INTERMEDIATE LEVEL

Advanced Integrity Vulnerability Concepts

1. Supply Chain Attacks

- **Definition: Compromising software by targeting its development or distribution chain**
- **Attack Vectors:**
  - **Compromised build systems**
  - **Malicious dependencies**
  - **Corrupted update servers**
  - **Modified source code repositories**
- **Real-World Impact: SolarWinds attack affected thousands of organizations**

2. Advanced Deserialization Vulnerabilities

- **Polymorphic Deserialization: Type confusion during object reconstruction**
- **Gadget Chains: Using legitimate classes in unintended ways during deserialization**
- **Memory Corruption: Buffer overflows during complex deserialization**
- **Format-Specific Attacks: XML, JSON, YAML-specific deserialization flaws**

3. Runtime Integrity Issues

- **Dynamic Code Loading: Security risks when loading code at runtime**
- **In-Memory Tampering: Modifying running code in memory**
- **DLL/Library Hijacking: Replacing legitimate libraries with malicious ones**
- **JIT Spraying: Manipulating just-in-time compilation for code execution**

4. Data Pipeline Integrity Failures

- **ETL Process Vulnerabilities: Data corruption during extraction, transformation, loading**
- **Data Transit Tampering: Modifying data between systems**
- **Schema Poisoning: Altering data models to affect interpretation**
- **Cache Poisoning: Corrupting cached data for later use**

Complex Real-World Examples

1. Dependency Confusion Attacks

- **Technique: Publishing malicious packages with names matching private dependencies**
- **Mechanism: Package managers preferring public repositories over private ones**
- **Impact: Automatic inclusion of malicious code in build processes**
- **Affected Organizations: Microsoft, Apple, PayPal, and many others**

2. Plugin/Extension System Abuse

- **Technique: Compromising plugin ecosystems for popular software**
- **Examples: VS Code extensions, WordPress plugins, browser extensions**
- **Attack Surface: Large install base, excessive permissions**
- **Impact: Data exfiltration, credential theft, environment compromise**

3. CI/CD Pipeline Compromises

- **Technique: Inserting malicious code during automated build processes**
- **Attack Points: Build scripts, action files, environment variables**
- **Example: GitHub Actions workflow compromise**
- **Impact: Backdoored builds affecting all downstream users**

Intermediate Detection Methods

1. Automated Integrity Verification

- **Software Composition Analysis (SCA): Identifying and verifying components**
- **Subresource Integrity (SRI): Verifying integrity of loaded web resources**
- **Binary Analysis: Examining compiled code for unauthorized modifications**
- **Tamper-Evident Logging: Detecting and alerting on unexpected changes**

2. Advanced Testing Techniques

- **Fuzzing Deserialization Functions: Finding parsing vulnerabilities**
- **Supply Chain Testing: Verifying integrity of dependencies**
- **Penetration Testing for Integrity: Targeted attacks against integrity controls**
- **Runtime Application Self-Protection (RASP): Detecting integrity violations during execution**

3. Monitoring and Alerting

- **File Integrity Monitoring (FIM): Detecting unauthorized file changes**
- **Behavioral Analysis: Identifying unusual code execution patterns**
- **Change Monitoring: Alerting on unexpected system modifications**
- **Update Verification: Validating software updates before deployment**

Intermediate Prevention Strategies

1. Secure Development Practices

- **Secure Coding for Integrity: Defensive programming techniques**
- **Safe Deserialization Patterns: Type constraints, allowlists, validation**
- **Integrity by Design: Building integrity checks into architecture**

- **Dependency Management: Vetting and pinning trusted dependencies**

2. Implementation of Technical Controls

- **Code Signing Infrastructure: Managing signing certificates and processes**
- **Reproducible Builds: Ensuring build output matches source code**
- **Software Bill of Materials (SBOM): Cataloging software components**
- **Integrity Verification Systems: Automated checking of code and components**

3. Data Integrity Frameworks

- **Data Validation Pipelines: Consistent validation across systems**
- **Digital Signatures for Data: Cryptographically signing important data**
- **Blockchain for Data Integrity: Immutable ledgers for sensitive records**
- **Atomic Transactions: Ensuring data operations complete fully or not at all**