8.6.3 Application Development Security Facts

Secure coding concepts include the following:

Concept	Description
Error and Exception Handling	*Error and exception handling* is a programming language construct designed to handle the occurrence of exceptions. *Exceptions* are special conditions that change the normal flow of a program's execution. An exception is handled (resolved) by: 1. Saving the current state of execution in a predefined place 2. Switching the execution to a specific subroutine known as an exception handler Application code is exception-safe if run-time failures within the code do not produce ill effects such as memory leaks, garbled stored data, or invalid output.
Input Validation	*Input validation* is the process of ensuring that a program operates on clean, correct, and useful data. Input validation: ▪ Prevents data corruption or a security vulnerability. ▪ Uses routines (also called validation rules or check routines) that check for correctness, meaningfulness, and security in data that is input to the system. Validations can be performed on the server side or on the client side. *Server-side validation* is user input validation that takes place on the server side during a postback session. You use server-side validation if the user request requires server resources to validate the user input. *Client-side validation* is user input validation that takes place on the client side (web browser). Client-side validation does not require a postback. You use client-side validation if the user request does not require any server resources to validate the input.
Stored Procedures	A *stored procedure* is a subroutine available to applications that access a relational database management system (RDBMS). Stored procedures are: ▪ Stored in the database data dictionary. ▪ Included in data validation integrated into the database or access control mechanisms. ▪ Consolidated and centralize logic that was originally implemented in applications.
Code Signing	*Code signing* is the process of digitally signing (encrypting) executables and scripts to confirm the software author and guarantee that the code has not been altered or corrupted since it was signed. The process employs the use of a cryptographic hash to validate authenticity and integrity. Code signing: ▪ Provides security when deployed. ▪ Helps prevent namespace conflicts in some programming languages. ▪ Provides a digital signature mechanism to verify the identity of the author or build system. ▪ Provides a checksum to verify that the object has not been modified. ▪ Provides versioning information about an object or to store other metadata about an object.
Obfuscation	*Obfuscation* is the deliberate act of creating source or machine code that is difficult for humans to understand. In other words, the code is camouflaged. Programmers use roundabout expressions to compose statements that deliberately obfuscate code to conceal its purpose or its logic. They use implicit values embedded in it to prevent tampering, deter reverse engineering, or as a puzzle or recreational challenge for someone reading the source code. This is done using an automated tool, but can also be done manually.
Code Reuse	*Code reuse* is the practice of using existing software or software knowledge to build new software utilizing reusability principles.
Dead Code	*Dead code* is a section in the source code of a program that executes, but the result is never used in any computation. The execution of dead code wastes computation time and memory.
Memory Management	*Memory management* is a resource management process applied to computer memory. It allows your computer system to assign portions of memory called blocks to various running programs to optimize overall system performance. Memory management resides in the hardware, the operating system, programs, and applications. In the hardware, memory management involves components that physically store data, such as RAM chips, memory caches, and SSDs. In the OS, memory management involves the allocation of specific memory blocks to individual programs as user demands change. At the application level, memory management ensures the availability of adequate memory for the objects and data structures of each running program at all times. When the program requests a block of memory, the allocator in the memory manager assigns that block to the program. When a program no longer needs the data in the previously allocated memory blocks, those blocks become available for reassignment. This task can be done automatically by the memory manager or manually by the programmer.

Data Exposure	*Data exposure* is the exposure of sensitive information such as passwords, session tokens, credit card data, and private health data.
Fuzz Testing	*Fuzz testing* (also known as *fuzzing*) is a software testing technique that exposes security problems by providing invalid, unexpected, or random data to the inputs of an application. Fuzzing program types are: ▪ *Mutation-based*, which mutate existing data samples to create test data. ▪ *Generation-based*, which define new test data based on models of the input.

DevOps (short for development and operations) is a software development and delivery process that emphasizes communication and collaboration between product management, software development, and operations professionals. It utilizes an automating and monitoring process of software integration, testing, deployment, and infrastructure changes by establishing a culture and environment where software is built, tested, and released rapidly, frequently, and reliably.

DevSecOps (or Secure DevOps) is the inclusion of IT security into the DevOps processes. The following table outlines DevOps software development practices:

DevSecOps Practice	Description
Security Automation	*Security automation* is the automatic handling of security tasks without human intervention. Automation allows you to minimize the risk of human error. Security automation allows you to automate security administration tasks like the following: ▪ Vulnerability scans ▪ User provisioning and deprovisioning ▪ Security policies ▪ How alerts are handled ▪ The investigation, action, and remediation of a cyber threat ▪ An incident response workflow
Continuous Integration	*Continuous integration* is practice of merging all new or changed code into a central repository, after which automated builds and tests are run.
Baselining	*Baselining* is a method for analyzing computer network performance by comparing current performance to a historical metric, or baseline. With a baseline established, you can more easily identify abnormal activity and areas that need improvement.
Immutable Infrastructure	*Immutable infrastructure* (also known as immutable systems) is comprised of immutable components that are replaced in every deployment rather than being updated in-place. Immutable components are executed from a common image built once per deployment and can be tested and validated. The common image can be built through automation, but it doesn't have to be. Immutability is independent of any tool or workflow for building the images, and it is best used in a cloud or virtualized environment.
Infrastructure as Code	*Infrastructure as code* (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files rather than physical hardware configuration or interactive configuration tools. Physical equipment, virtual machines, and associated configuration resources are called infrastructure, even through they have nothing to do with actual infrastructure. Infrastructure as code approaches are used in cloud computing and are sometimes marketed as infrastructure as a service (IaaS). IaC supports IaaS, but should not be confused with it.
Static Analysis	*Static analysis (*also called static code analysis) is a method of computer program debugging. The code is examined without executing the program. The process provides an understanding of the code structure and ensures that the code adheres to industry standards. *Static code analyzers* are tools used to carrying out static analysis.
Stress Testing	*Stress testing* is the process of determining the ability of a computer, network, program, or device to determine its stability. It involves testing beyond normal operational capacity to: ▪ Determine breaking points or safe usage limits. ▪ Confirm that your mathematical model is accurate enough to predict breaking points or safe usage limits. ▪ Confirm that intended specifications are being met. ▪ Determine exactly how a system fails. ▪ Test stable operation of a part or system outside standard usage.
Sandboxing	A *sandbox* is a security mechanism for separating running programs on your network with untested or untrusted programs or code. A sandbox environment allows you to test software from unverified or untrusted third parties, suppliers, users, or websites without risking harm to your production network. Sandboxing is also used to test unverified programs that may contain a virus or other malicious code.
Model Verification	*Verification* of a simulation model is the process of confirming that the model is correctly implemented as the conceptual model intended. During verification, the model is tested to find and fix errors in the implementation of the model. You can verify a

and Validation	model as follows: ▪ Have the model checked by an expert. ▪ Make logic flow diagrams that include each logically possible action. ▪ Examine the model output for reasonableness under a variety of settings of the input parameters. ▪ Use an interactive debugger. *Validation* of a simulation model is a process used to see how accurately the model represents the real system. A model is built for a specific purpose or a set of objectives. Its validity is determined by that purpose. You can validate a model using subjective reviews, objective statistical tests, and a series of tests.
Code Review	A *code review* is a systematic examination of an application's source code. It is intended to find and fix overlooked mistakes, improving the overall quality and security of software. A code review is sometimes called a *peer review*.
Configuration Testing	*Configuration testing* is the process of testing an application under development on systems that have various combinations of hardware and software implemented.