Chapter #14: Architecture and Design

Chapter Objectives

- Examine secure development lifecycle models
- Explore secure coding concepts
- Learn to summarize secure application development and deployment concepts

Development Lifecycle Models

- The production of software is the result of a process
- There are a multitude of tasks, from gathering requirements, planning, design, coding, testing and support
- These tasks are performed by a team of people according to a process model
- There are several different process models

Waterfall

- The waterfall model is based on simple manufacturing design
- This is a linear, sequential process, and the model discourages backing up and repeating earlier stages
- Poorly suited for complex processes and systems
- Difficult to incorporate late in the cycle changes from a customer, making the development process inflexible
**SDLC Phases**

1. **System Need**
   - Define System
   - Design System Components
   - Implement System

2. **Project Plan**
   - Project Plan
   - System Definition

**System Definition**

- **Business Planning Process**
- **System Need**
- **Project Plan**

**Requirements Analysis**

- **System Definition**
- **Project Plan**
- **Requirements Analysis**
- **Approved User Requirements**
- **Component Design**

**Component Design**

- **Requirements Analysis**
- **Approved User Requirements**
- **Component Design**
- **System Design**
- **Implementation**
Implementation

Conversion types
1. Pilot
2. Phased
3. Parallel
4. Plunge

Five Elements

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Software</th>
<th>Data</th>
<th>Procedures</th>
<th>People</th>
</tr>
</thead>
</table>

Agile

- The agile model is a whole group of related methods
- Designed to increase innovation and efficiency of small programming teams
- Methods rely on quick turns involving small increases in functionality
- Small development cycles can enable different developer behaviors, which in turn can result in more efficient development

Scrum

- It is built around a 30-day release cycle
- Highly dependent upon a prioritized list of high-level requirements
- The concept is to keep the software virtually always ready for release
- Program changes are managed on a 24-hour and 30-day basis
- Security – through design phase and developer training
eXtreme Programming

- Extreme Programming is a structured process that is built around user stories
- They are used to architect requirements
- XP is built around the people side of software development
- Many small, incremental changes on a regular schedule
- XP stresses team-level communication, and is highly amenable to the inclusion of security methods

Secure DevOps

- DevOps is a combination of development and operations
- Emphasizes communication and collaboration between product management, software development, and operations professionals
- It facilitates continuous development, continuous integration, continuous delivery, and continuous monitoring processes - security can be added Secure DevOps
- Can be considered the anti-waterfall model

Security Automation

- Security automation can improve efficiency
- Automating routine and extensive processes allows fewer resources to cover more environment in a more effective and efficient manner
- Removes the manual labor so personnel can spend their time doing value-added work such as analysis

Continuous Integration

- Continuous integration is the DevOps manner of continually updating and improving the production code base
- It allows the DevOps team to test and update even very minor changes without a lot of overhead
- DevOps team can run a series of smaller, single-purpose integrations throughout the process and test them
- This can make DevOps more secure by reducing interaction errors and other errors that are difficult to detect
**Baselining**

- Baselining is the process of determining a standard set of functionality and performance
- Metrics-driven process, changes can be compared to the baseline to test impact on performance and other variables
- If a change improves the baseline elements in a positive fashion, a new baseline can be established
- Baselining is important for DevOps and security
- Without a reference point it is hard to show improvements

**Immutable Systems**

- An immutable system is a system that, once deployed, is never modified, patched, or upgraded
- If a patch or update is required, the system is merely replaced with a new system that is patched and updated
- These are systems that are very difficult to lock down and perform authorized system and software updates at the same time

**Infrastructure as Code**

- *Infrastructure as code* is the use of code to build systems, rather than manually configuring them via normal configuration mechanisms
- Using automation to build out systems, reproducible, efficient; key attribute of enabling best practices in DevOps
- Developers become more involved in defining system configuration, and the Ops team gets more involved in the actual development process – version of Infrastructure as a

**Version Control**

- Programs are developed, released, used and then changes are desired, either to change functionality, fix errors, or improve performance
- This leads to multiple versions of programs
- Version control is tracking which version of a program is being worked on, whether in development, testing, or production
- Version control systems use primary numbers to indicate major releases, and numbers after a decimal point to indicate minor changes
**Change Management**

- Change management addresses how an organization manages which versions are currently being used, and how it coordinates changes as they are released by a manufacturer.
- Tracking version numbers, and bug fixes, including what is being fixed, with the why and how behind the changes is important documentation.
- Advanced teams keep track not only of what went wrong, but the root cause analysis.

**Provisioning and Deprovisioning**

- Provisioning is the process of assigning to users permissions or authorities to access objects.
- Users can be provisioned into groups.
- Computer processes or threads can be provisioned to operate at higher levels of authority when executing.
- Best practice includes removing higher levels of permission when not needed.
- Deprovisioning is the removal of permissions or authorities.

**Secure Coding Concepts**

- Application security begins with code that is secure and free of vulnerabilities.
- Development
  - SDLC
  - common errors - [MITRE]
  - web applications - [OWASP]

- Important to ensure that later code changes do not reintroduce old vulnerabilities.
- In traditional software publishing, a new version required a new install and fairly significant testing.
- In DevOps developers and production work together to create a series of micro releases.
- Change management ensures that all changes in production are authorized, properly tested, and, in case of failure, rolled back – documentation!
Proper Error Handling

- Improper exception handling can lead to a wide range of disclosures (SQL statements can disclose data structures and data elements. Remote procedure call (RPC) errors can give up sensitive information such as filenames, paths, and server names. Programmatic errors can give up line numbers that an exception occurred on, the method that was invoked, and information such as stack elements.

Proper Input Validation

- Considering all inputs to be hostile until properly validated can mitigate many attacks.
- Validation efforts need to occur after all parsers have completed manipulating input streams.
- Proper input validation is especially well suited for: buffer overflow, reliance on untrusted inputs in a security decision, cross-site scripting (XSS), cross-site request forgery (XSRF), path traversal, and incorrect calculation of buffer size.

Normalization

- Normalization is an initial step in the input validation process.
- Creating the canonical form, or simplest form, of a string before processing.
- Strings can be encoded using Unicode and other encoding methods.
- Different libraries exist to assist developers in performing this part of input validation.

Stored Procedures

- Stored procedures are precompiled methods implemented within a database engine.
- They offer an isolation of user input from the actual SQL statements being executed.
- Primary defense mechanism against SQL injection attacks.
- Users can define the specificity of search, match, and so forth but users shall not write the actual SQL code – just "fixing" input has never worked.
- All major database engines support stored procedures.
- They are written in SQL, a DB programmer so is needed.
**Code Signing**

- Ensuring that software is genuine and has not been altered. *Signing* involves a digital signature so the end user can verify the code integrity.
- It also provides evidence to the source of the software.
- Code signing rests on public key infrastructure (PKI).
- Developers also publish hash values for downloads.

**Encryption**

- *Encryption* programming - "never roll your own crypto".
- Vetted, proven cryptographic libraries exist for all major languages – these libraries are considered best practice.
- Crypto is almost impossible to invent, and very hard to implement correctly.
- Adopt proven algorithms and utilize proven code bases.

**Obfuscation/Camouflage**

- *Obfuscation or camouflage* is the hiding of obvious meaning from observation.
- Works well for data names and other exposed elements.
- Does not work well in the construction of code.
- Obfuscated code, or code that is hard or even nearly impossible to read, is a ticking time bomb.
- Maintenance.

**Code Reuse**

- Software development includes extensive reuse of components, libraries to reduce development time and costs.
- *Code reuse* can also simplify a system through the reuse of known elements.
- Failure of a widely reused code component has a ripple effect across many applications.
- Development team should make decisions.
- Legacy code can reduce development efforts and risk.
Dead Code

- **Dead code** is code that while it may be executed, the results that it produces are never used elsewhere in the program.
- There are compiler options that can remove dead code, called dead code elimination.
- But you must use these options with care.

Server-Side vs. Client-Side Execution and Validation

- Input validation can be performed either on the server side or on the client side – never trust input without validation.
- Client can be programmed to assure input veracity.
- But client can become corrupted so the veracity cannot be guaranteed.
- Server-side execution can be secured.
- Client side: save the round-trip communication time, check for input errors, being complete and approximately correct.
- All input validation for completeness, correctness, and security checks must be done on the server side, and must be done before the user input is used.

Memory Management

- **Memory management**: control and coordinate computer memory, assigning memory to variables and reclaiming it.
- Errors in memory management can result in memory leak.
- To clean up memory is called garbage collection.
- In the C programming language the programmer must allocate and free memory explicitly.
- Java, C#, Python, and Ruby provide automatic memory management with garbage collection.

Use of Third-Party Libraries and SDKs

- Programming today is an exercise in using third-party libraries and software development kits (SDKs).
- Once code has been debugged and proven to work, rewriting it is generally not a valuable use of time.
- Proven library sets remove a lot of risk from programming.
- They can reduce errors and vulnerabilities in code.
- You don’t have all the dependency details – but if they are managed correctly, the benefits greatly outweigh the risks.
**Data Exposure**

- *Data exposure* is the loss of control over data
- Data must be protected during storage (data at rest), during communication (data in transit), and during use
- Exposed data can be lost to unauthorized parties (a failure of confidentiality) or, can be changed by an unauthorized party (a failure of integrity)
- Protection of the data will typically be done using various forms of cryptography

**Code Quality and Testing**

- Code quality does not end with development – it needs to be delivered and installed both intact and correctly
- Code analysis encompasses the processes used to inspect code for weaknesses and vulnerabilities
  - Static analysis: examination of the code without execution
  - Dynamic analysis: execution of the code as part of the testing
- They are typically performed with tools
- Code testing is the verification that the code meets to functional requirements

**Static Code Analyzers**

- Can be performed on both source code and object code
- By humans or tools, or automated tools - *static code analyzers*
  - “Binary scanners” or “byte code scanners,”
- Search for weaknesses and vulnerabilities
- Tools can check syntax, approved function/library calls, and examine rules and semantics associated with logic and calls
- They can catch elements a human could overlook

**Dynamic Analysis**

- *Dynamic analysis*: software is executed
- The system is fed specific test inputs to produce specific behaviors
- Dynamic analysis requires specialized automation to perform specific testing
- Programs designed to detect errors, race conditions and memory addressing errors
**Fuzzing**

- Fuzzing (or fuzz testing) is a brute force method of addressing input validation issues and vulnerabilities.
- Large numbers of inputs are used to determine which ones cause faults and which ones might be vulnerable to exploitation.
- Network protocols can be fuzzed, file protocols can be fuzzed, and web protocols can be fuzzed.
- The vast majority of browser errors are found via fuzzing.
- It works well in white, black, or gray box testing.
- Malformed inputs can be used to vary parser operations, to check for memory leaks, buffer overflows, and input validation issues.

**Stress Testing**

- Typical objective is to determine bottlenecks and performance factors.
- They are also referred to as load testing and stress testing.
- Load testing involves running the system under a controlled speed environment.
- Stress testing takes the system past this operating point to see how it responds to overload conditions.
- Determine the service levels that can be expected from the software in a production environment.

**Sandboxing**

- Sandbox refers to the execution of computer code in an isolated environment.
- They are used to execute untrusted code, code from guests, and unverified programs.
- They work like a virtual machine (VM) and can mediate a wide range of system interactions, from memory access to network access, and access to other programs, the file system, and devices.
- The level of protection offered by a sandbox depends upon the level of isolation and mediation offered.

**Model Verification**

- Validation is the process of checking whether the program specification captures the requirements from the customer.
- Verification is the process of checking that the software developed meets the model specification.
- Performing model verification testing is important, as this is the assurance that the code as developed meets the design requirements.
Compiled vs. Runtime Code

- Compiled code: source code is transformed into executable code that can be run on a system
- Compilers can optimize code and create smaller, faster-running programs on the actual hardware
- Can not be changed at runtime
- Interpreters create runtime code that can be executed via an interpreter engine
- Slower than compilers in execution – they manage the conversion on the fly

Stay Alert! There is no 100 percent secure system, and there is nothing that is foolproof!