CompTIA Security + 3.0 Architecture and Design

Filename: comptia-secplussy0501-3-6-secure_application_development_and_deployment
Title: Secure Application Development and Deployment
Subtitle: CompTIA Security+ (SY0-501)

3.6 Secure Application Development and Deployment

- 3.6 Summarize secure application development and deployment concepts
    - Development life-cycle models
        - Waterfall
            - Sequential Model
            - The whole process is divided into phases
            - Each phase needs to be completed before the next phase can begin
            - Use where requirements remain unchanged
            - Simple to implement small projects
            - Delivery of the final project is late
            - Very diffcult to move back to earlier phases
            - *Step 1* = Requirement Analysis
                - All requirements are captured in this phase
                - Walkthroughs, brainstorming to understand the requirements
                - Produce requirements documentation
            - *Step 2* = System Design
                - Capturing hardware and software requirements
                - Defining the system architecture
                - Produces the design documentation
            - *Step 3* = Implementation
                - Create the code in small programs called units
                - Unit testing of the code
                - Produce unit test cases and the results of the tests
            - *Step 4* = Testing (and integration)
                - Integration of tested units
                - Ensuring that units work as expected
                - Document any anomalies
                - Produce test cases and test reports
            - *Step 5* = Deployment
                - Performed after function and non-functional testing is finished
                - Making sure the environment is up and running
                - deploy to the market or the customer environment
                - Produce environment definitions or specifications
            - *Step 6* = Maintenance
                - Ensuring the application is up ad running
                - Application enhancements to incorporate more functionality or features
                - If users experience issues document and fix those issues
                - Produces fixes issues and creating a list of new features.
        - Agile
            - Treats every project differently by dividing tasks into "small timeframes"
            - Agile is an adaptive approach
            - Resource requirements are minimized
            - Delivers partial working solutions early on in the SDLC
            - Not suitable for handling complex dependencies
    - Secure DevOps
        - Security automation
        - Continuous integration
            - a development practice that requires development teams to store code changes into a central repository using version controls systems like Git
            - All changes are isolated and tested immediately through automatic builds
            - Easier to spot errors in code and correct them as soon as possible
        - Baselining
            - allows you to build a business case where you can apply targets, goals and measure the level of progress
        - Immutable systems
            - Components are replaced not changed
            - Applicationa and services a redeployed not reconfigured when a change occurs
        - Infrastructure as code
            - Treats infrastructure as software that can be managed with tools that software developers use.
            - Infrastructure changes are easily, faster while maintaining reliability
    - Version control and change management
        - Most software developers work in teams and are constantly writing changes to code
        - VC sofware maintains records of changes made to code
        - If mistakes are made or if bugs are found the developers can rollback to a previous version for comparison and correction.

        - This minimizes interruptions and elliminates file locking
    - Provisioning and deprovisioning
    - Secure coding techniques
        - Proper error handling
        - Proper input validation
        - Normalization
            - Data on the backend might be expecting say uppercase letters but a user enters lowercase letters on the frontend
            - When the data reachs the backend an unexpected value is retrieved and causes issues
            - The data is received from the front end in whatever state it may be run through a program that "normalizes" the data into the values expected on the back end.
        - Stored procedures
            - A group of SQL statements that form a logical group to perform a task, which can be locked down to prevent SQL injection attacks
        - Code signing
        - Encryption
        - Obfuscation/camouflage
            - Code might be able to easily be read, obfuscating the code makes it harder to disassemble or understand the purpose of the code
        - Code reuse/dead code
            - The reuse of code in software can carry over flaws, vulnerabilites. Open Web Application Security Project (OWASP) names it as part of the 10 application vulnerabilites
            - Dead code are sections of codes that results are never used by the program wasting processing time and causing poor quality.
        - Server-side vs. client-side execution and validation
            - Validating data on the backend with server-side scripting languages(ASP.Net or PHP), then feedback is sent back to the client. Server-side validation is slower but more secure.
            - Validating on the client-side is when the web browser does the validation prior to sending the data to the server. Performance is better but less secure
        - Memory management
        - Use of third-party libraries and SDKs
            - Third-party libraries can contain security vulnerabilities, flaws and security issues that are not recognized
        - Data exposure
    - Code quality and testing
        - Static code analyzers
            - allows for the analyzation of computer software without executing the code
            - Examples
                - Google CodePro Analytix
                - VisualCodeGrepper
                - OWASP Lapse+
                - RIPS
                - DebBug
        - Dynamic analysis (e.g., fuzzing)
            - Testing and evaluating a program through real-time execution
            - Unit testing is an example of dynamic testing
            - Identifying vulnerabilities and dependencies
            - Identifying errors, error handling and defects
        - Stress testing
            - Allows a developer the ability to test the effectiveness of a program under unfavorable conditions
            - Measuring errors, crashes
            - Ellimination of unpredictablity
        - Sandboxing
            - Experimenting with code in an isolated environment away from the production environment
        - Model verification
    - Compiled vs. runtime code