Contents

[Part I Secure Software Concepts 3](#_Toc21633934)

[Ch1. General Security Concepts 3](#_Toc21633935)

[1.1 General Security Concepts 3](#_Toc21633936)

[1.1.1 Security Basics 3](#_Toc21633937)

[1.1.2 System Tenets 3](#_Toc21633938)

[1.1.3 Secure Design Tenets 3](#_Toc21633939)

[Layered Security 3](#_Toc21633940)

[1.2 Security Models 3](#_Toc21633941)

[1.2.1 Access Control Models 3](#_Toc21633942)

[FIGURE 1-2 Using roles to mediate permission assignments 3](#_Toc21633943)

[1.2.2 Multilevel Security Model 3](#_Toc21633944)

[1.2.3 Integrity Models 3](#_Toc21633945)

[1.2.4 Information Flow Models 3](#_Toc21633946)

[FIGURE 1-3 Operational model of security 3](#_Toc21633947)

[1.3 Adversaries 3](#_Toc21633948)

[1.3.1 Adversary Type 3](#_Toc21633949)

[1.3.2 Adversary Groups 3](#_Toc21633950)

[1.3.3 Threat Landscape Shift 3](#_Toc21633951)

[Ch2 Risk Management 3](#_Toc21633952)

[2.1 Definitions and Terminology 3](#_Toc21633953)

[2.1.1 General Terms 3](#_Toc21633954)

[2.1.2 Quantitative Terms 3](#_Toc21633955)

[2.1.3 Risk Management Statements 4](#_Toc21633956)

[FIGURE 2-1 Well-formed risk statement 4](#_Toc21633957)

[2.2 Types of Risk 4](#_Toc21633958)

[2.2.1 Business Risk 4](#_Toc21633959)

[2.2.2 Technology Risk 4](#_Toc21633960)

[2.2.3 Risk Controls 4](#_Toc21633961)

[FIGURE 2-2 Controls framework 4](#_Toc21633962)

[2.2.4 Qualitative Risk Management 4](#_Toc21633963)

[2.2.5 Qualitative Matrix 4](#_Toc21633964)

[Table 2-1 Sample Qualitative Matrix 4](#_Toc21633965)

[2.2.6 Quantitative Risk Management 4](#_Toc21633966)

[FIGURE 2-3 Sample residual risk calculation 4](#_Toc21633967)

[2.2.7 Comparison of Qualitative & Quantitative Methods 4](#_Toc21633968)

[2.3 Governance, Risk, and Compliance 4](#_Toc21633969)

[2.3.1 Regulations and Compliance 4](#_Toc21633970)

[2.3.2 Legal 4](#_Toc21633971)

[2.3.3 Standards 4](#_Toc21633972)

[2.4 Risk Management Models 4](#_Toc21633973)

[2.4.1 General Risk Management Model 4](#_Toc21633974)

[2.4.2 Software Engineering Institute Model 5](#_Toc21633975)

[2.4.3 Model Application 5](#_Toc21633976)

[FIGURE 2-4 Risk complexity versus project size 5](#_Toc21633977)

[2.5 Risk Options 5](#_Toc21633978)

[Ch3 Security Policies & Regulations 5](#_Toc21633979)

[3.1 Regulations and Compliance 5](#_Toc21633980)

[3.1.1 FISMA 5](#_Toc21633981)

[3.1.2 Sarbanes-Oxley 5](#_Toc21633982)

[3.1.3 Gramm-Leach-Bliley 5](#_Toc21633983)

[3.1.4 HIPAA and HITECH 5](#_Toc21633984)

[3.1.5 Payment Card Industry Data Security Standard (PCI DSS) 5](#_Toc21633985)

[3.1.6 Other Regulations 5](#_Toc21633986)

[3.2 Legal Issues 5](#_Toc21633987)

[3.2.1 Intellectual Property 5](#_Toc21633988)

[3.3 Privacy 5](#_Toc21633989)

[3.3.1 Privacy Policy 5](#_Toc21633990)

[3.3.2 Personally Identifiable Information (PII) 5](#_Toc21633991)

[3.3.3 Personal Health Information (PHI) 5](#_Toc21633992)

[3.3.4 Breach Notifications 5](#_Toc21633993)

[3.3.5 Data Protection Principles 5](#_Toc21633994)

[3.3.6 California Consumer Privacy Act 2018 (AB 375) 5](#_Toc21633995)

[3.4 Security Standards 5](#_Toc21633996)

[3.5 Secure Software Architecture 5](#_Toc21633997)

[3.6 Trusted Computing 5](#_Toc21633998)

[3.7 Acquisition 5](#_Toc21633999)

[Ch4 Software Development Methodologies 5](#_Toc21634000)

[4.1 Secure Development Lifecycle 5](#_Toc21634001)

[4.2 Security Features != Secure Software 5](#_Toc21634002)

[4.3 Software Development Models 5](#_Toc21634003)

[4.4 Microsoft Security Development Lifecycle 6](#_Toc21634004)

[Part II Secure Software Requirements 6](#_Toc21634005)

[Ch5 Policy Decomposition 6](#_Toc21634006)

[5.1 Confidentiality, Integrity, Availability Requirements 6](#_Toc21634007)

[5.2 Authentication, Authorization, Auditing Requirements 6](#_Toc21634008)

[5.3 Internal & External Requirements 6](#_Toc21634009)

[Ch6 Data Classification and Categorization 6](#_Toc21634010)

[6.1 Data Classification 6](#_Toc21634011)

[6.2 Data Ownership 6](#_Toc21634012)

[6.3 Labeling 6](#_Toc21634013)

[6.4 Types of Data 6](#_Toc21634014)

[6.5 Data Lifecycle 6](#_Toc21634015)

[Ch7 Requirements 6](#_Toc21634016)

[7.1 Functional Requirements 6](#_Toc21634017)

[7.2 Operational Requirements 6](#_Toc21634018)

[7.3 Requirements Traceability Matrix 6](#_Toc21634019)

[7.4 Connecting the Dots 6](#_Toc21634020)

[Part III Secure Software Design 6](#_Toc21634021)

[Ch8 Design Processes 6](#_Toc21634022)

[8.1 Attack Surface Evaluation 6](#_Toc21634023)

[8.1.1 Attack Surface Measurement 6](#_Toc21634024)

[MS WINDOWS attack surface elements (17) 6](#_Toc21634025)

[8.1.2 Attack Surface Minimization 6](#_Toc21634026)

[8.2 Threat Modeling 6](#_Toc21634027)

[8.2.1 Threat Model Development 6](#_Toc21634028)

[8.2.1.1 Identify Security Objectives 6](#_Toc21634029)

[8.2.1.2 System Decomposition 6](#_Toc21634030)

[8.2.1.3 Threat Identification 7](#_Toc21634031)

[STRIDE Method 7](#_Toc21634032)

[8.2.1.4 Mitigation Analysis 7](#_Toc21634033)

[FIGURE 8-1 Attack tree 7](#_Toc21634034)

[8.2.1.5 Threat Model Validation 7](#_Toc21634035)

[8.3 Control Identification & Prioritization 7](#_Toc21634036)

[8.4 Risk Assessment for Code Reuse 7](#_Toc21634037)

[8.5 Documentation 7](#_Toc21634038)

[8.6 Design & Architecture Technical Review 7](#_Toc21634039)

[Ch9 Design Considerations 7](#_Toc21634040)

[9.1 Application of Methods to Address Core Security Concepts 7](#_Toc21634041)

[9.1.1 Confidentiality, Integrity, Availability 7](#_Toc21634042)

[9.1.2 Authentication, Authorization, Auditing 7](#_Toc21634043)

[9.1.3 Secure Design Principles 7](#_Toc21634044)

[9.1.3.1 Good Enough Security 7](#_Toc21634045)

[9.1.3.2 Least Privilege 7](#_Toc21634046)

[9.1.3.3 Separation of Duties 7](#_Toc21634047)

[9.1.3.4 Defense in Depth 7](#_Toc21634048)

[9.1.3.5 Fail Safe 7](#_Toc21634049)

[9.1.3.6 Economy of Mechanism 7](#_Toc21634050)

[9.1.3.7 Complete Mediation 7](#_Toc21634051)

[9.1.3.8 Open Design 7](#_Toc21634052)

[9.1.3.9 Least Common Mechanism 7](#_Toc21634053)

[9.1.3.10 Psychological Acceptability 7](#_Toc21634054)

[9.1.3.11 Weakest Link 7](#_Toc21634055)

[9.1.3.12 Leverage Existing Components 7](#_Toc21634056)

[9.1.3.13 Single Point of Failure 7](#_Toc21634057)

[9.1.4 Interconnectivity 7](#_Toc21634058)

[9.1.4.1 Session Management 7](#_Toc21634059)

[9.1.4.2 Exception Management 7](#_Toc21634060)

[9.1.4.3 Configuration Management 7](#_Toc21634061)

[9.2 Interfaces 7](#_Toc21634062)

[Ch10 Securing Commonly Used Architecture 7](#_Toc21634063)

[10.1 Distributed Computing 7](#_Toc21634064)

[10.1.1 Client Server 7](#_Toc21634065)

[FIGURE 10-1 N-tier architecture 7](#_Toc21634066)

[10.1.2 Peer-to-Peer 7](#_Toc21634067)

[10.1.3 Message Queuing 7](#_Toc21634068)

[10.2 Service-Oriented Architecture 7](#_Toc21634069)

[10.2.1 Enterprise Service Bus (ESB) 7](#_Toc21634070)

[FIGURE 10-2 Enterprise service bus 8](#_Toc21634071)

[10.2.2 Web Services 8](#_Toc21634072)

[10.2.2.1 REST = Representative State Transfer (REST) 8](#_Toc21634073)

[10.2.2.2 JSON = JavaScript Object Notation 8](#_Toc21634074)

[10.2.2.3 UDDI = Universal Description, Discovery, and Interface 8](#_Toc21634075)

[10.3 Rich Internet Applications 8](#_Toc21634076)

[10.3.1 Client-Side Exploits or Threats 8](#_Toc21634077)

[10.3.2 Remote Code Execution 8](#_Toc21634078)

[10.4 Pervasive/Ubiquitous Computing 8](#_Toc21634079)

[10.4.1 Wireless 8](#_Toc21634080)

[10.4.2 Location-Based 8](#_Toc21634081)

[10.4.3 Constant Connectivity 8](#_Toc21634082)

[10.4.4 Radio Frequency Identification (RFID) 8](#_Toc21634083)

[10.4.5 Near-Field Communication (NFC) 8](#_Toc21634084)

[10.4.6 Sensor Networks 8](#_Toc21634085)

[10.5 Mobile Applications 8](#_Toc21634086)

[10.6 Integration with Existing Architectures 8](#_Toc21634087)

[10.7 Cloud Architectures 8](#_Toc21634088)

[10.7.1 Software as a Service 8](#_Toc21634089)

[10.7.2 Platform as a Service 8](#_Toc21634090)

[10.7.3 Infrastructure as a Service 8](#_Toc21634091)

[Ch11 Technologies 8](#_Toc21634092)

[11.1 Authentication & Identity Management 8](#_Toc21634093)

[11.1.1 Identity Management 8](#_Toc21634094)

[11.1.2 Authentication 8](#_Toc21634095)

[FIGURE 11-1 RP and IdP relationships 8](#_Toc21634096)

[FIGURE 11-2 OpenID vs. Oauth 8](#_Toc21634097)

[11.2 Credential Management 8](#_Toc21634098)

[11.2.1 X.509 Credentials 8](#_Toc21634099)

[11.2.2 Single Sign-On 9](#_Toc21634100)

[11.3 Flow Control (Proxies, Firewalls, Middleware) 9](#_Toc21634101)

[11.3.1 Firewalls 9](#_Toc21634102)

[11.3.2 Proxies 9](#_Toc21634103)

[11.3.3 Application Firewalls 9](#_Toc21634104)

[11.3.4 Queuing Technology 9](#_Toc21634105)

[11.4 Logging 9](#_Toc21634106)

[11.4.1 Syslog 9](#_Toc21634107)

[11.5 Data Loss Prevention 9](#_Toc21634108)

[11.6 Virtualization 9](#_Toc21634109)

[11.7 Digital Rights Management 9](#_Toc21634110)

[11.8 Trusted Computing 9](#_Toc21634111)

[11.8.1 TCB 9](#_Toc21634112)

[11.8.2 TPM 9](#_Toc21634113)

[FIGURE 11-4 TPM hardware functions 9](#_Toc21634114)

[11.8.3 Malware 9](#_Toc21634115)

[11.8.4 Code Signing 9](#_Toc21634116)

[11.9 Database Security 9](#_Toc21634117)

[11.9.1 Encryption 9](#_Toc21634118)

[11.9.2 Triggers 9](#_Toc21634119)

[11.9.3 Views 9](#_Toc21634120)

[11.9.4 Privilege Management 9](#_Toc21634121)

[11.10 Programming Language Environment 10](#_Toc21634122)

[11.10.1 CLR 10](#_Toc21634123)

[11.10.2 JVM 10](#_Toc21634124)

[11.10.3 Compiler Switches 10](#_Toc21634125)

[11.10.4 Sandboxing 10](#_Toc21634126)

[11.10.5 Managed vs. Unmanaged Code 10](#_Toc21634127)

[11.11 Operating Systems 10](#_Toc21634128)

[11.12 Embedded Systems 10](#_Toc21634129)

[11.12.1 Control Systems 10](#_Toc21634130)

[11.12.2 Firmware 10](#_Toc21634131)

[Part IV Secure Software Implementation/ Programming 10](#_Toc21634132)

[Ch12 Common Software Vulnerabilities & Countermeasures 10](#_Toc21634133)

[12.1 CWE/SANS Top 25 Vulnerability Categories 10](#_Toc21634134)

[Insecure Interaction Between Components (6) 10](#_Toc21634135)

[Risky Resource Management (8) 10](#_Toc21634136)

[Porous Defenses (11) 10](#_Toc21634137)

[12.2 OWASP Vulnerability Categories 10](#_Toc21634138)

[Open Web Application Security Project (OWASP) Top 10 10](#_Toc21634139)

[12.3 Common Vulnerabilities & Countermeasures 10](#_Toc21634140)

[12.3.1 Injection Attacks 10](#_Toc21634141)

[12.3.2 Cryptographic Failures 10](#_Toc21634142)

[12.4 Input Validation Failures 10](#_Toc21634143)

[12.4.1 Buffer Overflow 10](#_Toc21634144)

[12.4.2 Canonical Form 10](#_Toc21634145)

[12.4.3 Missing Defense Functions 10](#_Toc21634146)

[12.4.4 General Programming Failures 10](#_Toc21634147)

[12.5 Common Enumerations 10](#_Toc21634148)

[12.5.1 Common Weakness Enumerations (CWE) 10](#_Toc21634149)

[12.5.2 Common Vulnerabilities and Exposures (CVE) 10](#_Toc21634150)

[12.6 Virtualization 11](#_Toc21634151)

[12.7 Embedded Systems 11](#_Toc21634152)

[12.8 Side Channel 11](#_Toc21634153)

[12.9 Social Engineering Attacks 11](#_Toc21634154)

[12.9.1 Phishing 11](#_Toc21634155)

[Ch13 Defensive Coding Practices 11](#_Toc21634156)

[13.1 Declarative vs. Programmatic Security 11](#_Toc21634157)

[13.1.1 Bootstrapping 11](#_Toc21634158)

[13.1.2 Cryptographic Agility 11](#_Toc21634159)

[13.1.3 Handling Configuration Parameters 11](#_Toc21634160)

[13.2 Memory Management 11](#_Toc21634161)

[13.2.1 Type-Safe Practice 11](#_Toc21634162)

[13.2.2 Locality 11](#_Toc21634163)

[13.3 Error Handling 11](#_Toc21634164)

[13.3.1 Exception Management 11](#_Toc21634165)

[13.4 Interface Coding 11](#_Toc21634166)

[13.5 Primary Mitigations 11](#_Toc21634167)

[13.6 Learning from Past Mistakes 11](#_Toc21634168)

[Ch14 Secure Software Coding Operations 11](#_Toc21634169)

[14.1 Code Analysis (Static & Dynamic) 11](#_Toc21634170)

[14.1.1 Static 11](#_Toc21634171)

[14.1.2 Dynamic 12](#_Toc21634172)

[14.2 Code/Peer Review 12](#_Toc21634173)

[14.3 Build Environment 12](#_Toc21634174)

[14.3.1 Integrated Development Environment (IDE) 12](#_Toc21634175)

[14.4 Antitampering Techniques 12](#_Toc21634176)

[14.5 Configuration Management: Source Code & Versioning 12](#_Toc21634177)

[Part V Secure Software Testing 12](#_Toc21634178)

[Ch15 Security Quality Assurance Testing 12](#_Toc21634179)

[15.1 Standards for Software Quality Assurance 12](#_Toc21634180)

[15.2 Testing Methodology 12](#_Toc21634181)

[15.3 Functional Testing 12](#_Toc21634182)

[15.4 Security Testing 12](#_Toc21634183)

[15.5 Environment 12](#_Toc21634184)

[15.6 Attack Surface Validation 12](#_Toc21634185)

[15.7 Testing Artifacts 12](#_Toc21634186)

[15.8 Test Data Lifecycle Management 12](#_Toc21634187)

[Ch16 Security Testing 12](#_Toc21634188)

[16.1 Scanning 12](#_Toc21634189)

[16.2 Penetration Testing 13](#_Toc21634190)

[16.3 Fuzzing 13](#_Toc21634191)

[16.4 Simulation Testing 13](#_Toc21634192)

[16.5 Testing for Failure 13](#_Toc21634193)

[16.6 Cryptographic Validation 13](#_Toc21634194)

[16.7 Regression Testing 13](#_Toc21634195)

[16.8 Impact Assessment and Corrective Action 13](#_Toc21634196)

[Part VI Secure Lifecycle Management 13](#_Toc21634197)

[Ch17 Secure Lifecycle Management 13](#_Toc21634198)

[17.1 Introduction to Acceptance 13](#_Toc21634199)

[17.2 Pre-release Activities 13](#_Toc21634200)

[17.3 Post-release Activities 13](#_Toc21634201)

[Part VII Software Deployment, Operations, Maintenance 13](#_Toc21634202)

[Ch18 Secure S/W Installation & Deployment 13](#_Toc21634203)

[18.1 Secure Software Installation & Deployment 13](#_Toc21634204)

[18.2 Configuration Management 13](#_Toc21634205)

[Ch19 Secure S/W Operations & Maintenance 13](#_Toc21634206)

[19.1 Secure Software Operations 13](#_Toc21634207)

[19.2 The Software Maintenance Process 13](#_Toc21634208)

[19.3 Secure DevOps 13](#_Toc21634209)

[19.4 Secure Software Disposal 13](#_Toc21634210)

[Part VIII Supply Chain & Software Acquisition 13](#_Toc21634211)

[Ch20 Supply Chain & Software Acquisition 13](#_Toc21634212)

[20.1 Supplier Risk Assessment 13](#_Toc21634213)

[20.2 Supplier Sourcing 13](#_Toc21634214)

[20.3 Software Development & Testing 13](#_Toc21634215)

[20.4 Software Delivery, Operations, Maintenance 13](#_Toc21634216)

[20.5 Supplier Transitioning 14](#_Toc21634217)

# Part I Secure Software Concepts

# Ch1. General Security Concepts

## 1.1 General Security Concepts

### 1.1.1 Security Basics

Attributes (CIA): confidentiality, integrity, availability

Action-Oriented: authentication, authorization, auditing (accounting)

Non-repudiation

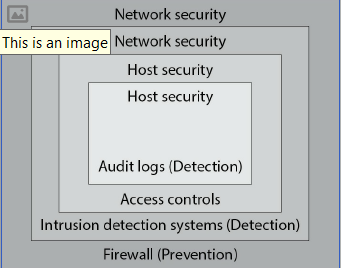
### 1.1.2 System Tenets

Communication between components requires management of a communication session, aka **session management**. Securely managing error conditions is referred to as **exception management**. Software systems require configuration in production: **configuration management** is key in the creation of secure systems.

### 1.1.3 Secure Design Tenets

* Good Enough Security
* Least Privilege
* Separation of Duties
* Defense in Depth: layered security + diversity defense
* Fail-safe
* Economy of Mechanism
* Complete Mediation
* Open Design
* Least Common Mechanism
* Psychological Acceptability
* Weakest Link
* Leverage Existing Components
* Single Point of Failure

#### Layered Security

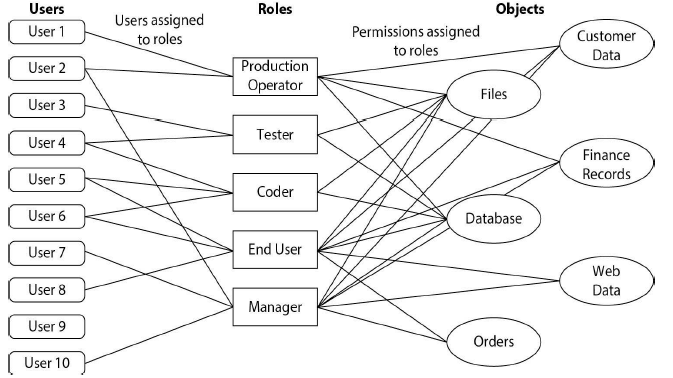


## 1.2 Security Models

### 1.2.1 Access Control Models

* Access Control List (ACL)
* Discretionary Access Control (DAC)
* Mandatory Access Control (MAC)
* Role-based Access Control (RBAC)

#### FIGURE 1-2 Using roles to mediate permission assignments



* Rule-based Access Control (RBAC)
* Attribute-Based Access Control (ABAC)
* Bell-LaPadula Confidentiality Model
* Take-Grant Model

### 1.2.2 Multilevel Security Model

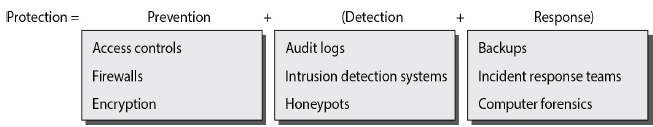
### 1.2.3 Integrity Models

* Biba Integrity Model
* Clark-Wilson Model

### 1.2.4 Information Flow Models

* Brewer-Nash Model (Chinese Wall)
* Data Flow Diagrams
* Use Case Models
* Assurance Models
* NIST CSF Model
* Operational Model of Security

#### FIGURE 1-3 Operational model of security



## 1.3 Adversaries

### 1.3.1 Adversary Type

* Script Kiddie
* Hacker
* Elite

### 1.3.2 Adversary Groups

* Unstructured Threat
* Structured Threat
* Highly Structured Threat
* Nation-state Threat
* Insider vs. Outsider Threat

### 1.3.3 Threat Landscape Shift

# Ch2 Risk Management

## 2.1 Definitions and Terminology

### 2.1.1 General Terms

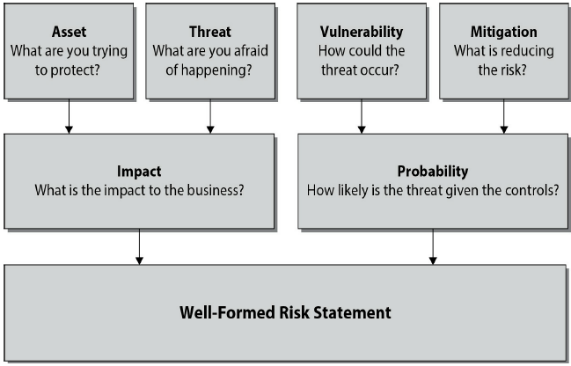
**Risk** = possibility of suffering harm or loss **Inherent risk** = product of likelihood and impact, before any controls are introduced to reduce the risk. **Residual risk** = remains after a control is utilized & reduces the specific risk associated with a vulnerability. Level of risk that must be borne by the entity. **Total risk** = sum of all risks associated with an asset, a process, or business. **Risk management** = overall decision-making process of identifying threats & vulnerabilities and potential impacts, determining the costs to mitigate such events, and deciding what actions are cost effective for controlling these risks **Risk assessment** = process of analyzing an environment to identify the risks (threats & vulnerabilities) and mitigating actions to determine (quantitatively/ qualitatively) the impact of an event that would affect a project, program, or business. It is also sometimes referred to as risk analysis **Asset** = resource or information an organization needs to conduct business **Vulnerability** = characteristic of an asset that can be exploited by a threat to cause harm (failed to instal patches to fix a cross-site scripting (XSS) error) **Attack** = instance of attempting to perform undesired or unauthorized activities via a vulnerability **Impact** = loss resulting when a threat exploits a vulnerability. A malicious hacker (the threat) uses an XSS tool to hack your unpatched website (the vulnerability), stealing credit card information that is used fraudulently. The credit card company pursues legal recourse against your company to recover the losses from the credit card fraud (the impact) **Threat** = any circumstance/ event with the potential to cause harm to asset **Mitigate** = action to reduce likelihood of threat occurring **Control** = measure to detect, prevent, mitigate risk associated with threat. The term control is also called countermeasure or safeguard **Qualitative risk assessment** = process of subjectively determining the impact of an event that affects a project, program, or business. Completing the qualitative risk assessment usually involves the use of expert judgment, experience, or group consensus to complete the assessment.

### 2.1.2 Quantitative Terms

**Quantitative risk assessment** = process of objectively determining impact of an event that affects a project, program, or business. Use of metrics & models **Single loss expectancy (SLE)** = monetary loss/impact of occurrence of a threat **Exposure factor** = measure of magnitude of loss of an asset. Used in calculation of single loss expectancy **Annualized rate of occurrence (ARO)** = frequency with which an event is expected to occur on an annualized basis **Annualized loss expectancy (ALE)** = how much event is expected to cost per year

### 2.1.3 Risk Management Statements

#### FIGURE 2-1 Well-formed risk statement



## 2.2 Types of Risk

### 2.2.1 Business Risk

**Treasury management** = Businesses operate as financial enterprises. The

management of company holdings in bonds, futures, currencies, and other financial instruments is a source of financial risk to the business. **Revenue management** = actions associated with customer behavior and the generation of revenue. As revenue is the lifeblood of business, revenue management is an important area where business risks can affect the enterprise. **Contract management** = refers to managing contracts with customers, vendors, and partners. Contract management can affect both costs & revenues, and is an important aspect of the financial operation of a business. **Fraud** = deliberate deception made for personal gain to obtain property or services, and is a form of business risk. **Regulatory** Security, privacy, other operation regulations can be sources of business risk. When the regulation effect is related to the technology being employed, it can also be seen as a technological risk. **Business continuity Management** of risks associated with recovering & restoring business functions after a disaster or major disruption. Software enterprises tend to be highly dependent upon personnel, so issues that impact personnel involved in software development can be viewed as a business continuity risk. **Technology** creates opportunities for risk, and the employment of technology can be a business risk.

### 2.2.2 Technology Risk

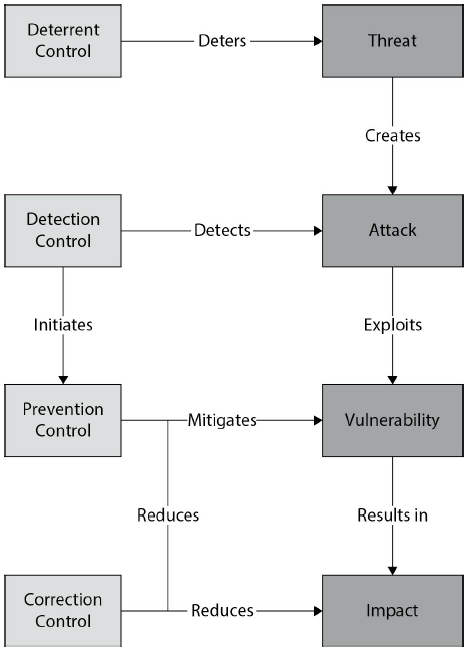
**Security - Privacy** - **Project risk management - Change management**

### 2.2.3 Risk Controls

**Classes of controls**: •Administrative •Technical •Physical

**Types of controls**: •Preventative (deterrent) •Detective •Corrective (recovery) •Compensating

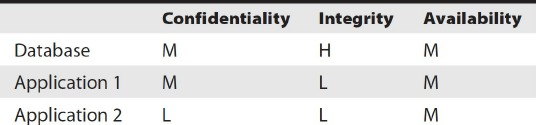
#### FIGURE 2-2 Controls framework



### 2.2.4 Qualitative Risk Management

### 2.2.5 Qualitative Matrix

#### Table 2-1 Sample Qualitative Matrix



* Failure mode effects analysis (FMEA)

### 2.2.6 Quantitative Risk Management

* Annualized Loss Expectancy Model

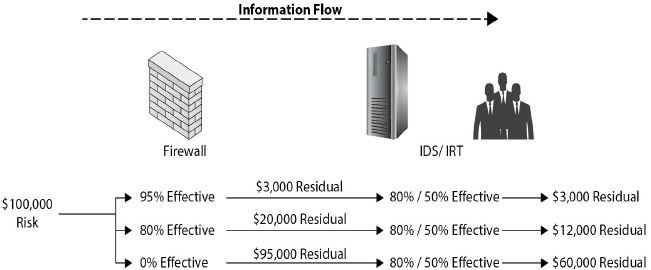
Single Loss Expectancy SLE = asset value \* exposure factor

Annualized Rate of occurrence (ARO) = number of events / number of years

ALE = SLE \* ARO

* Residual Risk Model

#### FIGURE 2-3 Sample residual risk calculation



* Return on Investment (ROI)

ROI (%) = (Avoided Loss – Control Cost) / (Control Cost) \* 100

ROI (Time) = (Avoided Annual Loss) / (Annual Control Cost)

**Example:**

A company owns five web servers, each of which is valued at $100,000 and contributes equally to the company’s capacity. The web servers are geographically spaced at the different regional offices. Each web server provides internal web services to the regional office. The daily value of the content server is calculated at $10,000 to support workers in the office.

Mountain West

SLE = loss \* duration = $10,000 \* 2 = $20,000

ARO = 1/5 = 0.2

ALE = SLE \* ARO = $20,000 \* 0.2 = $4,000

Southeast

SLE = loss \* duration = $10,000 \* 2 = $20,000

ARO = 1/5 = 0.2

ALE = SLE \* ARO = $20,000 \* 0.2 = $4,000

A backup generator costs $40,000, annual maintenance costs $2,000

🡺 Annual cost = $8,000 ($40,000 / 5) + $2,000 annual maintenance cost

= $10,000 total annual cost

For the Mountain West office, ROI has no meaningful value, as it costs more

for the control than the loss it would prevent. For the Southeast Region office,

ROI % = 100% with a payback period of six months.

### 2.2.7 Comparison of Qualitative & Quantitative Methods

## 2.3 Governance, Risk, and Compliance

### 2.3.1 Regulations and Compliance

### 2.3.2 Legal

### 2.3.3 Standards

## 2.4 Risk Management Models

### 2.4.1 General Risk Management Model

Step 1: Asset Identification

Step 2: Threat Assessment: common weaknesses (CWE from mitre.org), SANS Top 25 list, OWASP Top 10 list

Step 3: Impact Determination & Quantification

Step 4: Control Design and Evaluation

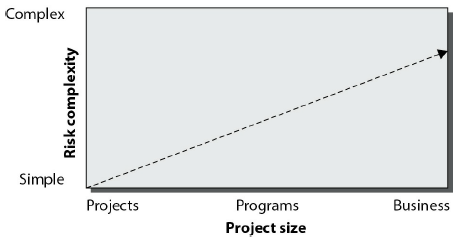
Step 5: Residual Risk Management

### 2.4.2 Software Engineering Institute Model

**1. Identify**: Examine system, enumerating potential risks. **2. Analyze**: Convert risk data into information to make decisions. Evaluate impact, probability, time frame of risks. Classify and prioritize each of the risks. **3. Plan**: Review & evaluate the risks and decide what actions to mitigate. Implement the plan. **4. Track**: Monitor risks & mitigation plans. Trends provide info to activate plans & contingencies. Review periodically to measure progress & identify new risks. **5. Control**: Make corrections for deviations from risk mitigation plans. Correct products & processes as required. Changes in business procedures may require adjustments in plans or actions, as do faulty plans and risks.

### 2.4.3 Model Application

#### FIGURE 2-4 Risk complexity versus project size



## 2.5 Risk Options

# Ch3 Security Policies & Regulations

## 3.1 Regulations and Compliance

### 3.1.1 FISMA

Federal Information Security Management Act of 2002 (FISMA)

Guidelined developed by National Institute of Standards and Technology (NIST): risk management framework (RMF), Information Security Automation Program and the Security Content Automation Protocol (SCAP)

### 3.1.2 Sarbanes-Oxley

Section 404 mandates a specific level of internal control measures

### 3.1.3 Gramm-Leach-Bliley

The Financial Modernization Act of 1999, aka Gramm-Leach-Bliley Act (GLBA): designed to protect consumers’ personal financial information (PFI).

### 3.1.4 HIPAA and HITECH

Healthcare Insurance Portability and Accountability Act (HIPAA) deals with personal health information (PHI). The Health Information Technology for

Economic and Clinical Health Act (HITECH Act) is part of the American Recovery

and Reinvestment Act of 2009 (ARRA),

### 3.1.5 Payment Card Industry Data Security Standard (PCI DSS)

Payment Card Industry PCI standards: Data Security Standard (PCI DSS), Payment Application Data Security Standard (PA DSS), PIN Transaction Security (PTS)

### 3.1.6 Other Regulations

Federal Financial Institutions Examination Council (FFIEC) rules

## 3.2 Legal Issues

### 3.2.1 Intellectual Property

* Patents
* Copyrights
* Trademarks
* Trade Secrets
* Warranty

## 3.3 Privacy

### 3.3.1 Privacy Policy

### 3.3.2 Personally Identifiable Information (PII)

### 3.3.3 Personal Health Information (PHI)

### 3.3.4 Breach Notifications

### 3.3.5 Data Protection Principles

* **Safe Harbor Principles**: allowed non-EU firms to deal with the EUDPD by following 7 elements: •**Notice**: Customers be informed that data is being collected & how it will be used •**Choice**: Customers can opt out of collection & forward transfer of data to 3rd parties •**Onward Transfer**: Transfers of data to 3rd parties may only occur to other organizations that follow adequate data protection principles •**Security**: Reasonable efforts be made to prevent loss of collected information •**Data Integrity**: Data be relevant & reliable for purpose it was collected for •**Access**: Customers be able to access information held about them and correct/delete if inaccurate •**Enforcement**: Effective means of enforcing these rules
* **GDPR Personal Data Elements**: personal data = information relating to identified/ identifiable natural person, including • online identifiers • IP addresses • and cookies

### 3.3.6 California Consumer Privacy Act 2018 (AB 375)

## 3.4 Security Standards

3.4.1 ISO

3.4.2 NIST

## 3.5 Secure Software Architecture

3.5.1 Security Frameworks

## 3.6 Trusted Computing

3.6.1 Principles

3.6.2 Trusted Computing Base

3.6.3 Trusted Platform Module

3.6.4 Microsoft Trustworthy Computing Initiative

## 3.7 Acquisition

3.7.1 Definitions and Terminology

3.7.2 Build vs. Buy Decision

3.7.3 Outsourcing

3.7.4 Contractual Terms and Service Level Agreements

# Ch4 Software Development Methodologies

## 4.1 Secure Development Lifecycle

4.1.1 Principles

4.1.2 Security vs. Quality

## 4.2 Security Features != Secure Software

4.2.1 Secure Development Lifecycle Components

4.2.2 Software Team Awareness and Education

4.2.3 Gates and Security Requirements

4.2.4 Bug Tracking

4.2.5 Threat Modeling

4.2.6 Fuzzing

4.2.7 Security Reviews

4.2.8 Mitigations

## 4.3 Software Development Models

4.3.1 Waterfall

4.3.2 Spiral

4.3.3 Prototype

4.3.4 Agile Methods

4.3.5 Open Source

## 4.4 Microsoft Security Development Lifecycle

4.4.1 History

4.4.2 SDL Foundation

4.4.3 SDL Components

# Part II Secure Software Requirements

# Ch5 Policy Decomposition

## 5.1 Confidentiality, Integrity, Availability Requirements

5.1.1 Confidentiality

5.1.2 Integrity

5.1.3 Availability

## 5.2 Authentication, Authorization, Auditing Requirements

5.2.1 Identification and Authentication

5.2.2 Authorization

5.2.3 Access Control Mechanisms

5.2.4 Auditing

## 5.3 Internal & External Requirements

5.3.1 Internal

5.3.2 External

# Ch6 Data Classification and Categorization

## 6.1 Data Classification

6.1.1 Data States

6.1.2 Data Usage

6.1.3 Data Risk Impact

## 6.2 Data Ownership

6.2.1 Data Owner

6.2.2 Data Custodian

## 6.3 Labeling

6.3.1 Sensitivity

6.3.2 Impact

## 6.4 Types of Data

6.4.1 Structured

6.4.2 Unstructured

## 6.5 Data Lifecycle

6.5.1 Generation

6.5.2 Retention

6.5.3 Disposal

# Ch7 Requirements

## 7.1 Functional Requirements

7.1.1 Role and User Definitions

7.1.2 Objects

7.1.3 Activities/Actions

7.1.4 Subject-Object-Activity Matrix

7.1.5 Use Cases

7.1.6 Abuse Cases (Inside/Outside Adversaries)

7.1.7 Sequencing and Timing

7.1.8 Secure Coding Standards

## 7.2 Operational Requirements

7.2.1 Deployment Environment

## 7.3 Requirements Traceability Matrix

## 7.4 Connecting the Dots

# Part III Secure Software Design

# Ch8 Design Processes

Secure Development Lifecycle (SDL)

## 8.1 Attack Surface Evaluation

### 8.1.1 Attack Surface Measurement

#### MS WINDOWS attack surface elements (17)

•Open sockets •Open remote procedure call (RPC) endpoints •Open named pipes •Services •Services running by default •Services running as SYSTEM •Active web handlers (ASP files, HTR files, etc) •Active Internet Server Application Programming Interface (ISAPI) filters •Dynamic webpages (ASP and such) •Executable virtual directories •Enabled accounts •Enabled accounts in admin group •Null sessions to pipes & shares •Guest account enabled •Weak ACLs in file system •Weak ACLs in registry •Weak ACLs on shares

### 8.1.2 Attack Surface Minimization

⬩Turn off ⬩Minimization form of least privilege ⬩Calculated in development ⬩Use attack surface as surrogate

## 8.2 Threat Modeling

Threat modeling = process to identify and document system threats. Part of description include mitigating actions that resolve the exposure.

### 8.2.1 Threat Model Development

#### 8.2.1.1 Identify Security Objectives

#### 8.2.1.2 System Decomposition

DFD Elements for Threat Modeling: External Entities •Users (by type) •Other systems Data Stores •Files •Database •Registry •Shared memory •Queues/stack Trust Boundaries •Users •File systems •Process boundaries Data Flows •Function calls •Remote procedure calls (RPCs) •Network traffic

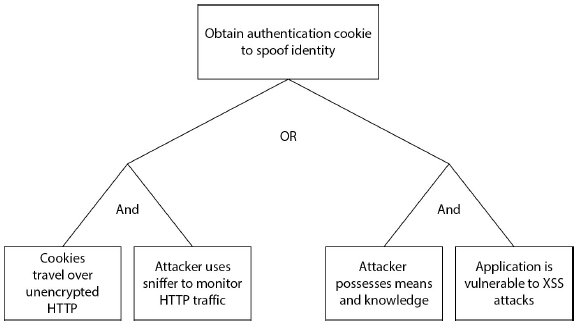
#### 8.2.1.3 Threat Identification

##### STRIDE Method

|  |  |
| --- | --- |
| Threat | Security Property |
| Spoofing  Tampering  Repudiation  Information disclosure  Denial of service  Elevation of privilege | Authentication  Integrity  Nonrepudiation  Confidentiality  Availability  Authorization |

#### 8.2.1.4 Mitigation Analysis

##### FIGURE 8-1 Attack tree



**DREAD Model**: damage potential, reproducibility, exploitability, affected users, and discoverability

#### 8.2.1.5 Threat Model Validation

## 8.3 Control Identification & Prioritization

## 8.4 Risk Assessment for Code Reuse

## 8.5 Documentation

## 8.6 Design & Architecture Technical Review

# Ch9 Design Considerations

## 9.1 Application of Methods to Address Core Security Concepts

### 9.1.1 Confidentiality, Integrity, Availability

**Confidentiality** = preventing the disclosure of information to unauthorized parties

**Integrity** = protecting data from unauthorized alteration (changing a value or deleting a value). Integrity builds upon confidentiality, for if data is to be altered or deleted, then it is probably also visible to the user account ⇨ Use ACL, hashing

**Availability** = system being available to authorized users when appropriate

### 9.1.2 Authentication, Authorization, Auditing

**Authentication** = verify to computer system or network that the individual is who they claim to be. **Authorization** applies the predetermined access levels to the user. **Accounting** = function of measuring specific IT activities. Typically done through the logging of crucial data elements of an activity as it occurs

### 9.1.3 Secure Design Principles

#### 9.1.3.1 Good Enough Security

#### 9.1.3.2 Least Privilege

#### 9.1.3.3 Separation of Duties

#### 9.1.3.4 Defense in Depth

#### 9.1.3.5 Fail Safe

#### 9.1.3.6 Economy of Mechanism

#### 9.1.3.7 Complete Mediation

#### 9.1.3.8 Open Design

#### 9.1.3.9 Least Common Mechanism

#### 9.1.3.10 Psychological Acceptability

#### 9.1.3.11 Weakest Link

#### 9.1.3.12 Leverage Existing Components

#### 9.1.3.13 Single Point of Failure

### 9.1.4 Interconnectivity

#### 9.1.4.1 Session Management

#### 9.1.4.2 Exception Management

#### 9.1.4.3 Configuration Management

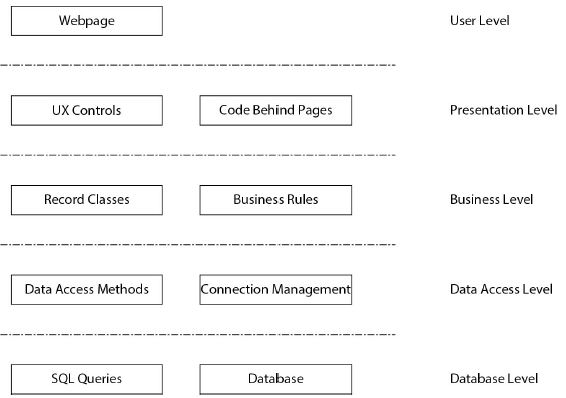
## 9.2 Interfaces

# Ch10 Securing Commonly Used Architecture

## 10.1 Distributed Computing

### 10.1.1 Client Server

##### FIGURE 10-1 N-tier architecture



### 10.1.2 Peer-to-Peer

File sharing

### 10.1.3 Message Queuing

Message queuing technology = use of intermediate server that mediates transmission and delivery of information between processes; constructed to manage guaranteed delivery, logging, and security of the data flows

## 10.2 Service-Oriented Architecture

**Service-oriented architecture (SOA)** = distributed architecture with

characteristics: •Platform neutrality •Interoperability •Modularity & reusability

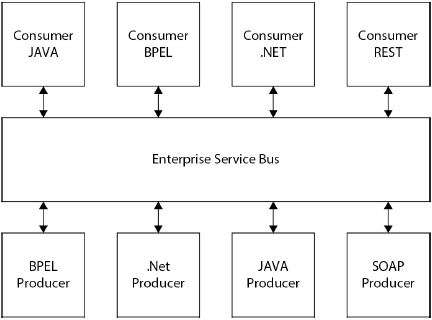
•Abstracted business functionality •Contract-based interfaces •Discoverability

**Technologies**: common object model (COM), common object request broker architecture (CORBA), and web services (WS) **Messaging methodology**: secured by XML encryption or transport over secure channels (SSL/TLS) **Protocols**: SOAP (Simple Object Access Protocol), REST (Representational State Transfer)

### 10.2.1 Enterprise Service Bus (ESB)

ESB can be configured to: •Perform protocol conversions and handle translation and transformation of communications •Handle defined events •Perform message queuing and mapping of data flows. ESB acts as conduit between protocols. Each connector operates through an adapter enabling the cross-communication between protocols: XML, EDI, WSDL, REST, DCOM, CORBA, etc

#### FIGURE 10-2 Enterprise service bus



### 10.2.2 Web Services

Web services characterized by a machine-readable format referred to as **Web Services Description Language (WSDL)** = XML-based interface describing functionality offered by a web service (how service can be called, what parameters it expects, and what data structures it returns)

#### 10.2.2.1 REST = Representative State Transfer (REST)

#### 10.2.2.2 JSON = JavaScript Object Notation

#### 10.2.2.3 UDDI = Universal Description, Discovery, and Interface

Designed as a protocol-based registry through which services worldwide can list

themselves on the Internet. Includes a mechanism to register and locate web service applications

## 10.3 Rich Internet Applications

**Rich Internet applications (RIAs)** = form of architecture that use the Web as a

transfer mechanism and the client as a processing device. Example: Facebook

Frameworks: Adobe Flash, Java, Microsoft Silverlight, HTML5/JavaScript

### 10.3.1 Client-Side Exploits or Threats

Never trust input without validation

### 10.3.2 Remote Code Execution

Triggering arbitrary code execution from another machine across network

## 10.4 Pervasive/Ubiquitous Computing

### 10.4.1 Wireless

Protocols: cellular-based systems, 802.11 Wi-Fi, 802.15 Zigbee, Bluetooth, Wi-Max

### 10.4.2 Location-Based

### 10.4.3 Constant Connectivity

### 10.4.4 Radio Frequency Identification (RFID)

### 10.4.5 Near-Field Communication (NFC)

### 10.4.6 Sensor Networks

## 10.5 Mobile Applications

## 10.6 Integration with Existing Architectures

## 10.7 Cloud Architectures

•On-demand self-service •Broad network access •Resource pooling •Rapid elasticity •Measured service

### 10.7.1 Software as a Service

### 10.7.2 Platform as a Service

### 10.7.3 Infrastructure as a Service

# Ch11 Technologies

## 11.1 Authentication & Identity Management

**Identity Management (IDM)** = set of policies, processes, and technologies for managing digital identity information **Identity and access management (IAM)** associated with comprehensive set of policies, processes, and technologies for managing digital identity information

### 11.1.1 Identity Management

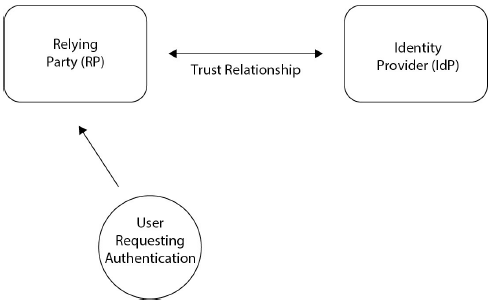
**Identity management** = set of processes associated with the identity lifecycle, including the provisioning, management, and deprovisioning of identities[[1]](#footnote-1)

### 11.1.2 Authentication

**Authentication** = process of verifying that a user is who they claim to be and

applying the correct values in the access control system. **Federated ID systems** allow users to connect to systems through known systems (Facebook). 2 main parties: ➊**relying party (RP)** ➋**identity provider (IdP)**.

##### FIGURE 11-1 RP and IdP relationships

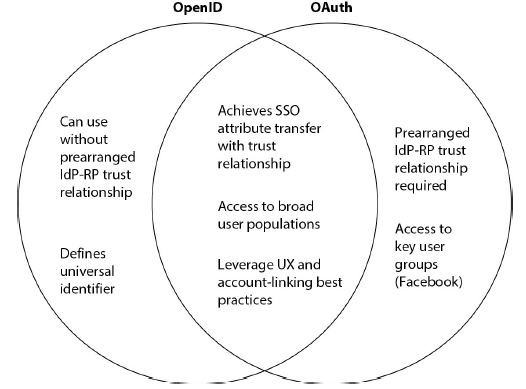


2 systems: ➊**OpenID** created federated authentication; allows 3rd party to authenticate users for you; enables websites/ applications (consumers) to grant access to their own applications by using another service or application (provider) for authentication ➋**OAuth** created to eliminate the need for users to share their passwords with 3rd-party applications. The OAuth protocol enables websites or applications (consumers) to access protected resources from a web service (service provider) via an application programming interface (API), without requiring users to disclose their service provider credentials to the consumers. At

their core, both protocols have an assertion verification method. They differ in that OpenID is limited to the “this is who I am” assertion, while OAuth provides an

“access token” that can be exchanged for any supported assertion via an API

##### FIGURE 11-2 OpenID vs. Oauth



## 11.2 Credential Management

Each set of credentials, regardless of source, requires safekeeping by the receiving entity. Managing includes tasks: credential generation, storage, synchronization, reset, and revocation; all of these activities should be logged.

### 11.2.1 X.509 Credentials

**X.509** = series of standards for certificates to transfer asymmetric keys between parties. Digital certificate binds an individual’s identity to public key; contains information for receiver to assure identity of public key owner. Registration authority (RA) verifies individual’s identity ⇨ certificate authority (CA) generates digital certificate, with information to facilitate authentication. Parties involved in securing PKI solution: certificate authorities, registration authorities, certificate revocation mechanisms: Certificate Revocation Lists (CRLs) or Online Certificate Status Protocol (OCSP). **X.509 Digital Certificate Fields**: •**Version number** of X.509 standard; indicates format + fields to use. •**Serial number** unique id of certificate issued by CA •**Signature algorithm** specifies hashing & digital signature algorithms to digitally sign certificate •**Issuer** •**Validity** dates certificate valid for use •**Subject** owner of certificate •**Public key** bound to certified subject; identifies algorithm to create private/public key pair •**Certificate usage** approved use of certificate (intended use of public key) •**Extensions** additional data (expanded in X.509 v3) to be encoded into certificate to expand functionality

### 11.2.2 Single Sign-On

2 methods: ➊**Kerberos** ➋**Security Assertion Markup Language (SAML)**

## 11.3 Flow Control (Proxies, Firewalls, Middleware)

### 11.3.1 Firewalls

**Firewalls** operate on a packet level: stateless or stateful.

### 11.3.2 Proxies

**Proxies** similar to firewalls (mediate traffic flows); differ - act as middlemen. Traffic from untrusted sources terminated at a proxy, where the traffic is received and processed. If the traffic meets the correct rules, it can then be forwarded on to the intended system. Proxies come in a wide range of capabilities, from simple to very complex, both in their rule-processing capabilities and additional functionalities. One of these functionalities is caching—a temporary local storage of web information that is frequently used and seldom changed, like images. In this role, a proxy acts as a security device and a performance-enhancing device.

### 11.3.3 Application Firewalls

Application firewalls act as application-specific gateways between users and web-based applications. Acting as a firewall proxy, web application firewalls can monitor traffic in both directions, client to server and server to client, watching for anomalies. Web application firewalls act as guards against both malicious intruders and misbehaving applications.

### 11.3.4 Queuing Technology

Message transport from sender to receiver can be done either synchronously or

asynchronously, and either have guaranteed transport or best effort. Internet

protocols can manage the guarantee/best effort part, but a separate mechanism is needed if asynchronous travel is permissible. Asynchronous transport can alleviate network congestion during periods where traffic flows are high and can assist in the prevention of losing traffic due to bandwidth restrictions.

## 11.4 Logging

➊Compliance programs—HIPAA, SOX, PCI DSS, EOC, etc.—have logging

requirements ➋Incident response

### 11.4.1 Syslog

**Syslog** = Internet Engineering Task Force (**IETF**)–approved protocol for log

messaging; designed and built around UNIX to send log information across an IP network; in native form, uses **User Datagram Protocol (UDP)** and transmits information in the clear; wrappers that provide **Transport Layer Security (TLS)–**based security and TCP-based communication guarantees.

## 11.5 Data Loss Prevention

**Data loss prevention (DLP)** act by screening traffic, looking for traffic that

meets profile parameters: size of transfer, destination, data elements that are protected. If any of these elements are detected, then a data exfiltration event is in progress and the connection is terminated.

## 11.6 Virtualization

Benefits: •Reduced cost of servers resulting from server consolidation •Improved operational efficiencies from administrative ease of certain tasks •Improved portability and isolation of applications, data, and platforms •Operational agility to scale environments, i.e., cloud computing

## 11.7 Digital Rights Management

**Digital rights management (DRM**) for content owners to exert control over digital content; not just about copy protection, but also about usage rights, authenticity,

and integrity; can allow a file to be shared, but not edited or printed; can restrict content to a specific piece of hardware. Three entities in the DRM relationship: users, contents, and rights. Formal language: **Rights Expression Language (REL)** XML based and designed to convey rights in a machine-readable form; define the license and describe the terms of the permissions or restrictions they imply for how the related content may then be used by a system. RELs: •**ccREL** = RDF schema used by the Creative Commons project and the GNU project to express their general public license (GPL) in machinereadable form •**ODRL** (Open Digital Rights Language) = open standard for an XML-based REL •**MPEG-21 Part 5** of this MPEG standard includes an REL •**XrML eXtensible rights Markup Language** based on work at Xerox in the 1990s.

## 11.8 Trusted Computing

Promoted by Trusted Computing Group, designed to ensure that the computer behaves in a consistent and expected manner. Key element: **Trusted Platform Module (TPM)**, a hardware interface for security operations

### 11.8.1 TCB

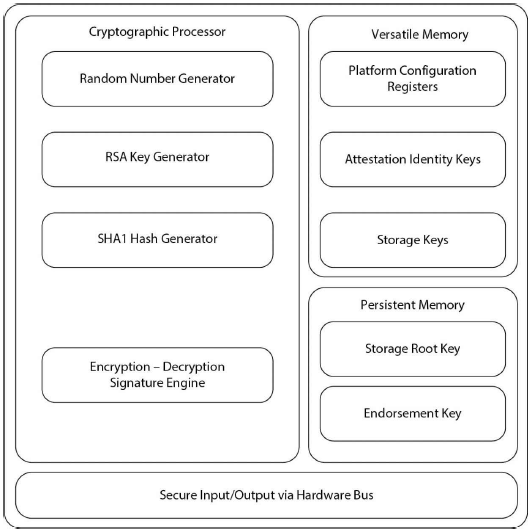
**Trusted computing base (TCB)** of computer system = set of all hardware,

firmware, and/or software components that are critical to its security. Key issue: **privilege escalation -** If any element of the computer system has the ability to effect an increase in privilege without it being authorized, then this would be a violation of the security, and this part of the system would be part of the TCB - foundation for security principles such as **complete mediation**.

### 11.8.2 TPM

**Trusted Platform Module** = HW implementation of cryptographic functions on a computer’s motherboard. Intent: to provide a base level of security that is deeper than the operating system and virtually tamperproof from the software side of the machine; can hold an encryption key that is not accessible to the system except through the TPM chip.

##### FIGURE 11-4 TPM hardware functions



### 11.8.3 Malware

Malware requires a vulnerability in a SW system to gain its initial infection vector

### 11.8.4 Code Signing

**Code signing** = application of digital signature technology to computer code; provide useful information: establishes who provider or author of software; integrity level of the code (altered since signing)?

## 11.9 Database Security

### 11.9.1 Encryption

Primary keys used to index and join tables ⇨ cannot be obfuscated or encrypted ⇨ not to use personally identifiable information (PII) and personal health information (PHI) as keys in a database structure. Considerations for a database encryption strategy: •Level of risk classification associated with data? •Usage pattern of the data—how is it protected in transit and in use? •Differing classification across elements of the data? •How encryption handled in enterprise for other projects? •Available encryption options to development team?

### 11.9.2 Triggers

### 11.9.3 Views

### 11.9.4 Privilege Management

## 11.10 Programming Language Environment

### 11.10.1 CLR

.NET languages compiled into **common intermediate language (CIL)**, aka **MSF Intermediate Language (MSIL)**, executed in JIT compiler **common language runtime (CLR)**, which can insert traps, garbage collection, type safety, index checking, sandboxing, etc. - provides functional & stable execution environment.

### 11.10.2 JVM

Java language source code is compiled to an intermediate stage: **byte code** - similar to processor instruction codes, but not executable directly. The target machine has a **Java Virtual Machine (JVM)** that executes the byte code. The Java architecture = **Java Runtime Environment (JRE)**, composed of: JVM + set of standard class libraries: **Java Class Library**. Together, these elements provide for the managed execution of Java on the target machine

### 11.10.3 Compiler Switches

Compiler switches enable development to control how compiler handles aspects of program construction. Options: manipulating elements such as memory, stack protection, and exception handling. Flags enable development force specific

behaviors: the **/GS flag** enables a security cookie on the stack to prevent stack-based overflow attacks; the **/SAFEH switch** enables a safe exception handling table option that can be checked at runtime

### 11.10.4 Sandboxing

**Sandboxing** = execution of computer code in an environment designed to isolate the code from direct contact with the target system; used to execute untrusted code, code from guests, and unverified programs

### 11.10.5 Managed vs. Unmanaged Code

**Managed code** (e.g., (.NET and Java) executed in intermediate system with range of controls: sandboxing, garbage collection, index checking, type safe, memory management, and multiplatform capability. **Unmanaged code** executed directly on target OS; always compiled to specific target system; performance advantages; memory allocation, type safety, garbage collection, etc., done by developer

## 11.11 Operating Systems

## 11.12 Embedded Systems

### 11.12.1 Control Systems

**Control systems** = specialized computer systems for automated control of equipment; referred to by many names, including SCADA, Industrial Control Systems, and operational technology (OT). Wide range of types: programmable

logic controllers (PLCs), remote terminal units (RTUs). Commonly referred to as supervisory control and data acquisition (SCADA) systems; viewed as form of embedded system, for they are integrated into a physical environment for

the sole purpose of providing computer control in that environment

### 11.12.2 Firmware

Firmware held in nonvolatile memory, read-only memory (ROM), erasable programmable readonly memory (EPROM), or in flash memory; can hold operational code base (SW component of system); In computers, firmware acts as step 1 in startup process. In personal computers, the **basic input output**

**system (BIOS)** is the firmware-based interface between the hardware and operating system. BIOS was replaced by a more advanced version, called the **unified extensible firmware interface (UEFI)**

# Part IV Secure Software Implementation/ Programming

# Ch12 Common Software Vulnerabilities & Countermeasures

## 12.1 CWE/SANS Top 25 Vulnerability Categories

#### Insecure Interaction Between Components (6)

Insecurely send/receive data to components, modules, programs, processes, threads, systems: [CWE-89](http://cwe.mitre.org/top25/index.html#CWE-89) :Improper Neutralization of Special Elements in SQL ('SQL Injection') [CWE-78](http://cwe.mitre.org/top25/index.html#CWE-78): Improper Neutralization of Special Elements in OS Command ('OS Command Injection') [CWE-79](http://cwe.mitre.org/top25/index.html#CWE-79): Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') [CWE-434](http://cwe.mitre.org/top25/index.html#CWE-434): Unrestricted Upload of File with Dangerous Type [CWE-352](http://cwe.mitre.org/top25/index.html#CWE-352): Cross-Site Request Forgery (CSRF) [CWE-601](http://cwe.mitre.org/top25/index.html#CWE-601): URL Redirection to Untrusted Site ('Open Redirect')

#### Risky Resource Management (8)

SW not properly manage creation, usage, transfer, destruction of important system resources: [CWE-120](http://cwe.mitre.org/top25/index.html#CWE-120): Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') [CWE-22](http://cwe.mitre.org/top25/index.html#CWE-22): Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') [CWE-494](http://cwe.mitre.org/top25/index.html#CWE-494): Download of Code Without Integrity Check [CWE-829](http://cwe.mitre.org/top25/index.html#CWE-829): Inclusion of Functionality from Untrusted Control Sphere [CWE-676](http://cwe.mitre.org/top25/index.html#CWE-676): Use of Potentially Dangerous Function [CWE-131](http://cwe.mitre.org/top25/index.html#CWE-131): Incorrect Calculation of Buffer Size [CWE-134](http://cwe.mitre.org/top25/index.html#CWE-134): Uncontrolled Format String [CWE-190](http://cwe.mitre.org/top25/index.html#CWE-190): Integer Overflow or Wraparound

#### Porous Defenses (11)

Misused, abused, or ignored defensive techniques: [CWE-306](http://cwe.mitre.org/top25/index.html#CWE-306): Missing Authentication for Critical Function [CWE-862](http://cwe.mitre.org/top25/index.html#CWE-862): Missing Authorization [CWE-798](http://cwe.mitre.org/top25/index.html#CWE-798): Use of Hard-coded Credentials [CWE-311](http://cwe.mitre.org/top25/index.html#CWE-311): Missing Encryption of Sensitive Data [CWE-807](http://cwe.mitre.org/top25/index.html#CWE-807): Reliance on Untrusted Inputs in Security Decision [CWE-250](http://cwe.mitre.org/top25/index.html#CWE-250): Execution w/ Unnecessary Privileges [CWE-863](http://cwe.mitre.org/top25/index.html#CWE-863): Incorrect Authorization [CWE-732](http://cwe.mitre.org/top25/index.html#CWE-732): Incorrect Permission Assignment for Critical Resource [CWE-327](http://cwe.mitre.org/top25/index.html#CWE-327): Broken or Risky Cryptographic Algorithm [CWE-307](http://cwe.mitre.org/top25/index.html#CWE-307): Improper Restriction of Excessive Authentication Attempts [CWE-759](http://cwe.mitre.org/top25/index.html#CWE-759): Use of a One-Way Hash without a Salt

## 12.2 OWASP Vulnerability Categories

#### Open Web Application Security Project (OWASP) Top 10

➊Injection ➋Broken Authentication ➌Sensitive Data Exposure ➍XML External Entities (XEE) ➎Broken Access Control ➏Security Misconfiguration ➐Cross-Site Scripting ➑Insecure Deserialization ➒Using Components With Known Vulnerabilities ➓Insufficient Logging And Monitoring

## 12.3 Common Vulnerabilities & Countermeasures

### 12.3.1 Injection Attacks

Attacks include: ➊SQL Injection ➋OS Command Injection ➌Integer Overflow or Wraparound ➍Path Traversal (dot-dot-slash, directory traversal, directory

climbing, backtracking attacks) ➎Cross-Site Scripting (XSS) ➏Cross-Site Request Forgery (CSRF). Can be used against LDAP, XML, and other common protocols

### 12.3.2 Cryptographic Failures

Attacks include: ➊Hard-Coded Credentials ➋Missing Encryption of Sensitive Data ➌Use of a Broken or Risky Cryptographic Algorithm ➍Download of Code Without Integrity Check ➎Use of a One-Way Hash Without a Salt

## 12.4 Input Validation Failures

Input validation well suited for vulnerabilities: Buffer Overflow, Reliance on Untrusted Inputs in a Security Decision, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Path Traversal, and Incorrect Calculation of Buffer Size

### 12.4.1 Buffer Overflow

Classification of buffer overflows includes: static buffer overruns, indexing errors, format string bugs, Unicode and ANSI buffer size mismatches, heap overruns

### 12.4.2 Canonical Form

Canonicalization = process by which application programs manipulate strings to a base form, creating a foundational representation of the input

### 12.4.3 Missing Defense Functions

Vulnerabilities = Missing Authentication for Critical Functions, Missing

Authorization, Unrestricted Upload of File with Dangerous Type, Incorrect

Authorization, Incorrect Permission Assignment for Critical Resource, Execution

with Unnecessary Privileges, Improper Restriction of Excessive Authentication

Attempts, URL Redirection to Untrusted Site (“Open Redirect”), Uncontrolled

Format String.

### 12.4.4 General Programming Failures

## 12.5 Common Enumerations

Enumerations of known software weaknesses & vulnerabilities compiled & published in MITRE Corporation’s “Making Security Measureable” program

### 12.5.1 Common Weakness Enumerations (CWE)

Structured list of identifying information, including the time of introduction of a weakness, the location of the weakness (configuration, code, or environment), the intent of the weakness, and other information.

### 12.5.2 Common Vulnerabilities and Exposures (CVE)

List of standard identifiers for known software

## 12.6 Virtualization

## 12.7 Embedded Systems

## 12.8 Side Channel

From cryptography: attack against implementation of a cryptosystem, rather than the strength of the algorithm itself (e.g., cold booting).

## 12.9 Social Engineering Attacks

### 12.9.1 Phishing

# Ch13 Defensive Coding Practices

## 13.1 Declarative vs. Programmatic Security

**Declarative security** = define security relations with respect to container ⇨ More flexible ⇨ Security rules configured as part of deployment and not code ⇨ Security managed by operation not development. **Imperative programming**, aka **programmatic security** = security implementation embedded in code⇨ More granular ⇨ Enforce complex business rules for specific conditions ⇨ code less portable or reusable

### 13.1.1 Bootstrapping

Bootstrapping = power on self-test (POST) routines, boot loaders, o/s initialization activities. Can be interrupted by malicious s/w with hooks into o/s ⇨ Ensure values not changed outside application control ⇨ Use configuration files

### 13.1.2 Cryptographic Agility

Cryptographic agility = ability to manage specifics of cryptographic function through a configuration file without recompiling. When communications between elements involve sessions—unique communication channels tied to transactions or users—secure the session to prevent failures that can cascade into unauthorized activity. Session management requires sufficient security provisions to guard against attacks such as brute-force, man-in-the-middle, hijacking, replay, and prediction attacks

### 13.1.3 Handling Configuration Parameters

## 13.2 Memory Management

Memory holds operational code, data, variables, working space. Management complex because dynamic nature of usage of memory across single/ multiple programs, and operating system. Allocation and management of memory is responsibility of operating systems and the application. Managed code applications: combination of managed code and intermediate code execution engine takes care of memory management, and type safety makes the tasking easier. In addition: automatic lifetime control over all resources (the code runs in a sandbox environment and runtime engine maintains control over all resources). Unmanaged code situations: responsibility for memory management is shared between operating system and application. More difficult because issues associated with variable type mismatch. All operations associated with resources and memory are responsibility of developer, including garbage collection, thread pooling, memory overflows, and more. Complexity is enemy of security.

### 13.2.1 Type-Safe Practice

Type safety ⇨ programming language prevents errors resulting from different data types in a program. Enforced either statically at compile time or dynamically at runtime to prevent errors. Type safety linked to memory safety. Type-safe code will not inadvertently access arbitrary locations of memory outside the expected memory range. Type safety defines all variables, hence the memory lengths: Resolves many memory-related issues automatically.

### 13.2.2 Locality

Given memory reference by program, subsequent memory accesses predictable and in proximity to previous references ⇨ Buffer overflows + Memory attacks. Defenses: Address Space Layout Randomization (ASLR) by Microsoft

## 13.3 Error Handling

Logging errors: do not expose sensitive information e.g. personally identifiable information (PII). On screen or terminal: disclosing paths, locations, passwords, userids, etc. could be useful to an adversary.

### 13.3.1 Exception Management

Exception management = programmatic response to exception; exceptions should be handled by special functions: Exception handlers designed to specifically address known exceptions and handle them according to pre-established business rules. Broad classes of exceptions trapped and handled by software (e.g., Arithmetic overflows). Properly coded for, trapped, and handled with business logic, this type of error can be handled inside software itself instead of operating system. In development, examine how application could fail, and also the correct ways to address those failures. This is a means of defensive programming: if the exceptions are not trapped and handled by the application, they will be handled by the operating system, which does not have the embedded knowledge to properly handle the exceptions. Exceptions are not security issues. However, unhandled exceptions can become security issues.

## 13.4 Interface Coding

Application programming interfaces (APIs) connect functionality of various modules. APIs are significant because they represent entry points into software. The attack surface analysis and threat model should identify the APIs that could be attacked and the mitigation plans to limit the risk. Third-party APIs that are being included as part of the application should also be examined, and errors or issues be mitigated as part of the SDL process. Older, weak, and deprecated APIs should be identified and not allowed into the final application. On all interface inputs into your application, it is important to have the appropriate level of authentication. It is also important to audit the external interactions for any privileged operations performed via an interface.

## 13.5 Primary Mitigations

Standard best practice–based primary mitigations:

• Lock down your environment.

• Establish and maintain control over all of your inputs.

• Establish and maintain control over all of your outputs.

• Assume external components can be subverted/ code can be read by anyone.

• Use libraries/ frameworks that make it easier to avoid introducing weaknesses.

• Use industry-accepted security features instead of inventing your own.

• Integrate security into the entire software development lifecycle.

• Use broad mix of methods to comprehensively find & prevent weaknesses.

Defensive coding not black art ⇨ apply materials detailed in threat report. Foundational elements: attack surface reduction, understanding of common coding vulnerabilities, standard mitigations, code analysis/ review, versioning, cryptographic agility, memory management, exception handling, interface coding, and managed code. Concurrency = process of threads in a program executing concurrently. Can be an issue when threads access common object, creating a shared object property. Should they change the state of the shared object, the conditions for a race condition[[2]](#footnote-2) apply. Controlling concurrency is one method of controlling for race conditions. Tokenization = replace sensitive data with data that has no external connection to the sensitive data. Credit card transaction example: credit card number and expiration date considered sensitive and are not to be stored, so restaurants typically print only the last few digits with XXXXs for the rest, creating a token for the data, but not disclosing the data.

## 13.6 Learning from Past Mistakes

Learning from others and adding their failures to your own list of failures to avoid is a good business practice. Part of the role of the security team is keeping the list of security requirements up to date for projects. Examining errors from other companies and updating your own set of security requirements to prevent your firm from falling into known pitfalls will save time and money in the long run.

# Ch14 Secure Software Coding Operations

## 14.1 Code Analysis (Static & Dynamic)

### 14.1.1 Static

Static code analysis = code is examined without being executed. Performed on source and object code bases. Source code = original code base (high-level language code or machine code). Performed by humans (high-level language), tools (any form of code base). Tools = source code analyzers, binary scanners or byte code scanners. Tools can check syntax, approved function/library calls; examine rules and semantics associated with logic and calls.

### 14.1.2 Dynamic

Dynamic analysis performed while software is executed. The system is fed specific test inputs designed to produce specific forms of behaviors. Dynamic analysis important on systems such as embedded systems (high degree of operational autonomy). Dynamic analysis requires specialized automation: dynamic test suites designed to monitor operations for programs that have high degrees of parallel functions; thread-checking routines to ensure multicore processors and software are managing threads correctly. There are programs designed to detect race conditions and memory addressing errors.

## 14.2 Code/Peer Review

###### Table 14-1 Issues for Code Reviews

## 14.3 Build Environment

Microsoft’s SDL guidelines have required settings for compilers, linkers, and code analysis tools. Enabling these options will result in more work earlier in the process, but will reduce the potential for errors later in the development process, where remediation is more time consuming and expensive. In addition to the actual tools used for building, define safe libraries for common problem functions such as buffer overflows, XSS, and injection attacks. Examples of these libraries are the OWASP Enterprise Security API project and the Microsoft Anti-Cross Site Scripting Library for .NET.

### 14.3.1 Integrated Development Environment (IDE)

Automated tools can be built into the integrated development environment, to do both forms of static and dynamic checking automatically. Using automation such as a modern IDE is an essential part of an SDL.

## 14.4 Antitampering Techniques

Code signing involves applying a digital signature to code, providing mechanism where the end user can verify the code integrity. Requires established public key infrastructure. Developer needs a key pair, which needs be signed by a recognized certificate authority. Automatic update services, such as Microsoft’s Windows Update service, use code signing technologies. This technology is built into the update application, requiring no specific interaction from the end user. Code signing provides a means of authenticating the source and integrity of code. It cannot ensure that code is free of defects or bugs. Steps to Code Signing:

1. The code author uses a one-way hash of the code to produce a digest.

2. The digest is encrypted with the signer’s private key.

3. The code and the signed digest are transmitted to end users.

4. The end user produces a digest of the code using the same hash function

as the code author.

5. The end user decrypts the signed digest with the signer’s public key.

6. If the two digests match, the code is authenticated and integrity is assured.

Code signing should be used for all software distribution, and is essential when

the code is distributed via the Web. End users should not update or install software without some means of verifying the proof of origin and the integrity of the code being installed. Code signing will not guarantee that the code is defect free; it only demonstrates that the code has not been altered since it was signed and identifies the source of the code.

## 14.5 Configuration Management: Source Code & Versioning

Managing the versions and changes associated is version control, or revision

control. The version control system can manage access to source files, locking

sections of code so that only one developer at a time can check out and modify

pieces of code. This can also be done by allowing multiple edits and performing a version merge of the changes, although this can create issues if collisions are not properly managed by the development team. Configuration management and version control operations are highly detailed, with lots of recordkeeping. Management of this level of detail is best done with an automated system that removes human error from the operational loop. The level of detail across the breadth of a development team makes automation the only way in which this can be done in an efficient and effective manner. A wide range of software options are available to a development team to manage this information. Once a specific product is chosen, it can be integrated into the SDL process to make its use a nearly transparent operation from the development team’s perspective.

# Part V Secure Software Testing

# Ch15 Security Quality Assurance Testing

## 15.1 Standards for Software Quality Assurance

15.1.1 ISO 9216

15.1.2 SSE-CMM

15.1.3 OSSTMM

## 15.2 Testing Methodology

## 15.3 Functional Testing

15.3.1 Unit Testing

15.3.2 Integration or Systems Testing

15.3.3 Performance Testing

15.3.4 Regression Testing

## 15.4 Security Testing

15.4.1 White-Box Testing

15.4.2 Black-Box Testing

15.4.3 Grey-Box Testing

## 15.5 Environment

15.5.1 Bug Tracking

15.5.2 Defects

15.5.3 Errors

15.5.4 Vulnerabilities

15.5.5 Bug Bar

## 15.6 Attack Surface Validation

## 15.7 Testing Artifacts

## 15.8 Test Data Lifecycle Management

# Ch16 Security Testing

## 16.1 Scanning

16.1.1 Attack Surface Analyzer

## 16.2 Penetration Testing

## 16.3 Fuzzing

## 16.4 Simulation Testing

## 16.5 Testing for Failure

## 16.6 Cryptographic Validation

16.6.1 FIPS 140-2

## 16.7 Regression Testing

## 16.8 Impact Assessment and Corrective Action

# Part VI Secure Lifecycle Management

# Ch17 Secure Lifecycle Management

## 17.1 Introduction to Acceptance

17.1.1 Software Qualification Testing

17.1.2 Qualification Testing Plan

17.1.3 Qualification Testing Hierarchy

## 17.2 Pre-release Activities

17.2.1 Implementing the Pre-release Testing Process

17.2.2 Completion Criteria

17.2.3 Risk Acceptance

## 17.3 Post-release Activities

17.3.1 Validation and Verification

17.3.2 Independent Testing

# Part VII Software Deployment, Operations, Maintenance

# Ch18 Secure S/W Installation & Deployment

## 18.1 Secure Software Installation & Deployment

18.1.1 Installation Validation and Verification

18.1.2 Planning for Operational Use

## 18.2 Configuration Management

18.2.1 Organizing the Configuration Management Process

18.2.2 Configuration Management Roles

18.2.3 The Configuration Management Plan

18.2.4 The Configuration Management Process

# Ch19 Secure S/W Operations & Maintenance

## 19.1 Secure Software Operations

19.1.1 Operations Process Implementation

## 19.2 The Software Maintenance Process

19.2.1 Monitoring

19.2.3 Incident Management

19.2.4 Problem Management

19.2.5 Change Management

19.2.6 Backup, Recovery, Archiving

## 19.3 Secure DevOps

## 19.4 Secure Software Disposal

19.4.1 Software Disposal Planning

19.4.2 Software Disposal Execution

# Part VIII Supply Chain & Software Acquisition

# Ch20 Supply Chain & Software Acquisition

## 20.1 Supplier Risk Assessment

20.1.1 What Is Supplier Risk Assessment?

20.1.2 Risk Assessment for Code Reuse

20.1.3 Intellectual Property

20.1.4 Legal Compliance

20.1.5 Supplier Prequalification

## 20.2 Supplier Sourcing

20.2.1 Contractual Integrity Controls

20.2.2 Vendor Technical Integrity Controls for Suppliers

20.2.3 Managed Services

20.2.4 Service Level Agreements

## 20.3 Software Development & Testing

20.3.1 Code Testing

20.3.2 Security Testing Controls

20.3.3 Software Requirements Testing/ Validation

20.3.4 Software Requirements Testing/ Validation for

Subcontractors

## 20.4 Software Delivery, Operations, Maintenance

20.4.1 Chain of Custody

20.4.2 Publishing and Dissemination Controls

20.4.3 System-of-Systems Integration

20.4.4 Software Authenticity & Integrity

20.4.5 Product Deployment & Sustainment Controls

20.4.6 Monitoring and Incident Management

20.4.7 Vulnerability Management, Tracking, Resolution

## 20.5 Supplier Transitioning

1. IAM controls are security controls, which are audited annually under Sarbanes-Oxley (SOX) section 404 [↑](#footnote-ref-1)
2. Race conditions occurs when two thread operate on same object without proper synchronization and there operation interleaves on each other [↑](#footnote-ref-2)