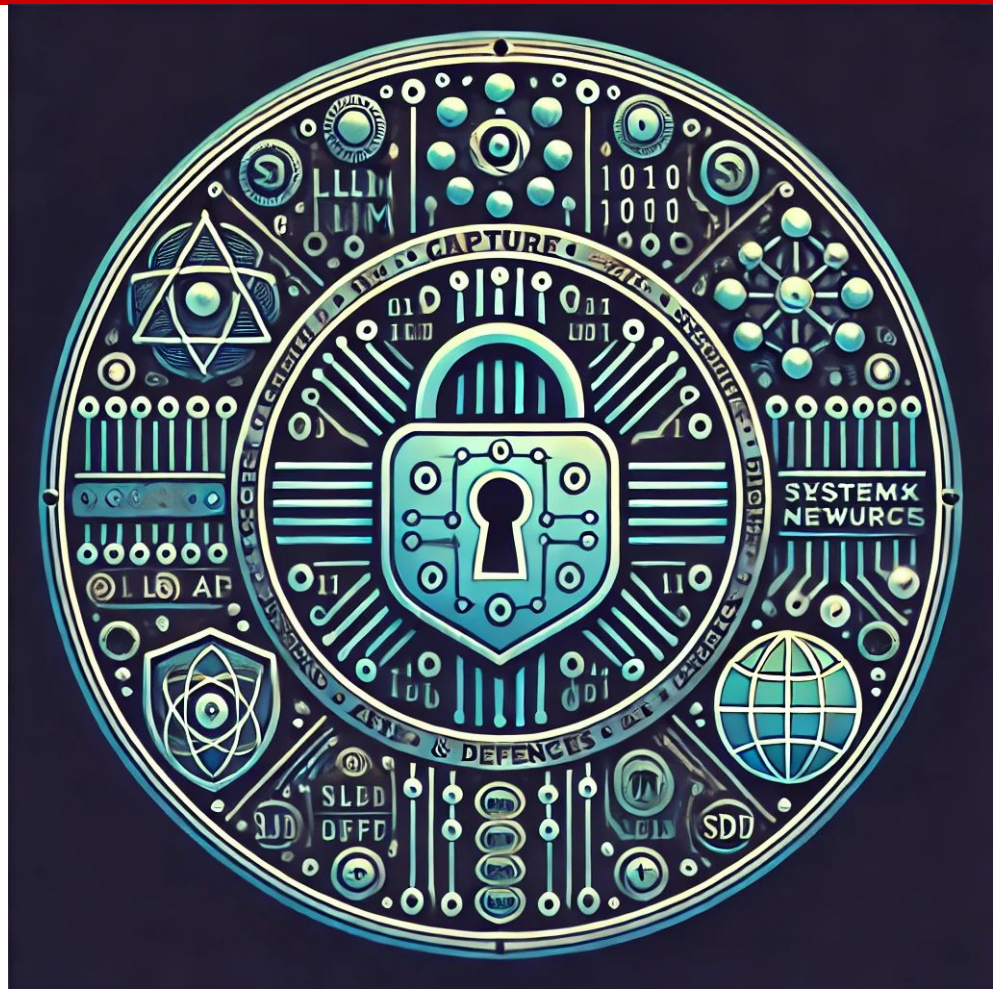
CSC-537
Systems Attacks and Defenses

Secure Coding Principles

Alexandros Kapravelos
akaprav@ncsu.edu



Least Privilege

- Grant users, processes, and systems only the absolute minimum permissions necessary to perform their required functions. This limits the potential damage from a successful attack or an error.
- Example: A web application should not have database administrator privileges.

Example

Give me a dockerfile that runs nginx

What user does it run it as?

Input Validation

- Never trust data received from external sources (users, other systems, APIs, etc.). Thoroughly validate all input for type, length, format, and range before using it.
- Example: Check that an email address field actually contains a valid email address format.

Output Encoding

- Properly encode or escape data before sending it to another component or system. The encoding method should be appropriate for the receiving context (e.g., HTML, JavaScript, SQL).
- Example: Encoding special characters like < and > as < and > when displaying user-supplied data in a web page prevents Cross-Site Scripting (XSS) attacks.

Secure Error Handling

- Handle errors gracefully without revealing sensitive information that could aid attackers. Log error details securely for debugging purposes, but present generic error messages to users.
- Example: Instead of displaying a detailed SQL error message, show a user-friendly message like "An unexpected error occurred. Please try again later."

Defense in Depth

- Implement multiple layers of security controls. If one layer fails, others are in place to provide protection.
- Example: Combining firewalls, intrusion detection systems, input validation, strong authentication, and access controls.

Fail Securely

- Design systems to fail in a way that prioritizes security. This means defaulting to a secure state in case of errors or unexpected conditions.
- Example: If a system component fails, ensure it doesn't leave data unprotected or allow unauthorized access. If an authentication module cannot reach the authentication server, don't permit access by default.

Keep it Simple

- Simple code is easier to understand, review, and maintain. Complexity increases the likelihood of security vulnerabilities.
- Example: Avoid overly complicated algorithms or convoluted code structures.

Secure Defaults

- Configure systems and applications with secure settings out-of-the-box. Users should not have to manually enable security features.
- Example: Require strong passwords by default, enable logging of security-relevant events, and disable unnecessary services.

Data Protection

Protect Data in Transit

- Use strong encryption protocols (e.g., TLS/SSL) to protect data transmitted over networks, especially the internet.
- Example: Always use HTTPS for websites that handle sensitive information.

Protect Data at Rest

- Encrypt sensitive data stored in databases, files, or other storage media.
- Example: Use database encryption or file system encryption to protect data even if physical access to the storage is compromised.

Proper Session Management

- Securely manage user sessions to prevent session hijacking and other related attacks.
- Example: Use strong session ID generation, set appropriate session timeouts, and invalidate sessions upon logout.

Authentication and Authorization

Strong Authentication

- Implement robust authentication mechanisms to verify user identities.
- Example: Use strong, unique passwords. Enforce multi-factor authentication (MFA) whenever possible.

Secure Password Storage

- Never store passwords in plain text. Use strong, one-way hashing algorithms (e.g., bcrypt, Argon2) with salting to protect stored passwords.
- Example: Salt each password with a unique, random value before hashing it.

Code Quality and Maintenance

Regular Security Testing

Conduct regular security assessments, including vulnerability scanning, penetration testing, and code reviews, to identify and remediate security weaknesses.

Example: Use automated scanning tools and schedule periodic manual security audits.

Keep Software Up-to-Date

Regularly update all software components (operating systems, libraries, frameworks) to patch known vulnerabilities.

Example: Enable automatic updates or have a process for promptly applying security patches.

Secure Development Lifecycle (SDL)

- Integrate security into every phase of the software development process, from design to deployment and maintenance.
- Example: Perform threat modeling during the design phase, conduct security code reviews during development, and perform penetration testing before release.

Operational Security

Principle of Separation of Duties

Divide critical tasks among multiple individuals to prevent fraud or errors.

Example: The person who approves a financial transaction should not be the same person who initiates it.

Auditing and Logging

Log security-relevant events (e.g., authentication attempts, authorization decisions, system errors) for monitoring, incident response, and auditing purposes.

Example: Log failed login attempts to detect brute-force attacks.

Use Secure Libraries and Frameworks

Leverage well-established and security-focused libraries and frameworks to avoid reinventing the wheel and potentially introducing vulnerabilities.

Example: Use a reputable web framework that handles common security tasks like input validation and output encoding.

Can LLMs write better code if you keep asking them to “write better code”?

source: https://minimaxir.com/2025/01/write-better-code/