5.2. Student Handout

Secure Coding Principles: Student Handout

Introduction to Secure Coding

What is Secure Coding?

Secure coding involves writing software to protect it from vulnerabilities and attacks. It ensures that your software is resistant to hackers and malicious users.

Importance of Secure Coding

- Software is ubiquitous, increasing the risk of cyberattacks.
- Hackers exploit vulnerabilities to steal data or disrupt services.
- Secure coding prevents these attacks by building security into software from the start.

Common Security Vulnerabilities

1. SQL Injection

- **Definition**: Manipulation of database queries by inserting malicious code.
- **Example 1**: An attacker enters `' OR '1'='1` in a login form to bypass authentication.
- **Example 2**: Using unsanitized user input in a search query can expose sensitive data.
- **Example 3**: An attacker modifies a URL parameter to execute unauthorized database commands.

2. Cross-Site Scripting (XSS)

- **Definition**: Injection of malicious scripts into a website, executed by users.
- **Example 1**: An attacker injects a script in a comment section that steals cookies.
- **Example 2**: A malicious script in a forum post redirects users to a phishing site.
- **Example 3**: An attacker uses XSS to display fake login forms to capture credentials.

3. Cross-Site Request Forgery (CSRF)

- **Definition**: Tricks users into performing actions they didn’t intend.

- **Example 1**: A malicious link causes a user to unknowingly transfer money.
- **Example 2**: An attacker sends a request to change a user's email address.
- **Example 3**: A hidden form submission changes account settings without user consent.

4. Insecure Authentication

- **Definition**: Weak login mechanisms that allow unauthorized access.
- **Example 1**: Using weak passwords without enforcing complexity requirements.
- **Example 2**: Lack of multi-factor authentication for sensitive accounts.
- **Example 3**: Session IDs not being invalidated after logout, allowing session hijacking.

The OWASP Top 10 Security Risks

1. **Injection (e.g., SQL Injection)**
2. **Broken Authentication**
3. **Sensitive Data Exposure**
4. **XML External Entities (XXE)**
5. **Broken Access Control**
6. **Security Misconfiguration**
7. **Cross-Site Scripting (XSS)**
8. **Insecure Deserialization**
9. **Using Components with Known Vulnerabilities**
10. **Insufficient Logging & Monitoring**

How CodeGuru Reviewer Helps with Secure Coding

Identifying Security Vulnerabilities

- **Example 1**: Flags SQL injection vulnerabilities and suggests input sanitization.
- **Example 2**: Detects insecure authentication practices and recommends improvements.
- **Example 3**: Identifies XSS risks and advises on proper input encoding.

Reviewing Security Recommendations

- CodeGuru provides detailed feedback on fixing identified security issues.

Integrating Secure Coding Practices into DevOps

DevSecOps

- Integrate security into every stage of the development process.
- Catch security issues early and fix them before deployment.

Automating Security Checks

- Use CodeGuru in your CI/CD pipeline to automatically scan for vulnerabilities.

Combining CodeGuru with Other AWS Security Tools

- **AWS Inspector**: Assesses applications for vulnerabilities.
- **AWS GuardDuty**: Monitors AWS environments for malicious activity.

Best Practices for Secure Development

1. Input Validation

- **Example 1**: Validate email addresses to ensure they follow the correct format.
- **Example 2**: Check that numeric inputs are within expected ranges.
- **Example 3**: Use regular expressions to validate complex input patterns.

2. Error Handling

- **Example 1**: Display generic error messages to users.
- **Example 2**: Log detailed error information for internal use only.
- **Example 3**: Avoid exposing stack traces in production environments.

3. Encryption

- **Example 1**: Encrypt passwords using strong hashing algorithms.
- **Example 2**: Use HTTPS to encrypt data in transit.
- **Example 3**: Encrypt sensitive data stored in databases.

Writing Secure APIs and Handling Sensitive Data Safely

- **Authentication**: Use strong mechanisms like OAuth or JWT.
- **Authorization**: Ensure users access only authorized resources.
- **Data Encryption**: Encrypt sensitive data, such as personal or financial information.

Hands-On: Applying Secure Coding Techniques and Using CodeGuru

Step 1: Write a Simple Web Application

- Create a web application with a login feature.

Step 2: Introduce a Vulnerability

- Intentionally add an SQL injection vulnerability by not validating input.

Step 3: Run CodeGuru

- Use CodeGuru to identify the vulnerability and receive recommendations.

Step 4: Fix the Vulnerability

- Implement CodeGuru's recommendations to secure the application.

Conclusion

Secure coding is crucial in modern software development. By following best practices, using tools like CodeGuru, and integrating security into your DevOps pipeline, you can build robust software that protects user data and resists attacks.