Check for encryption and other security measures protecting the code repository.

Stmt 21.3: Appropriate Segregation of Duties

Checklist:

1. **Role Definitions:**
   - Verify that roles and responsibilities in the development process are clearly defined.
   - Ensure that duties are segregated to minimize conflicts of interest and reduce risk.
2. **Access Control:**
   - Review access control mechanisms to ensure they enforce segregation of duties.
3. **Documentation:**
   - Confirm that documentation reflects the segregation of duties and is reviewed periodically.

Stmt 21.4: Root Cause Analysis Process

Checklist:

1. **Process Documentation:**
   - Review the documented process for root cause analysis.
   - Ensure it includes steps for identifying, analyzing, and addressing underlying issues in code vulnerabilities.
2. **Implementation:**
   - Verify that the root cause analysis process is implemented and followed in practice.
3. **Reporting and Review:**
   - Confirm that findings from root cause analysis are reported and reviewed for continuous improvement.

Stmt 21.5: Inventory of Third-Party Components

Checklist:

1. **Inventory Review:**
   - Ensure that there is an updated inventory of third-party components used in development.
   - Verify that the inventory includes version information and security status.
2. **Monitoring and Updates:**
   - Confirm that the inventory is regularly updated and monitored for vulnerabilities.
3. **Documentation:**
   - Review documentation to ensure it reflects the current state of third-party components.

Stmt 21.6: Separate Environments for Production and Non-Production Systems

Checklist:

1. **Environment Configuration:**
    - Verify that separate environments are established for production and non-production systems.
    - Ensure that non-production environments are isolated from production systems.
2. **Access Controls:**
    - Review access control mechanisms for each environment to ensure appropriate restrictions.
3. **Documentation:**
    - Confirm that documentation reflects the separation of environments and is kept up-to-date.

Stmt 21.7: Secure Code Training for Personnel

Checklist:

1. **Training Programs:**
    - Verify that all software development personnel receive training in writing secure code.
    - Ensure that the training is specific to their development environment and responsibilities.
2. **Training Records:**
    - Review records of training sessions and participant completions.
3. **Ongoing Education:**
    - Confirm that training is updated regularly and includes the latest secure coding practices.

Stmt 21.8: Static and Dynamic Analysis Tools

Checklist:

1. **Tool Deployment:**
    - Verify that static and dynamic analysis tools are applied within the application life cycle.
    - Ensure that these tools are integrated into the development and testing processes.
2. **Usage and Coverage:**
    - Confirm that the tools are used consistently and cover relevant aspects of secure coding practices.
3. **Results and Actions:**
    - Review how results from the analysis tools are addressed and whether vulnerabilities are mitigated.

Stmt 21.9: Process for Addressing Software Vulnerabilities

Checklist:

1. **Vulnerability Reporting:**
    - Verify that there is a process for accepting and addressing reports of software vulnerabilities.
    - Ensure that there is a means for external entities to report vulnerabilities.
2. **Response and Resolution:**
    - Review how reported vulnerabilities are tracked, evaluated, and resolved.
3. **Communication:**
    - Confirm that there is a communication mechanism for informing stakeholders about resolved vulnerabilities.
4. **Documentation:**
    - Ensure that all reported vulnerabilities and actions taken are documented appropriately.

Analysis Tools

Friday, August 16, 2024 2:50 PM

1. Static Analysis Tools
- **Purpose**: Analyze code without executing it, focusing on finding vulnerabilities, code quality issues, and adherence to coding standards.
- **Examples**:
    - **SonarQube**: Provides continuous inspection of code quality and security vulnerabilities.
    - **Checkmarx**: Analyzes source code for security vulnerabilities.
    - **Fortify Static Code Analyzer**: Identifies security vulnerabilities and coding errors.
    - **FindBugs/SpotBugs**: Detects potential bugs and code quality issues in Java programs.
    - **PMD**: Analyzes Java source code for potential problems, including coding style violations.

2. Dynamic Analysis Tools
- **Purpose**: Analyze code during runtime to detect issues such as memory leaks, performance bottlenecks, and security vulnerabilities.
- **Examples**:
    - **OWASP ZAP (Zed Attack Proxy)**: Performs dynamic application security testing to find vulnerabilities.
    - **Burp Suite**: An integrated platform for performing security testing of web applications.
    - **AppScan**: Provides dynamic testing of web applications to uncover security vulnerabilities.
    - **Dynatrace**: Monitors application performance and detects runtime issues.

3. Interactive Application Security Testing (IAST) Tools
- **Purpose**: Combine elements of both static and dynamic analysis by monitoring applications during runtime and providing real-time feedback.
- **Examples**:
    - **Contrast Security**: Provides real-time application security analysis.
    - **Veracode IAST**: Offers interactive security testing with real-time feedback.

4. Software Composition Analysis (SCA) Tools
- **Purpose**: Analyze third-party libraries and components used in applications to identify vulnerabilities and licensing issues.
- **Examples**:
    - **Snyk**: Scans dependencies for vulnerabilities and provides remediation advice.
    - **WhiteSource**: Identifies and manages open-source vulnerabilities and licensing issues.
    - **Black Duck**: Provides visibility into open-source components and their associated risks.

5. Performance Analysis Tools
- **Purpose**: Assess the performance characteristics of an application, including load handling, response times, and resource usage.
- **Examples**:
    - **JMeter**: Apache JMeter is used for performance testing and load testing.
    - **Gatling**: A performance testing tool designed for web applications.
    - **New Relic**: Provides performance monitoring and management.

6. Code Quality Tools
- **Purpose**: Evaluate and improve the overall quality of code, including readability, maintainability, and adherence to coding standards.
- **Examples**:
    - **Codacy**: Offers automated code reviews and quality analysis.
    - **CodeClimate**: Provides code quality metrics and insights.
    - **SonarLint**: An IDE extension that helps developers spot code quality issues in real-time.

7. Security Information and Event Management (SIEM) Tools
- **Purpose**: Collect, analyze, and respond to security events and incidents across the application environment.
- **Examples**:
    - **Splunk**: Provides insights into security events and incidents.
    - **ELK Stack (Elasticsearch, Logstash, Kibana)**: Collects and analyzes log data for security monitoring.

8. Code Coverage Tools
- **Purpose**: Measure the extent to which the codebase is exercised by tests, helping to identify untested parts of the code.
- **Examples**:
    - **JaCoCo**: Provides code coverage metrics for Java applications.
    - **Cobertura**: Analyzes code coverage for Java projects.
    - **Coverage.py**: Measures code coverage for Python programs.

Tools

Friday, August 16, 2024 2:52 PM

1. OWASP ZAP (Zed Attack Proxy)

- **Focus**: Web application security testing.
- **Key Features**:
  - Automated scanners to find common vulnerabilities (e.g., SQL injection, XSS).
  - Manual tools for exploring and testing web applications.
  - Support for scripting and extensions.
  - API integration for CI/CD pipelines.
- **Ease of Use**: User-friendly interface, suitable for both beginners and experts.
- **Integration**: Integrates well with CI/CD tools like Jenkins, Docker, and various scripting languages.
- **Performance**: Performs well for small to medium applications; however, it might require fine-tuning for large-scale testing.
- **Cost**: Open-source and free.

2. Burp Suite

- **Focus**: Web application security testing.
- **Key Features**:
  - Automated vulnerability scanning and manual testing capabilities.
  - Extensive set of tools, including a proxy, intruder, repeater, and sequencer.
  - Customizable with extensions via the BApp store.
  - Advanced scanning features in the Professional version, such as scanning for complex vulnerabilities.
- **Ease of Use**: Slight learning curve due to the wide range of features; however, the interface is intuitive for regular users.
- **Integration**: Integrates with CI/CD pipelines and various security tools via API.
- **Performance**: High-performance, especially with the Professional version, which offers faster scanning and more advanced features.
- **Cost**: Free Community version; Professional version is paid.

3. AppScan (IBM Security AppScan)

- **Focus**: Enterprise-level web and mobile application security testing.
- **Key Features**:
  - Advanced dynamic analysis for finding security vulnerabilities.
  - Scanning for mobile, web, and web services applications.
  - Provides comprehensive reports and compliance checking (e.g., PCI DSS).
  - Integration with various development tools and environments.
- **Ease of Use**: Enterprise-level tool with a robust interface; may require some training for full utilization.
- **Integration**: Extensive integration capabilities with IBM's other security tools and various CI/CD platforms.
- **Performance**: Efficient scanning, suitable for large-scale enterprise environments.
- **Cost**: Commercial product with pricing based on enterprise needs.

4. Veracode Dynamic Analysis

- **Focus**: Web application security testing.
- **Key Features**:
  - Scans web applications for vulnerabilities during runtime.
  - Provides detailed remediation guidance based on findings.
  - Continuous monitoring and testing in pre-production environments.
  - Scalable to large enterprise applications and multiple projects.
- **Ease of Use**: Cloud-based platform, easy to set up and use, especially for teams familiar with security testing.
- **Integration**: Integrates well with CI/CD tools, development environments, and other Veracode products.
- **Performance**: High-performance, with robust scanning capabilities tailored for large, complex applications.
- **Cost**: Commercial product with tiered pricing based on the number of applications and scans.

5. Acunetix

- **Focus**: Web application and network security testing.
- **Key Features**:
  - Automated dynamic application security testing (DAST) with additional network security scanning.
  - Scans for over 6,500 vulnerabilities, including SQL injection, XSS, and more.
  - Offers deep scanning for HTML5, JavaScript, and single-page applications (SPAs).
  - Provides detailed reports and integrates with popular issue trackers.
- **Ease of Use**: User-friendly with a well-designed interface; suitable for both small and large organizations.
- **Integration**: Supports integration with CI/CD tools and issue tracking systems like Jira.
- **Performance**: Fast and efficient, even with complex and large applications.
- **Cost**: Commercial product with various pricing options depending on the scale and features needed.

6. Netsparker

- **Focus**: Web application security testing.
- **Key Features**:
  - Automated scanning with Proof-Based Scanning™ to eliminate false positives.
  - Supports both static and dynamic vulnerability scanning.
  - Integrates with CI/CD pipelines and various development tools.
  - Detailed vulnerability analysis and remediation guidance.
- **Ease of Use**: Easy to set up and use, with a focus on reducing false positives.
- **Integration**: Strong integration with CI/CD pipelines and issue tracking tools.
- **Performance**: High-performance with efficient scanning, designed for both small and large enterprises.
- **Cost**: Commercial product with pricing based on features and number of targets
- **Summary Comparison Table**

Tool	Focus	Key Strengths	Integration	Ease of Use	Cost
OWASP ZAP	Web app security	Open-source, flexible, community-driven	High	High	Free
Burp Suite	Web app security	Comprehensive tools, extensibility	High	Medium	Paid
AppScan	Enterprise security	Enterprise features, compliance-focused	High	Medium	Paid
Veracode	Enterprise security	Cloud-based, scalable, detailed remediation	High	High	Paid
Acunetix	Web & network security	Extensive vulnerability coverage, fast	High	High	Paid
Netsparker	Web app security	Accurate scanning, integration capabilities	High	High	Paid

Choosing the Right Tool:

- **Small to Medium Organizations**: Tools like **OWASP ZAP** and **Burp Suite** are great for organizations needing robust security testing without the high costs associated with enterprise tools.
- **Large Enterprises**: **AppScan**, **Veracode**, and **Netsparker** are ideal for larger organizations requiring comprehensive security analysis with extensive integration capabilities and enterprise-level support.
- **Specific Needs**: Tools like **Acunetix** offer a balance between web application and network security, which might be valuable for organizations needing both in a single tool.

API Security

Friday, August 16, 2024 2:53 PM

1. Security

- **Authentication**:
  - Verify that authentication mechanisms (e.g., OAuth, JWT, API keys) are properly implemented.
  - Ensure secure storage and transmission of credentials.
  - Confirm that access tokens have appropriate lifetimes and scopes.
- **Authorization**:
  - Ensure proper role-based access control (RBAC) is implemented.
  - Verify that users can only access resources they are authorized for.
  - Test for improper access to endpoints (e.g., through IDOR - Insecure Direct Object References).
- **Data Protection**:
  - Ensure that sensitive data (e.g., PII, financial data) is encrypted in transit (using TLS/SSL) and at rest.
  - Verify that APIs do not expose sensitive data unnecessarily in responses or logs.
- **Rate Limiting & Throttling**:
  - Implement and verify rate limiting and throttling to protect against DoS attacks.
  - Ensure proper error handling for rate-limited requests.
- **Input Validation**:
  - Verify that all inputs are validated to prevent injection attacks (e.g., SQL injection, XSS).
  - Check for proper handling of input length, format, and type.
- **Error Handling**:
  - Ensure consistent and secure error messages that do not expose sensitive information.
  - Verify proper handling of HTTP status codes (e.g., 4xx, 5xx).

2. Functionality

- **Endpoint Testing**:
  - Verify that each API endpoint is functioning as expected.
  - Check for correct responses for valid and invalid inputs.
  - Ensure that CRUD operations (Create, Read, Update, Delete) work as intended.
- **Data Integrity**:
  - Ensure that data is correctly saved, retrieved, and modified as expected.
  - Verify that data consistency is maintained across different API calls.
- **Dependency Testing**:
  - Verify that the API correctly handles dependencies (e.g., databases, external services).
  - Ensure graceful handling of failures in dependent services.
- **Documentation Validation**:
  - Ensure that the API documentation is complete, accurate, and up-to-date.
  - Verify that the documentation includes sample requests and responses.

3. Performance

- **Load Testing**:
  - Test the API under expected load to ensure it can handle the required number of concurrent requests.
  - Identify and address any bottlenecks or performance degradation.
- **Stress Testing**:
  - Push the API beyond its normal load to test its breaking point and recovery behavior.
  - Monitor the API's performance under stress (e.g., response times, error rates).
- **Latency**:
  - Measure and optimize the API's response times.
  - Ensure that the API meets the required performance benchmarks.
- **Caching**:
  - Verify that caching mechanisms are correctly implemented and used.
  - Ensure that cache invalidation is handled properly when data changes.

4. Compliance

- **Legal & Regulatory Compliance**:
  - Ensure the API complies with relevant regulations (e.g., GDPR, CCPA, HIPAA).
  - Verify that data retention policies are properly implemented.
- **Logging & Monitoring**:

    - Ensure that the API logs all critical actions (e.g., authentication attempts, data access).
    - Verify that logs do not expose sensitive data.
    - Implement monitoring to track API usage and detect anomalies.
- **Auditability**:
    - Ensure that the API provides enough data to support audits (e.g., access logs, change logs).
    - Verify that audit logs are secure and tamper-proof.

5. Usability

- **Versioning**:
    - Ensure that API versioning is implemented to support backward compatibility.
    - Verify that deprecated versions are properly handled and documented.
- **Consistency**:
    - Check that naming conventions, URL structures, and parameter formats are consistent across endpoints.
    - Ensure that response formats (e.g., JSON, XML) are consistent and follow the specified schema.
- **Error Messages**:
    - Verify that error messages are informative and follow a consistent structure.
    - Ensure that error codes are correctly implemented and documented.
- **Localization**:
    - If applicable, ensure that the API supports localization for different languages and regions.

6. Reliability

- **Availability**:
    - Ensure that the API is available and responsive under normal conditions.
    - Verify that uptime meets the required service level agreements (SLAs).
- **Redundancy**:
    - Verify that redundancy mechanisms are in place to ensure availability (e.g., failover systems).
    - Test the API's behavior in the event of server failures or outages.
- **Resilience**:
    - Ensure that the API gracefully handles unexpected failures (e.g., network issues, service interruptions).
    - Test the API's recovery process after a failure or crash.
- **Retry Mechanisms**:
    - Implement and verify retry logic for handling transient failures.
    - Ensure that retries are properly managed to avoid excessive load or resource exhaustion.

7. Interoperability

- **Compatibility Testing**:
    - Ensure that the API is compatible with different client environments (e.g., browsers, mobile apps).
    - Verify that the API works well with various data formats (e.g., JSON, XML).
- **Third-Party Integration**:
    - Test the API's integration with third-party services and systems.
    - Ensure that API interactions with third-party services are secure and reliable.
- **Backward Compatibility**:
    - Verify that changes to the API do not break existing client integrations.
    - Test older clients against the new API version to ensure continued functionality.

8. Maintainability

- **Code Quality**:
    - Ensure that the API codebase follows best practices for readability, modularity, and documentation.
    - Verify that automated tests (e.g., unit tests, integration tests) are in place and maintained.
- **Continuous Integration/Continuous Deployment (CI/CD)**:
    - Ensure that the API is integrated with a CI/CD pipeline for automated testing and deployment.
    - Verify that deployments are smooth and that rollback procedures are in place.
- **Documentation Maintenance**:
    - Ensure that API documentation is regularly updated with changes and new features.
    - Verify that the API documentation is accessible and easy to understand for developers.

Resources

Friday, August 16, 2024 2:54 PM

Reference: 12 C.F.R. § 748.0 (b)(2)

- [ ] Reviewed ISO/IEC/IEEE 12207:2017 Systems and software engineering — Software life cycle processes

NIST.SP.800-160v1
NIST.SP.800-37r2

https://owasp.org/Top10/

https://github.com/0xRadi/OWASP-Web-Checklist

https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/stable-en/02-checklist/05-checklist

https://owasp.org/www-project-developer-guide/draft/06-design/02-web-app-checklist/00-toc

https://mas.owasp.org/checklists/

https://owasp.org/www-project-api-security/

https://owasp-risk-rating.com/

OWASP_MAS_Check...

OWASP_MASTG

OWASP_Code_Revi...

Comprehensive Review Process for Credit Union's Software Development – Compliance with 12 CFR Part 748

Monday, October 28, 2024 9:06 AM

Stmt 21: Defined Systems/Software Development Life Cycle Process

Objective: Ensure that a robust, secure, and compliant SDLC process is in place.

- **Documentation Review**:
    - Verify a documented SDLC process that clearly outlines phases: planning, design, development, testing, deployment, and maintenance.
    - Ensure documentation complies with industry standards and regulatory requirements, such as 12 CFR Part 748.
- **Compliance Check**:
    - Confirm that the SDLC incorporates secure coding practices and complies with relevant regulations.

Stmt 21.1: Documented Secure Application Development Process

Objective: Ensure a secure application development process is documented, approved, and integrated into the SDLC.

- **Process Documentation**:
    - Review the secure application development documentation to ensure it covers security requirements, coding standards, and secure design principles.
- **Approval and Updates**:
    - Verify approval from appropriate stakeholders and check for a regular review process to update the development process.
- **Integration with SDLC**:
    - Confirm integration of secure application development into the SDLC.

Stmt 21.2: Secure Code Repository and Controls

Objective: Ensure a secure code repository with appropriate access controls and logging capabilities.

- **Repository Review**:
    - Confirm the presence of a secure code repository that supports version control, access control, and audit logging.
- **Access Controls**:
    - Review access control measures to ensure only authorized personnel can check code in and out, with logged and reviewed changes.

- **Security Measures**:
    - Check for encryption and security protocols safeguarding the code repository.

Stmt 21.3: Appropriate Segregation of Duties

Objective: Ensure segregation of duties to prevent conflicts of interest and reduce risks.

- **Role Definitions**:
    - Confirm clearly defined roles and responsibilities within the development process.
- **Access Control**:
    - Review access mechanisms enforcing segregation of duties.
- **Documentation**:
    - Ensure documentation reflects duty segregation and undergoes regular review.

Stmt 21.4: Root Cause Analysis Process

Objective: Establish a root cause analysis process to identify and address code vulnerabilities.

- **Process Documentation**:
    - Review the root cause analysis process, ensuring it includes steps to identify, analyze, and address code vulnerabilities.
- **Implementation**:
    - Verify the root cause analysis process is practiced and implemented effectively.
- **Reporting and Review**:
    - Confirm findings from root cause analyses are documented and reviewed for continuous improvement.

Stmt 21.5: Inventory of Third-Party Components

Objective: Maintain an updated inventory of third-party components used in development, with security and version details.

- **Inventory Review**:
    - Verify an up-to-date inventory of third-party components, including versions and security information.
- **Monitoring and Updates**:
    - Confirm regular updates and monitoring of third-party components for vulnerabilities.
- **Documentation**:

- Ensure documentation accurately reflects third-party components.

Stmt 21.6: Separate Environments for Production and Non-Production Systems

Objective: Ensure production and non-production systems are kept separate.

- **Environment Configuration**:
  - Verify separate environments for production and non-production, ensuring non-production systems are isolated from production.
- **Access Controls**:
  - Review access control settings specific to each environment to ensure proper restrictions.
- **Documentation**:
  - Confirm that documentation reflects environment separation and is regularly updated.

Stmt 21.7: Secure Code Training for Personnel

Objective: Ensure that all development personnel are trained in secure coding specific to their environment.

- **Training Programs**:
  - Verify that secure coding training is provided to all software development personnel, tailored to their roles.
- **Training Records**:
  - Review training records, including completed sessions and participant lists.
- **Ongoing Education**:
  - Ensure training is updated regularly to reflect the latest secure coding practices.

Stmt 21.8: Static and Dynamic Analysis Tools

Objective: Integrate static and dynamic analysis tools into the SDLC to validate secure coding practices.

- **Tool Deployment**:
  - Verify the use of static and dynamic analysis tools within the application life cycle.
- **Usage and Coverage**:
  - Confirm consistent tool usage across development and testing stages, covering secure coding practices.
- **Results and Actions**:

- Review how results from analysis tools are addressed and ensure mitigation of identified vulnerabilities.

Stmt 21.9: Process for Addressing Software Vulnerabilities

Objective: Establish a process for accepting and addressing software vulnerabilities, with a channel for external reporting.

- **Vulnerability Reporting**:
    - Verify the presence of a process for receiving and addressing vulnerability reports, including from external entities.
- **Response and Resolution**:
    - Review procedures for tracking, evaluating, and resolving reported vulnerabilities.
- **Communication**:
    - Confirm a mechanism to notify stakeholders of resolved vulnerabilities.
- **Documentation**:
    - Ensure that all vulnerabilities and actions taken are well-documented and maintained for audit purposes.

Questions

Monday, October 28, 2024 9:07 AM

Comprehensive Review Process for Credit Union's Software Development – Compliance with 12 CFR Part 748

Stmt 21: Defined Systems/Software Development Life Cycle Process (SDLC)

1. **Documentation**:
    - Is there a documented systems/software development life cycle (SDLC) process that outlines all phases, including planning, design, development, testing, deployment, and maintenance?
    - How often is the SDLC documentation reviewed and updated?
    - Does the SDLC incorporate security requirements and controls for each phase?
2. **Compliance**:
    - How does the SDLC process align with industry standards (e.g., OWASP, NIST) and regulatory requirements, including 12 CFR Part 748?
    - Is there a process to ensure that any regulatory updates are incorporated into the SDLC process?

Stmt 21.1: Documented Secure Application Development Process

1. **Process Documentation**:
    - Is there a documented secure application development process that includes security requirements, coding standards, and secure design principles?
    - Who is responsible for maintaining and updating the secure application development process documentation?
2. **Approval and Updates**:
    - Has the secure application development process been formally approved by appropriate stakeholders?
    - How often is the process reviewed and updated to reflect new threats or regulatory changes?
3. **Integration with SDLC**:
    - Is the secure application development process integrated into the overall SDLC?

Stmt 21.2: Secure Code Repository and Controls

1. **Repository Review**:
    - Is there a secure code repository with version control, access control, and audit logging?
    - How are access controls configured for this repository?

2. **Access Controls**:
    - Who is responsible for managing access to the code repository, and how are permissions granted or removed?
    - How are code check-ins and check-outs logged and reviewed?
3. **Security Measures**:
    - What encryption or security measures are in place to protect the code repository from unauthorized access?

Stmt 21.3: Appropriate Segregation of Duties

1. **Role Definitions**:
    - Are roles and responsibilities within the development process clearly defined to ensure segregation of duties?
    - How are conflicts of interest managed within the development and change management processes?
2. **Access Control**:
    - What access control mechanisms are in place to enforce segregation of duties in the software development process?
3. **Documentation**:
    - Is the segregation of duties documented, and how often is it reviewed?

Stmt 21.4: Root Cause Analysis Process

1. **Process Documentation**:
    - Is there a documented process for performing root cause analysis on code vulnerabilities?
    - What steps are included in the root cause analysis process for identifying, analyzing, and remediating vulnerabilities?
2. **Implementation**:
    - How is the root cause analysis process implemented in practice?
3. **Reporting and Review**:
    - Are findings from root cause analyses documented, reported, and reviewed for continuous improvement?

Stmt 21.5: Inventory of Third-Party Components

1. **Inventory Review**:
    - Is there an updated inventory of third-party components used in software development, including version information and security status?

- How frequently is this inventory reviewed?

2. **Monitoring and Updates**:
    - What process is in place to monitor third-party components for vulnerabilities and update them when necessary?
3. **Documentation**:
    - Is there documentation detailing the inventory of third-party components, and who is responsible for maintaining it?

Stmt 21.6: Separate Environments for Production and Non-Production Systems

1. **Environment Configuration**:
    - Are production and non-production environments separated and properly configured to prevent unauthorized access?
    - How are non-production systems isolated from production systems?
2. **Access Controls**:
    - What access controls are enforced in each environment, and who has access to each environment?
3. **Documentation**:
    - Is the separation of environments documented, and how often is this documentation reviewed?

Stmt 21.7: Secure Code Training for Personnel

1. **Training Programs**:
    - Are all software development personnel provided training in secure coding specific to their development environment and responsibilities?
    - How is the content of secure code training aligned with the development environment?
2. **Training Records**:
    - Are training records maintained for all personnel who complete secure coding training?
    - Who is responsible for ensuring that personnel complete secure coding training?
3. **Ongoing Education**:
    - How often is secure code training updated, and are there refresher courses for employees?

Stmt 21.8: Static and Dynamic Analysis Tools

1. **Tool Deployment**:

- Are static and dynamic analysis tools integrated into the development life cycle to validate secure coding practices?
- How are these tools deployed, and at what stages of development are they used?

2. **Usage and Coverage**:
    - How frequently are these tools used, and what areas of secure coding practices do they cover?
3. **Results and Actions**:
    - How are vulnerabilities identified by these tools addressed, and what is the process for remediating issues?

Stmt 21.9: Process for Addressing Software Vulnerabilities

1. **Vulnerability Reporting**:
    - Is there a defined process for receiving and addressing vulnerability reports, both internally and externally?
    - How is information about the process communicated to external parties?
2. **Response and Resolution**:
    - What is the process for evaluating, tracking, and resolving reported vulnerabilities?
3. **Communication**:
    - How are stakeholders informed of vulnerabilities and their resolution status?
4. **Documentation**:
    - Is there a system for documenting reported vulnerabilities, actions taken, and resolutions?

Answers

Monday, October 28, 2024 9:08 AM

Stmt 21: Defined Systems/Software Development Life Cycle Process (SDLC)

1. **Documentation**:
    - **Positive**:
        - *Yes, a documented SDLC process outlines each phase, covering planning, design, development, testing, deployment, and maintenance.*
        - *Reviewed annually for updates, aligning with industry standards.*
    - **Negative**:
        - *No, there is no formal SDLC documentation available.*
        - *Documented phases lack security requirements, creating compliance gaps.*
2. **Compliance**:
    - **Positive**:
        - *Yes, the SDLC aligns with industry standards (e.g., NIST, OWASP) and regulatory requirements.*
        - *Updates are made following regulatory changes to maintain compliance.*
    - **Negative**:
        - *No, the SDLC does not adhere to any specific regulatory standards.*
        - *There is no process in place to update the SDLC for regulatory compliance.*

Stmt 21.1: Documented Secure Application Development Process

1. **Process Documentation**:
    - **Positive**:
        - *Yes, a secure application development process is documented, including security requirements, coding standards, and secure design principles.*
    - **Negative**:
        - *No, there is no documented secure application development process.*
        - *Coding standards lack security specifications.*
2. **Approval and Updates**:
    - **Positive**:

        - *Yes, the secure application development process is approved by stakeholders and reviewed semi-annually.*
    - **Negative**:
        - *No formal approval process exists.*
        - *The process is rarely or never updated.*
3. **Integration with SDLC**:
    - **Positive**:
        - *Yes, secure application development is integrated into the SDLC with checkpoints at each phase.*
    - **Negative**:
        - *No, the secure application development process operates independently of the SDLC.*

Stmt 21.2: Secure Code Repository and Controls

1. **Repository Review**:
    - **Positive**:
        - *Yes, a secure code repository with version control, access control, and audit logging is in place.*
    - **Negative**:
        - *No, there is no secure code repository, and code control is managed manually.*
2. **Access Controls**:
    - **Positive**:
        - *Yes, access is restricted, with logging and regular review of all code check-ins.*
    - **Negative**:
        - *Access is not well-controlled, and no logging exists for code changes.*
3. **Security Measures**:
    - **Positive**:
        - *Yes, the code repository is encrypted and safeguarded with additional security layers.*
    - **Negative**:
        - *No encryption is applied to the repository, and it lacks sufficient security measures.*

Stmt 21.3: Appropriate Segregation of Duties

1. **Role Definitions**:
    - **Positive**:
        - *Roles and responsibilities are clearly defined, minimizing conflicts of interest.*
    - **Negative**:
        - *Roles are not well-defined, creating potential conflicts in responsibilities.*
2. **Access Control**:
    - **Positive**:
        - *Access controls enforce strict segregation, ensuring no overlap in duties.*
    - **Negative**:
        - *Access controls are weak, allowing some roles to overlap unnecessarily.*
3. **Documentation**:
    - **Positive**:
        - *Documentation on segregation of duties is up-to-date and reviewed annually.*
    - **Negative**:
        - *There is little or no documentation supporting the segregation of duties.*

Stmt 21.4: Root Cause Analysis Process

1. **Process Documentation**:
    - **Positive**:
        - *A well-defined root cause analysis process is documented, covering identification and remediation of vulnerabilities.*
    - **Negative**:
        - *No formal root cause analysis process is documented.*
2. **Implementation**:
    - **Positive**:
        - *The root cause analysis process is routinely implemented, with documented outcomes.*
    - **Negative**:
        - *The process is rarely or never followed in practice.*

3. **Reporting and Review**:
    - **Positive**:
        - *Findings are regularly reported, reviewed, and used for process improvement.*
    - **Negative**:
        - *Findings are either not documented or not reviewed for improvement.*

Stmt 21.5: Inventory of Third-Party Components

1. **Inventory Review**:
    - **Positive**:
        - *An up-to-date inventory of third-party components is maintained with version and security details.*
    - **Negative**:
        - *No comprehensive inventory exists for third-party components.*
2. **Monitoring and Updates**:
    - **Positive**:
        - *Third-party components are routinely monitored and updated for vulnerabilities.*
    - **Negative**:
        - *Third-party components are rarely monitored or updated.*
3. **Documentation**:
    - **Positive**:
        - *The documentation accurately reflects the current third-party components.*
    - **Negative**:
        - *Documentation is outdated and lacks recent component details.*

Stmt 21.6: Separate Environments for Production and Non-Production Systems

1. **Environment Configuration**:
    - **Positive**:
        - *Production and non-production environments are fully isolated.*
    - **Negative**:
        - *Production and non-production systems are not separated, risking data integrity.*

2. **Access Controls**:
    - **Positive**:
        - *Access is tightly controlled, and access to production is limited to essential personnel only.*
    - **Negative**:
        - *Access controls are lax, and non-production systems have excessive access.*
3. **Documentation**:
    - **Positive**:
        - *Environment separation is fully documented and reviewed periodically.*
    - **Negative**:
        - *Documentation is minimal or non-existent on environment separation.*

Stmt 21.7: Secure Code Training for Personnel

1. **Training Programs**:
    - **Positive**:
        - *All development personnel receive comprehensive training tailored to their roles.*
    - **Negative**:
        - *Training is minimal and not tailored to specific roles.*
2. **Training Records**:
    - **Positive**:
        - *Training completions are documented and reviewed.*
    - **Negative**:
        - *Training records are inconsistent or non-existent.*
3. **Ongoing Education**:
    - **Positive**:
        - *Regular refresher courses are held to update personnel on secure coding practices.*
    - **Negative**:
        - *Training is sporadic and lacks updates.*

Stmt 21.8: Static and Dynamic Analysis Tools

1. **Tool Deployment**:
    - **Positive**:
        - *Static and dynamic tools are deployed throughout the development life cycle.*
    - **Negative**:
        - *Tools are either not used or used inconsistently.*
2. **Usage and Coverage**:
    - **Positive**:
        - *Tools are applied consistently, covering all relevant secure coding aspects.*
    - **Negative**:
        - *Coverage is partial, leaving some code segments untested.*
3. **Results and Actions**:
    - **Positive**:
        - *Results are reviewed, with vulnerabilities remediated immediately.*
    - **Negative**:
        - *Results are ignored or not acted upon promptly.*

Stmt 21.9: Process for Addressing Software Vulnerabilities

1. **Vulnerability Reporting**:
    - **Positive**:
        - *A clear process exists for internal and external vulnerability reporting.*
    - **Negative**:
        - *No structured reporting process is in place.*
2. **Response and Resolution**:
    - **Positive**:
        - *Reported vulnerabilities are tracked, evaluated, and resolved promptly.*
    - **Negative**:
        - *Vulnerabilities often go untracked or unresolved.*
3. **Communication**:
    - **Positive**:
        - *Stakeholders are informed on vulnerability status regularly.*

- **Negative**:
    - *Communication is limited or inconsistent.*

4. **Documentation**:
    - **Positive**:
        - *Each vulnerability is documented, along with remediation actions.*
    - **Negative**:

Documentation is inconsistent, leading to incomplete vulnerability

12 CFR Part 748 – Security Program, Report of Suspected Crimes, Suspicious Transactions, Catastrophic Acts, and Bank Secrecy Act Compliance

Monday, October 28, 2024 9:08 AM

- **§ 748.0 Security Program Requirements**:
    - This section requires credit unions to implement a written security program that protects against unauthorized access to or use of member information, which is foundational to secure software development practices.
    - It mandates the creation of safeguards for physical and technical security, directly impacting secure application development, access controls, and data protection requirements within software.
- **§ 748.1 Filing Requirements**:
    - Requires a report to the credit union's board on the implementation of the information security program, which encompasses the software development lifecycle (SDLC) for any systems handling member data.
    - This impacts practices like secure code training, regular vulnerability assessments, and maintaining current inventories of software and third-party components.
- **Appendix A to Part 748 – Guidelines for Safeguarding Member Information**:
    - **III. Develop and Implement the Information Security Program**: Credit unions must establish security policies and practices that are designed to protect member information, which includes secure software development processes.
    - **B. Manage and Control Risk**:
        - Credit unions must develop controls around access to systems, data encryption, and secure data handling within software, aligning with statements like maintaining secure code repositories, segregation of duties, and root cause analysis in development.
    - **C. Security Awareness and Training**:
        - Personnel must be trained in security practices, which includes secure coding training (Stmt 21.7) to minimize vulnerabilities from human error.
    - **D. Testing and Monitoring**:
        - Requires regular testing and monitoring of systems, including the use of static and dynamic code analysis tools (Stmt 21.8), to verify adherence to secure coding standards.
    - **F. Oversight of Service Provider Arrangements**:
        - Credit unions must maintain an inventory of third-party components and conduct due diligence on software from third-

party providers, aligning with Stmt 21.5.

- **Appendix B to Part 748 – Response Programs for Unauthorized Access to Member Information and Member Notice**:
    - Requires credit unions to develop an incident response program that includes processes for identifying and addressing software vulnerabilities (Stmt 21.9).
    - Emphasizes the need for continuous monitoring, vulnerability response protocols, and maintaining channels for external vulnerability reports, which supports secure SDLC practices.

2. 12 CFR Part 749 – Records Preservation Program and Appendices

- **§ 749.0 Records Preservation Program Requirements**:
    - Mandates that credit unions implement secure record preservation practices, ensuring that data integrity and confidentiality are maintained, which directly affects software development practices, particularly around secure code repositories, access controls, and environment separation (Stmt 21.2 and Stmt 21.6).
    - Also requires maintaining a backup and recovery program, impacting software design for data redundancy and secure data handling in development.
- **Appendix A to Part 749 – Record Retention Guidelines**:
    - Establishes guidelines for the preservation and disposition of records, influencing software development to include controls for data retention, access, and secure disposal, which aligns with policies and practices for data management in development environments.

3. 12 CFR Part 717 – Fair Credit Reporting

- **Subpart J – Identity Theft Red Flags**:
    - Requires procedures to identify, detect, and respond to identity theft risks, which impacts software development requirements for logging, monitoring, and detecting suspicious activity. This aligns with requirements for implementing analysis tools within the SDLC (Stmt 21.8).
- **Appendix A to Part 717 – Interagency Guidelines on Identity Theft Detection, Prevention, and Mitigation**:
    - Recommends guidelines that credit unions apply to prevent identity theft, which impacts secure coding practices and vulnerability response protocols within the SDLC to prevent unauthorized access or breaches.

Notes

Thursday, May 1, 2025 10:10 AM

OWASP Code Review Checklist

The OWASP Code Review Guide v2 provides a comprehensive checklist for secure code review. Key points include:

- Data Validation: Ensuring all input is validated, sanitized, and escaped.
- Authentication: Verifying that authentication mechanisms are robust and secure.
- Session Management: Ensuring sessions are managed securely.
- Authorization: Confirming that access controls are properly implemented.
- Cryptography: Using strong cryptographic algorithms and managing keys securely.
- Error Handling: Ensuring that error messages do not leak sensitive information.
- Logging: Implementing secure logging practices.
- Configuration Management: Ensuring secure configuration settings.
- Data Protection: Protecting sensitive data at rest and in transit.
- Code Quality: Ensuring code is maintainable, readable, and follows best practices.

Software Development Life Cycle Standard

The Software Development Life Cycle Standard establishes a consistent approach to software development and quality assurance. Key sections include:

- Requirements Gathering: Using the Jira ticketing process to gather detailed requirements from business partners.
- Design Review: Creating and reviewing design documents before coding, following industry-standard design principles.
- Development: Focusing on quality and alignment with organizational needs, conducting feasibility studies, and documenting requirements.
- CI/CD Process: Following Patelco Credit Union's Continuous Improvement and Continuous Development process for seamless integration and efficient deployment.
- Code Reviews: Ensuring all custom application code changes undergo review, adhering to secure coding guidelines.
- Quality Assurance Testing: Involving the QA team early in the project, creating test case scripts, and using automated tests.
- Deployment and Documentation: Preparing comprehensive deployment plans, maintaining thorough documentation, and conducting retrospective meetings for continuous improvement [1].

Software Asset Management and EOL SOP

The (2Software Asset Management and EOL SOP 1 (2) outlines procedures for managing software that has reached its End of Life (EOL). Key points include:

- Governance and Rules: Establishing clear policies and processes for EOL software management, maintaining accurate software inventory, and implementing risk management strategies.
- Scope and Audience: Applicable to all authorized software within the organization, targeting the Vendor Management team, Technology Department, Asset Management Specialist, and Cyber Security team.
- Procedures: Determining EOL based on vendor support/contract dates, updating the application list in Oomnitza, and performing bi-annual reviews of critical applications without recorded EOL dates.
- Responsibilities: Asset Management Specialist, Software Owners, Cyber Security team, and Vendor Management team have specific roles in maintaining software inventory, assessing risks, and

ensuring compliance
[2]
.

Comparison and Gaps

1. Data Validation:
- OWASP: Emphasizes the importance of validating, sanitizing, and escaping all input.
- SDLC Standard: Mentions validating input to prevent user data from altering commands and queries
[1]
.
- EOL SOP: Does not address data validation.
Recommendation: Ensure the SDLC Standard includes detailed guidelines on data validation, sanitization, and escaping for all input sources.
2. Authentication and Authorization:
- OWASP: Focuses on robust authentication mechanisms and proper access controls.
- SDLC Standard: Includes authentication and encryption of sensitive communications
[1]
.
- EOL SOP: Does not address authentication and authorization.
Recommendation: Incorporate specific guidelines on authentication and authorization in the SDLC Standard, ensuring robust mechanisms are in place.
3. Session Management:
- OWASP: Ensures sessions are managed securely.
- SDLC Standard: Does not explicitly mention session management.
- EOL SOP: Does not address session management.
Recommendation: Add session management guidelines to the SDLC Standard, ensuring secure handling of sessions.
4. Cryptography:
- OWASP: Uses strong cryptographic algorithms and manages keys securely.
- SDLC Standard: Mentions using strong cryptographic algorithms and keys
[1]
.
- EOL SOP: Does not address cryptography.
Recommendation: Ensure the SDLC Standard includes detailed guidelines on cryptographic practices, including key management.
5. Error Handling:
- OWASP: Ensures error messages do not leak sensitive information.
- SDLC Standard: Mentions using generic error messages to avoid leaking information
[1]
.
- EOL SOP: Does not address error handling.
Recommendation: Expand the SDLC Standard to include comprehensive error handling practices, ensuring no sensitive information is leaked.
6. Logging:
- OWASP: Implements secure logging practices.
- SDLC Standard: Does not explicitly mention logging.
- EOL SOP: Does not address logging.
Recommendation: Add logging guidelines to the SDLC Standard, ensuring secure and comprehensive logging practices.
7. Configuration Management:
- OWASP: Ensures secure configuration settings.
- SDLC Standard: Does not explicitly mention configuration management.
- EOL SOP: Does not address configuration management.

Recommendation: Include configuration management guidelines in the SDLC Standard, ensuring secure settings are maintained.

8. Data Protection:
- OWASP: Protects sensitive data at rest and in transit.
- SDLC Standard: Mentions encrypting sensitive communications [1].
- EOL SOP: Does not address data protection.
  Recommendation: Ensure the SDLC Standard includes comprehensive data protection guidelines, covering both data at rest and in transit.

9. Code Quality:
- OWASP: Ensures code is maintainable, readable, and follows best practices.
- SDLC Standard: Emphasizes code reviews, proper formatting, and secure coding guidelines [1].
- EOL SOP: Does not address code quality.
  Recommendation: Maintain the emphasis on code quality in the SDLC Standard and ensure it aligns with OWASP guidelines.

Conclusion

The Software Development Life Cycle Standard aligns well with many aspects of the OWASP Code Review Checklist, but there are areas for improvement. The (2Software Asset Management and EOL SOP 1 (2) does not address many of the security aspects covered by OWASP. By incorporating the recommendations above, Patelco Credit Union can enhance its software development and asset management practices, ensuring a more secure and robust approach.

From <https://outlook.office.com/hosted/semanticoverview/Users('OID:1bd7671d-2a8e-430e-883a-eb6186b3b616@7db3915c-28e1-4949-b6e1-30b66f6612a8')?culture=en-US&hostName=Office&hostClientType=desktop&referrer=copilot&navigationType=direct&sessionId=10c98937-0bea-4cb2-9aef-6e2f5bfc7fc0&appState=Active&osType=Win11&hwaVersion=V2&hostApp=hub&isanonymous=true&worktab=1&webtab=0>