- 1. 安全开发生命周期（SDL）
  - 1.1 SDL的核心理念
  - 1.2 SDL的核心组成
  - 1.3 SDL的挑战与重要性
  - 1.4 SDL的最佳实践
- 2. 软件安全成熟度模型
  - 2.1 Cigital的内置安全成熟度模型（BSMM）
  - 2.2 OWASP的软件保证成熟度模型（SAMM）
  - 2.3 ISO/IEC 27034：信息技术、安全技术、应用安全
  - 2.4 其他SDL最佳实践的资源
    - 2.4.1 SAFECode
    - 2.4.2 美国国土安全软件保障计划
    - 2.4.3 美国国家标准与技术研究院（NIST）
    - 2.4.4 MITRE公司公共计算机漏洞和暴露（CVE）
    - 2.4.5 SANS研究所高级网络安全风险
  - 2.5 关键工具和人才
    - 2.5.1 工具
      - 2.5.1.1 模糊测试（Fuzzing）
      - 2.5.1.2 静态分析（Static Analysis）
  - 2.5.2 人才
    - 2.5.2.1 软件安全架构师
  - 2.6 最小特权原则
    - 重要性
    - 实际应用
    - 注意事项
  - 2.7 隐私
    - 重要性
    - 关键隐私设计原则
    - 隐私保护的重要性
  - 2.8 度量标准的重要性
    - 重要性概述
    - 框架和最佳实践
    - 度量报告机制
  - 2.9 把SDL映射到软件开发生命周期（SDLC）
    - 映射的重要性
  - 2.10 软件开发方法
    - 2.10.1 瀑布开发

            - 特性
            - 假设和限制
            - 缺点
            - 转变趋势
        - 2.10.2 敏捷开发方法
            - 2.10.2.1 Scrum
                - 主要特点
                - 增值属性
            - 2.10.2.2 精益开发
                - 精益原则
                - 关键要素
- 安全评估(A1)
    - 解决的问题
    - 活动
    - PIA 主要内容
    - 成功关键因素和度量标准
- 安全开发生命周期（SDL）阶段概述
- 架构（A2）
    - 威胁建模
    - DREAD模型
    - 成功的关键因素和度量标准
- 设计和开发（A3）
    - 安全测试
    - 成功的关键因素和度量标准
- 发布（A4）
    - 策略一致性分析
    - 安全测试
    - 代码审查
    - 安全分析工具
    - 成功的关键因素和度量标准
- 成功的关键因素和度量标准 - 发布（A5）
- 发布（A5）
    - 成功的关键因素
    - 度量标准
- 对SDL的需求
    - 隐私与安全
    - 微软可信计算关注的方面
    - 当前软件开发方法的不足

- 软件安全开发生命周期（SDL）过程
- 第0阶段：教育和意识
- 第1阶段：项目启动
- 第2阶段：定义并遵从设计最佳实践
- 第3阶段：产品风险评估
- 第4阶段：风险分析
- 第5阶段：创建安全文档、工具以及客户最佳实践
- 第6阶段：安全编码策略
- 第7阶段：安全测试策略
- 第8阶段：安全推进活动
- 第9阶段：最终安全评审
  - 评审步骤
- 第10阶段：安全响应规划
  - 准备响应的原因
  - 响应规划过程
- 第11阶段：产品发布
  - 发布前确认
  - 签字通过
- 第12阶段：安全响应执行
  - 执行原则
  - 主要步骤

安全开发生命周期（SDL）及软件安全成熟度模型总结

1. 安全开发生命周期（SDL）

安全开发生命周期（SDL）是一种全面的方法，旨在将安全性嵌入到软件开发的整个过程中。它涵盖了从项目启动到产品发布以及后续维护的每一个阶段，确保在软件开发的每一步骤中都考虑到安全性。

1.1 SDL的核心理念

SDL的核心理念是在软件开发的早期阶段就开始考虑安全性，将安全需求作为软件需求的一部分，并在整个开发过程中持续进行安全评估、测试和改进。这有助于在软件开发生命周期的早期阶段就发现和修复潜在的安全漏洞，降低软件发布后的安全风险。

1.2 SDL的核心组成

- **安全需求分析**：在项目的起始阶段，就明确软件的安全需求，确保产品能够满足相关的安全标准和法规要求。
- **安全设计**：在软件设计阶段，考虑如何设计安全的软件架构，包括数据保护、访问控制、身份认证等安全措施。
- **安全编码**：采用安全的编程技术和最佳实践，避免常见的安全漏洞，如跨站脚本（XSS）、SQL注入等。
- **安全测试**：通过静态代码分析、动态测试、渗透测试等手段，对软件进行全面的安全测试，发现和修复潜在的安全漏洞。
- **安全发布和维护**：确保在软件发布和维护过程中，能够及时修复已知的安全漏洞，并持续监控和应对新的安全威胁。

1.3 SDL的挑战与重要性

随着网络安全威胁的日益严重，传统的网络安全措施已无法完全保护系统资产。攻击者可以利用软件中的安全漏洞进行攻击，导致数据泄露、系统瘫痪等严重后果。因此，将安全性嵌入到软件开发的整个过程中变得尤为重要。SDL通过在整个开发过程中实施安全控制，确保软件在设计、开发、测试和维护过程中都能抵御潜在的威胁和攻击，降低安全风险。

1.4 SDL的最佳实践

为了有效实施SDL，组织可以采取以下最佳实践：

- **建立安全文化**：在组织中树立安全优先的价值观，鼓励员工积极参与安全工作。
- **培训和教育**：提供安全培训和教育，提高开发人员的安全意识和技术能力。
- **使用安全工具和技术**：采用静态代码分析工具、自动化测试工具等安全工具和技术，提高安全工作的效率和准确性。
- **持续监控和改进**：建立安全监控机制，持续跟踪和分析安全事件，及时发现问题并进行改进。

2. 软件安全成熟度模型

软件安全成熟度模型是评估组织在软件安全方面成熟度和能力的一个框架。它可以帮助组织了解自身在软件安全方面的现状，识别存在的问题和改进的方向。

2.1 Cigital的内置安全成熟度模型（BSMM）

BSMM是一个基于真实软件安全措施的研究框架，旨在帮助组织确定其软件安全措施的有效性和如何优化工作流程。它包含12个实践类别，分为四个主要领域：策略与度量、培训与教育、威胁建模与安全设计、验证与测试。通过评估这些实践类别的实施情况，组织可以了解自身在软件安全方面的成熟度和能力，并采取相应的改进措施。

2.2 OWASP的软件保证成熟度模型（SAMM）

SAMM是一个灵活的框架，用于指导组织将安全性融入其软件开发流程中。它涵盖了软件开发的各个方面，包括管理、开发和部署。SAMM允许组织评估其当前的安全保障程序，并基于特定的风险提供改进建议的路线图。通过迭代建立安全保证程序，组织可以实现量化的安全改进，提高软件的安全性。

2.3 ISO/IEC 27034：信息技术、安全技术、应用安全

ISO/IEC 27034是一个国际标准，为组织提供了在应用安全方面的指导。它涵盖了应用安全管理的各个方面，包括安全策略、风险评估、安全控制、监控和审计等。通过遵循ISO/IEC 27034标准，组织可以确保其应用程序符合国际安全标准，降低安全风险。该标准还提供了关于如何建立和维护一个有效的应用安全管理体系的详细指导，帮助组织提高其在应用安全方面的能力和水平。

2.4 其他SDL最佳实践的资源

2.4.1 SAFECode

- 组织简介： SAFECode（Software Assurance Forum for Excellence in Code）是一个非营利性组织，专注于通过先进的软件保证方法，提高ICT（信息、通信技术）产品和服务的可信性。
- 目标： 确定和推广开发更安全可靠的软件、硬件和服务的最佳实践。
- 成果： 提供了一系列基础的安全开发实践，这些实践在不同开发环境下被SAFECode成员实现，有效改善了软件安全性。

2.4.2 美国国土安全软件保障计划

- **背景：** 自2004年以来，美国国土安全部（DHS）的软件保障计划资助了“Build Security In”（BSI）的开发。
- **原则：** 软件安全本质上是软件工程问题，需要在整个SDLC（软件开发生命周期）中以系统的方式管理。
- **目标：** 减少软件漏洞，改善可信软件产品的日常开发和部署。
- **行动：** 促使公共和私营部门合作，包括开发、发布和推广实用指南与工具。

2.4.3 美国国家标准与技术研究院（NIST）

- **软件保障度量和工具测量（SAMATE）：** 致力于开发软件工具测量方法，以提高软件保障。
- **特别出版物800-64：** 协助联邦政府机构将重要的信息技术安全步骤整合到现有的系统开发生命周期中。
- **国家漏洞数据库（NVD）：** 基于标准的漏洞管理数据使用SCAP（安全内容自动化协议）表示，实现漏洞管理、安全测量和合规的自动化。

2.4.4 MITRE公司公共计算机漏洞和暴露（CVE）

- **目标：** 提供一个被广泛认同的信息安全漏洞和暴露的清单，使独立的漏洞功能（工具库、存储库和服务）间共享信息更加容易。

2.4.5 SANS研究所高级网络安全风险

- **简介：** 之前的SANS 20个最关键的互联网安全漏洞，是互联网安全中最关键问题领域的一个共识列表。
- **内容：** 提供了纠正这些安全缺陷的步骤说明和更多信息的链接。
- **特点：** 列表包括CVE标识符，有助于系统管理员使用CVE兼容的产品和服务，使网络更安全。

2.5 关键工具和人才

2.5.1 工具

2.5.1.1 模糊测试（Fuzzing）

模糊测试（Fuzz testing/Fuzzing）是一种自动或半自动化的黑盒软件测试技术，通过向计算机软件程序输入无效、意外或随机数据来发现bug或安全缺陷。该技术已成为测试软件或计算机系统安全问题的关键要素。

- 优势：
  - 测试设计简单，与系统行为无关。
  - 适用于安全和质量保障测试。
- 常用工具：
  - Codenomicon
  - Peach Fuzzing Tool
- 重要性：
  - 模糊测试是软件安全的关键要素，必须嵌入到SDL中。
  - 是美国国防部DIACAP的要求。

2.5.1.2 静态分析（Static Analysis）

静态分析是指在不实际运行程序的条件下，对计算机软件进行分析的方法。静态分析通过自动化的软件工具进行，主要用于在开发或质量保障阶段识别漏洞。

- 优势：
  - 相对于人工分析，分析更频繁，安全知识普遍优于标准开发人员。
  - 允许安全架构师或工程师在必要时介入。
  - 提供行代码级检测，使开发团队能快速修补漏洞。
- 选择工具的考虑因素：
  - 支持的语言。
  - 可伸缩性。
  - 是否可以嵌入到开发过程中。
  - 误报率。
- 使用静态分析的好处：
  - 缩短时间：静态分析工具可以快速评审大量代码。
  - 不易疲劳：连续运行效果不变。

- 教育开发团队：通过工具和供应商资源进行软件安全教育。

2.5.2 人才

2.5.2.1 软件安全架构师

软件安全架构师负责提供整个X公司产品安全的架构和技术指导。

- **角色和职责：**
  - 推动整体软件安全架构。
  - 提供技术指导，如在全面计划、开发和X公司软件安全努力的执行中。
  - 与产品和工程开发团队紧密合作，确保产品符合或超过用户的安全和认证需求。
  - 监控安全技术的发展趋势和需求。
  - 制定并执行安全计划，包括与第三方供应商的合作开发。
  - 保证和创建安全策略、过程、实践和操作。
  - 从事实际深入的软件分析、评审和设计，包括技术评审和分析。
  - 在安全认证过程中提供主要技术角色。
  - 为员工、合约商、开发人员等提供培训。
  - 指导X公司软件开发团队通过面向SDLC的安全开发生命周期（SDL）设计。

2.6 最小特权原则

在信息安全、计算机科学等领域，最小特权原则（the principle of least privilege）规定了在计算环境特定的抽象层中，每一个模块（如过程、用户或程序）仅仅能访问合法目的所必需的信息和资源。

重要性

- **限制特权提升：**是威胁建模的重要部分，也是SDL架构（A2）阶段的核心组成部分。
- **防止敏感数据泄露：**确保用户只能访问他们所需的信息和资源。
- **软件设计关键因素：**软件设计应遵循最小特权原则。

实际应用

- **确定最小特权集合：**在SDL/SDLC的设计阶段，确定执行工作所需的最小特权集合。

- **用户和资源权限限制**：不仅限制用户权限，还包括资源权限，如CPU、内存、网络和文件系统权限。
- **软件隔离**：将软件隔离到单独组件中，减少攻击面。
- **访问权限的慎重授权**：限制攻击者成功地攻击软件或系统。

注意事项

- **资源权限的多次检查**：将权限授予一个对象之前满足多个条件，增强安全性。
- **最小权利的时间限制**：资源的最小权利和访问应限于完成该任务需要的最短时间内。

2.7 隐私

保护用户隐私是SDL过程的重要组成部分，应被视为SDLC所有阶段重要的系统设计原则。

重要性

- **信任危机**：用户隐私安全出问题会导致信任危机。
- **法律法规**：新的隐私法律法规强调了在软件和系统设计中包含隐私的重要性。
- **早期集成**：在SDLC的适当阶段融入隐私保护方法和技术，以维护个人隐私。

关键隐私设计原则

- **适当通知**：提供有关数据收集、存储或共享的适当通知。
- **用户策略和控制**：使用户可以对自己的个人信息做出明智的决策。
- **最小化数据收集和敏感性**：仅收集必要的数据，并最小化数据的敏感性。
- **存储保护和数据传送**：确保数据的存储和传输过程中得到保护。

隐私保护的重要性

- **避免法律诉讼**：忽略用户隐私问题可能导致法律诉讼。
- **媒体负面报道**：隐私泄露可能导致媒体负面报道，损害企业形象。
- **建立信任**：保护用户隐私有助于建立企业与客户之间的信任关系。

2.8 度量标准的重要性

安全度量标准是评估组织软件安全计划有效性的关键资源。它们不仅有助于改进产品安全程序的性能，证明合规性，还能提高管理和利益相关者的安全意识水平，并辅助决策者做出资金分配决策。

重要性概述

- **持续改进**：度量标准可用于不断改进产品安全性能。
- **合规性证明**：证明组织符合相关安全标准和法规。
- **提高安全意识**：提高管理和利益相关者的安全意识水平。
- **资金请求支持**：协助决策者提出资金请求，以支持安全相关项目。

框架和最佳实践

- **适当的框架**：确保有适当的框架来得出有意义的度量数据。
- **统一最佳实践**：在整个组织中统一部署最佳安全做法和措施。
- **管理支持**：要求在整个组织中强有力地执行管理支持。

度量报告机制

- **有效管理**：为了有效地衡量安全状况，必须得到有效的管理。
- **集中报告**：集中度量报告机制对于生成有意义的指标至关重要。

2.9 把SDL映射到软件开发生命周期（SDLC）

将SDL映射到当前的SDLC是确保安全内置到每个开发阶段的关键。

映射的重要性

- **内置安全**：安全内置到每个SDLC阶段可以提高软件的默认安全性。
- **降低风险**：后续软件变化不太可能危及整体安全。
- **协作**：与SDL的拥有者和利益相关者一起工作，构建高效和可实现的安全。

2.10 软件开发方法

2.10.1 瀑布开发

特性

- **传统方法**：瀑布开发是较传统的软件开发方法。
- **高风险**：通常高风险、高成本，且效率低于敏捷方法。
- **阶段性**：每个阶段结束前，下一个阶段不会开始。
- **文档驱动**：大量文档是主要的通信机制。

假设和限制

- **需求明确**：适用于需求得到充分理解或不太复杂的项目。
- **无回头路**：一旦某阶段完成，通常不会重复访问。
- **最小化变化**：通过变更控制委员会最小化变化。
- **项目经理驱动**：项目经理在项目中承担大部分责任。

缺点

- **不灵活**：对于需求频繁变化的项目不够灵活。
- **高成本**：通常成本较高，因为需要详细的规划和文档。
- **晚期反馈**：直到项目晚期才有可行软件，可能导致问题难以修复。

转变趋势

- **行业转向敏捷**：由于瀑布方法的上述缺点，行业正在转向更加灵活和响应式的敏捷开发方法。

2.10.2 敏捷开发方法

2.10.2.1 Scrum

Scrum 是一种迭代和增量的敏捷软件开发方法，用于管理软件项目、产品或应用程序开发。

主要特点

- **迭代和增量**：通过一系列的冲刺（Sprint）来迭代开发产品。
- **经验方法**：接受需求不能完全被理解和定义，注重提高团队的交付能力和响应新需求。
- **自组织团队**：协同定位、多学科团队自我组织以达成目标。
- **可控时间**：每个冲刺时间可控，保证产品按时完成。
- **产品积压**：从产品积压中选取优先级最高的需求进行开发。
- **展示软件**：每个冲刺结束后，团队会演示完成的软件。

增值属性

- **自适应计划**：根据反馈调整计划。
- **早期反馈**：通过早期可行软件获取客户反馈。
- **最大化变化**：关注学习，接受好的变化。
- **团队责任**：小团队自主计划并重新计划自己的工作。
- **优化价值**：优化业务价值、上市时间和质量。

2.10.2.2 精益开发

精益开发是一种关注于避免浪费和最大化价值的开发方法。

精益原则

1. **避免浪费**：消除任何非增值活动。
2. **增强学习**：从每个迭代中学习并改进。
3. **尽量推迟决策**：直到需要时才做决策。
4. **尽快交付**：尽早向客户提供价值。
5. **授权团队**：让团队自主决策和行动。
6. **嵌入完整性**：确保产品质量和完整性。
7. **着眼整体**：从整个系统的角度考虑。

关键要素

- **着眼整体的模型**：即使在分散的团队中也能保持一致性。
- **相关学科**：精益软件开发方法已演变成一门相关学科，而非特定敏捷方法。

安全评估(A1)

安全评估是安全开发生命周期（SDL）的第一个阶段。

解决的问题

- 软件满足客户任务和关键程度如何？
- 软件所需的安全目标是什么？
- 哪些法规和政策适用于确定需要保护的内容？
- 软件运行环境可能存在哪些威胁？

活动

- 软件安全团队早期参与项目并主持发现会议。
- 创建SDL项目计划。
- 启动隐私影响评估（PIA）。

PIA 主要内容

- 立法摘要
- 所需的工艺步骤
- 技术和技巧
- 其他资源

成功关键因素和度量标准

- **早期参与**：安全团队在项目早期介入。
- **明确的计划**：制定详细的SDL项目计划。
- **隐私评估**：完成PIA并识别关键隐私风险。
- **度量标准**：定义并跟踪安全评估阶段的度量标准，如威胁数量、风险评估准确性等。

安全开发生命周期（SDL）阶段概述

架构（A2）

在安全开发生命周期的第二阶段，安全方面的考虑被带入软件开发生命周期，确保威胁、要求、功能和集成方面的潜在制约因素均被考虑到。

威胁建模

威胁建模是识别、量化并排序潜在威胁的过程。其通常包括五个步骤：

1. 确定安全目标
2. 调查应用程序

3. 分解应用程序
4. 识别威胁
5. 确定漏洞

DREAD模型

DREAD模型是一个风险评估框架，用于建立风险评级。它包括以下五个类别：

- 潜在损害（Damage）
- 可重复性（Reproducibility）
- 可利用性（Exploitability）
- 受影响的用户（Affected Users）
- 可发现性（Discoverability）

每个类别都用1-10的数字进行评分，数字越高风险越严重。

成功的关键因素和度量标准

- 业务微威胁、技术威胁和威胁参与者的列表
- 不满足的安全目标个数
- 遵守公司政策的百分比
- 软件入口点的数量
- 风险接受、减轻和容忍的百分比
- 重新定义的初始软件需求百分比
- 软件体系结构的变化数量
- 根据安全要求所需的软件体系结构更改数量

设计和开发（A3）

在这个阶段，安全被考虑进软件的设计和开发中，以消除潜在的安全和隐私问题。

安全测试

安全测试的通用步骤包括：

- 定义测试脚本

- 定义用户社区
- 识别项目障碍物
- 识别内部资源
- 识别外部资源

测试技术主要分为白盒、灰盒或黑盒。

- 白盒：从内部的角度测试软件，如源代码分析。
- 灰盒：分析源代码的同时使用黑盒技术。
- 黑盒：从外部的角度测试软件，如模糊测试。

成功的关键因素和度量标准

（未具体提供，但可能包括测试用例的覆盖率、发现的安全问题数量等）

发布（A4）

在发布阶段，软件经过各种测试以确保其质量和安全性。

策略一致性分析

A4策略一致性分析是A3策略依赖检查的延续，确保所有外部策略都得到遵守。

安全测试

安全测试在发布阶段继续进行，包括单元测试、性能测试、系统测试和回归测试。

代码审查

代码审查是发现软件安全缺陷的有效方法，包括自动扫描、人工渗透测试、静态分析和人工代码审查。

安全分析工具

- 静态分析（SATA）：在不执行代码的情况下进行的安全分析。

- 动态分析（DAST）：通过运行程序进行的安全分析。
- 模糊测试：提供非法或随机数据以发现漏洞。
- 人工代码审查：逐行检查代码以发现安全漏洞。

成功的关键因素和度量标准

（未具体提供，但可能包括安全测试的通过率、代码审查发现的问题数量等）

以上内容概述了安全开发生命周期中关键阶段的一些核心活动和度量标准。在实际操作中，每个组织可能会根据自己的需求和情况调整这些阶段和度量标准。

成功的关键因素和度量标准 - 发布（A5）

发布（A5）

在软件开发生命周期的最后阶段，发布阶段（A5），需要确保软件是安全的，并且隐私问题已经被解决到软件发布时可以接受的程度。

成功的关键因素

- **漏洞扫描和渗透测试**：确保通过漏洞扫描和渗透测试来识别潜在的安全问题。
- **开源软件管理**：遵循开源软件的许可证要求，并确保其满足内部安全标准。
- **最终安全性审查**：对所有安全活动进行重新评估，确保软件产品为发布做好准备。
- **最终隐私性审查**：确保隐私需求得到满足，符合监管部门、公司策略和客户期望。

度量标准

- **遵守公司策略的百分比**：衡量公司策略在发布阶段被遵守的程度。
- **安全漏洞扫描和渗透测试**：
  - **发现的安全问题的数量、类型和严重性**：量化漏洞扫描和渗透测试的结果。
  - **发现的安全问题的修复数量**：跟踪并报告已修复的安全问题数量。
  - **发现的严重问题的数量、类型、严重性**：特别关注严重问题的处理情况。
- **遵守安全和隐私需求的百分比**：衡量发布的软件产品在多大程度上满足了安全和隐私需求。

对SDL的需求

隐私与安全

- **隐私**：遵守策略的一种方式，核心是符合监管部门的要求、公司策略和客户期望。
- **安全**：执行策略的方式，需考虑服务水平协议（SLA）和维护时间。
- **可靠性**：虽然与安全有显著差异，但有时会产生冲突。

微软可信计算关注的方面

- 安全
- 隐私
- 可靠性

当前软件开发方法的不足

1. **“只要给予足够的关注，所有的缺陷都将无处遁形”**：此方法失败的原因在于人们通常不喜欢审核代码中的bug和漏洞，因为过程缺乏创新性且枯燥。
2. **专利软件开发模式**：文档中很少提及“安全”和“质量”，缺乏针对安全性的明确指导。
3. **敏捷软件开发模式**：安全最佳实践不够深入，可能导致安全问题的遗漏。
4. **通用评估准则（CC）**：虽然提供了安全特性执行的证据，但其目标并非确保软件没有实现上的安全bug。

总结：在软件开发生命周期的发布阶段（A5），成功的关键在于通过全面的安全测试和审查来确保软件的安全性和隐私保护。同时，当前的软件开发方法存在不足，需要在SDL中充分考虑这些因素，以确保软件的质量和安全性。

软件安全开发生命周期（SDL）过程

第0阶段：教育和意识

- **bug bush**：娱乐性高，奠定微软安全教育的根基。
- **安全意识演讲**：包含可信计算概述、SDL简介、安全设计基础等。

- **年度安全培训**：所有工程人员必须参加，也可进行在线培训。
- **成功要素**：
  - 管理层明确支持
  - 富有经验演讲者
  - 持续进行的教育

第1阶段：项目启动

- **SDL覆盖判断**：确定SDL是否适用于该项目。
- **安全人员指定**：指定安全顾问，负责安全活动指引。
- **SDL启动会议**：评审设计与威胁模型，分析并分类bug。
- **安全领导团队**：确保安全与隐私类bug被包含在bug跟踪管理中。
- **建立“bug标准”**：统一bug分类和管理标准。

第2阶段：定义并遵从设计最佳实践

- **安全设计原则**：
  - 经济机制
  - 默认失效保护
  - 完全中介
  - 公开设计
  - 权限分离
  - 最小特权
  - 最少公共机制
  - 心理可接受程度
- **受攻击面分析（ASR）**：理解并降低应用受攻击面。

第3阶段：产品风险评估

- **安全风险评估**：判断系统中易受攻击的漏洞级别。
- **隐私影响分级**：定义三个策略值，根据应用行为确定隐私分级。
- **统一因素**：确保资源分配与风险级别相匹配。

第4阶段：风险分析

- **威胁建模**：描述应用背景信息，定义高层应用模型。
- **威胁建模过程**：
  - 定义应用场景
  - 收集外部依赖列表
  - 定义安全假设
  - 创建外部安全备注
  - 绘制数据流图
  - 确定威胁类型
  - 识别系统威胁
  - 判断风险
  - 规划消减措施

第5阶段：创建安全文档、工具以及客户最佳实践

- **指导性安全最佳实践文档**：包括安装、使用、帮助、开发人员文档等。

第6阶段：安全编码策略

- **编译器内置防御特性**：
  - 缓冲区安全检查（/GS）
  - 安全异常处理（/SAFESEH）
  - 数据执行防护兼容性（/NXCOMPAT）
- **源代码分析工具**：辅助识别潜在的安全问题。
- **安全编码检查清单**：确保编码遵循安全最佳实践。

第7阶段：安全测试策略

- **模糊测试**：发现可靠性bug，也可发现安全bug。
- **渗透测试**：发现信息系统中的脆弱性。
- **运行时验证**：通过运行时数据监控发现潜在的安全问题。

第8阶段：安全推进活动

- **周密计划**：确保活动按计划进行。
- **专门培训**：使软件开发团队了解安全推进活动的期望和流程。
- **代码评审**：对所有代码进行评审，无论年限。
- **威胁模型评审**：架构师和程序经理需多次评审威胁模型。
- **重估受攻击面**：推动好的安全习惯和代码评审任务的优先程度评级。

第9阶段：最终安全评审

评审步骤

1. **产品团队协调**：
    - 非纯技术性活动。
    - 团队需填写安全评审调查问卷。
2. **威胁模型再次评审**：
    - 验证威胁模型的有效性和完整性。
    - 识别任何潜在的新威胁或遗漏的威胁。
3. **未修复的安全bug评审**：
    - 评估标记为“暂不修复”的安全bug的风险。
    - 验证是否真正可以不予理睬或需要采取其他措施。
4. **工具有效性验证**：
    - 确保使用的安全工具（如静态分析工具、动态分析工具等）是有效的。
    - 验证工具是否发现了已知的安全问题。

第10阶段：安全响应规划

准备响应的原因

- 开发团队会出错。
- 新漏洞会不断出现。
- 安全规则会发生变化。

响应规划过程

1. **建立安全响应过程**：

    - 定义如何响应新的安全漏洞。
    - 确保过程包括漏洞验证、风险评估、通知、修复和发布等环节。
2. 组建安全响应中心：
    - 由产品开发团队负责。
    - 专门负责处理安全漏洞的接收、分析、响应和跟踪。

第11阶段：产品发布

发布前确认

- 安全与隐私团队必须认可SDL过程已被正确无误地贯彻落实。
- 确保所有已知的安全问题已得到妥善处理。

签字通过

- 只有在所有必要的安全和隐私标准都得到满足后，软件才能被签字通过并发布。

第12阶段：安全响应执行

执行原则

1. 准备时间：
    - 安全响应的准备时间应在漏洞被上报之前。
2. 全员准备：
    - 每个软件团队都必须做好安全响应准备。

主要步骤

1. 组建响应团队：
    - 快速组建由开发、测试、支持等成员组成的响应团队。
2. 全线产品支持：
    - 确保所有产品都受到支持，并准备好修复方案。
3. 客户支持：
    - 提供对客户的安全支持和通知。

4. 产品更新能力：
    - 确保产品具有快速发布安全更新的能力。
5. 漏洞主动发现：
    - 在安全研究人员发现漏洞之前，通过内部测试和审计主动发现漏洞。