Secure Coding Review.

- Title: Secure Coding Review in Python Flask Applications.
- Subtitle:
- Best Practices and Vulnerability Mitigation

- Presenter Name; Saif Ali laghari

- Date: 20/8/2024

Introduction;

- What is Secure Coding?
- Definition: Writing software with a focus on preventing security vulnerabilities.
- Importance: Mitigates risks like data breaches, unauthorized access, and cyber attacks.
- Objective of the Presentation
- Reviewing a Flask application for common security vulnerabilities.
- Providing secure coding practices and solutions.

Overview of the Sample Flask Application

- Key Components:
- User Authentication System
- Session Management
- User Profiles
- Vulnerabilities Discussed:
- SQL Injection
- Cross-Site Scripting
- (XSS)Cross-Site Request Forgery (CSRF)
- Insecure Deserialization
- Insecure Configuration

SQL Injection Vulnerability;

- Problem:
- User input directly embedded into SQL queries.
- Example:
- query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'"
- Recommendation:
- Use parameterized queries to prevent SQL injection.
- Example of a secure implementation:
- cursor.execute("SELECT * FROM users WHERE username = ? AND password = ?", (username, password))

Hardcoded Secret Key;

- Problem:
- Secret key is hardcoded, which can be exposed if the code is leaked.
- Recommendation:
- Use environment variables or configuration files to store secret keys securely.
- import os app.secret_key = os.environ. get('SECRET_KEY   ', 'defaultsecret')

Cross-Site Scripting (XSS)

- Problem
- User input directly rendered in HTML without proper escaping.
- Example:
- <h1>{{ username }}</h1>
- Recommendation:
- Always escape and sanitize user-generated content:
- <h1>{{ username | e }}</h1> <!-- Escaping user input -->

Cross-Site Request Forgery (CSRF);

- Problem:
- No CSRF protection for forms, making the app vulnerable to unwanted requests.
- Recommendation:
- Use Flask-WTF to automatically add CSRF protection to forms:
- from flask_wtf.csrf import CSRFProtect
- csrf = CSRFProtect(app)

Insecure Deserialization

- Problem:
- Using signed cookies for session data can expose sensitive information if the secret key is compromised.
- Recommendation:
- Move session data to server-side storage like Redis or databases.
- Example:
- Implement Flask-Session to store session data securely on the server

Insecure Configuration;

- Problem:
- Application runs in debug mode in production, exposing sensitive details on errors.
- Recommendation:
- Disable debug mode in production environments:
- if __name__ == '__main__':

- app.run(debug=os.getenv('FLASK_DEBUG', False))

Tools for Automated Security Review

- Bandit: Static code analysis tool for Python security issues.
- Example command: bandit -r your_flask_app_directory/
- Flake8 with Security Plugins:
- Extend Flake8 with plugins to check for security vulnerabilities.
- SonarQube: Comprehensive code quality and security analysis tool.
- PyLint with Security Extensions: Use PyLint with security-focused plugins.

Conclusion;

- Summary:
- Secure coding is essential to prevent vulnerabilities in applications.
- Key areas to focus on: Input validation, session management, configuration security, and proper use of libraries.
- Next Steps:
- Regular code reviews, integrating static analysis tools, and following best practices can help ensure a secure application.