DevSecOps的六大支柱：务实的实现

CSA GCR cloud security alliance GREATER CHINA REGION ®

CSA cloud security alliance®

联盟简介

云安全联盟（Cloud Security Alliance,CSA）是中立、权威的全球性非营利产业组织，于2009年正式成立，致力于定义和提高业界对云计算和下一代数字技术安全最佳实践的认识，推动数字安全产业全面发展。

云安全联盟大中华区（Cloud Security Alliance Greater China Region, CSA GCR）作为CSA全球四大区之一，2016年在香港独立注册，于2021年在中国登记注册，是网络安全领域首家在中国境内注册备案的国际NGO，旨在立足中国，连接全球，推动大中华区数字安全技术标准与产业的发展及国际合作。

我们的工作

联盟会刊下载地址
了解联盟更多信息

加入我们

CSA大中华区官网
(https://c-csa.cn)

点击会员

加入联盟

填写相关申请信息

成为CSA会员

JOIN US

致谢

《DevSecOps 的六大支柱：务实的实现（The Six Pillars of DevSecOps:Pragmatic Implementation）》由 Roupe Sahans 编写，并由 CSA 大中华区专家组织翻译并审校。（以下排名不分先后）：

中文版翻译专家组

翻译组组长：

李岩

翻译组成员：

何伊圣　陈宏伟　殷铭　贺志生　吴嘉雯　高亚楠

伏伟任　江澎　江泽鑫　余晓光　谢绍志　屈伟

江楠　王岩　王贵宗　王彪　苏泰泉　欧建军

林艺芳　张坤

审校组成员：

李岩　江泽鑫　王彪　张坤

研究协调员：闭俊林、赵鹏

贡献单位：

北京天融信网络安全技术有限公司　中国电信股份有限公司研究院

华为技术有限公司　上海缔安科技股份有限公司

英文版本编写专家

主要作者：

Roupe Sahans

贡献者：

Charles Bideau　Aristide Bouix　Mauricio Cano　Eric Gauthier

Daniel Gora　Michael Holden　Brynna Nery　Abdul Rahman Sattar

Michael Roza　Sreeni Sharma　Damian Zawodnik

审校者：

Chris Latham　Fabiola Moyón　Douglas Needham

联合主席：

Kapil Bareja　Chris Kirschke　Sam Sehgal

CSA 分析师：

Josh Buker

CSA 全球员工：

Claire Lehnert　Stephen Lumpe

在此感谢以上专家。如译文有不妥当之处，敬请读者联系 CSA GCR 秘书处给予改正！

联系邮箱 research@c-csa.cn；云安全联盟 CSA 公众号。

序言

DevSecOps 随着 DevOps 的发展进入到了火爆的程度，DevSecOps 也为云原生流水线注入了新的安全活力。云原生技术以其高度的弹性、自动化特性，为 DevSecOps 的实施提供了更为广阔的空间。

在云原生环境下，DevSecOps 能够充分利用云计算的优势，实现更加灵活、高效的安全管理和软件开发。云原生技术强化了 DevSecOps 的安全能力。在云原生环境中，安全策略可以被更好地集成到整个开发流程中，实现安全前置和持续集成。通过利用云原生平台的安全特性，如加密、访问控制、安全审计等，DevSecOps 能够更有效地保护应用和数据的安全。

通过学习 CSA 务实的 DevSecOps 实施，帮助企业提升黄金流水线的合规和安全能力，实现基于快速交付的安全合规的新方案。

DevSecOps 专题也是 CSA 顶级云安全专家课程(CSA ACSE)中级课程，DevSecOps 是践行安全黄金流水线的实现，从一开始就考虑应用程序和基础设施安全。DevSecOps 专家认证（Certified DevSecOps Professional）是 ACSE 置换认证的前置课程之一。CSA DevSecOps 课程的价值主要体现在提升安全性、促进团队协作、加速软件交付、培养全栈人才和提高企业价值等方面。

李雨航 Yale Li

CSA 大中华区主席兼研究院院长

目 录

前言

云安全联盟（CSA）和 SAFECode 致力于提高软件安全成果。2019 年 8 月发布的论文《DevSecOps 的六大支柱》提供了一套高级方法并成功实施了解决方案，其作者使用这些方法快速构建软件并最大限度地减少与安全相关的错误。这六大支柱是：

- 支柱 1：集体责任（发布于 2020 年 2 月 20 日）
- 支柱 2：协作与整合（发布于 2024 年 2 月 21 日）
- **支柱 3：务实的实现（发布于 2022 年 12 月 14 日）**
- 支柱 4：桥接合规与发展（发布于 2022 年 8 月 2 日）
- 支柱 5：自动化（发布于 2020 年 6 月 7 日）
- 支柱 6：测量、监控、报告和行动（预计 2024 年 4 月）

支持每个支柱的成功解决方案是云安全联盟和 SAFECode 联合出版物的更详细的集合的主题。本文是后续出版物中的第四篇。

介绍

在将安全性纳入到软件开发生命周期 (SDLC) 中时，组织有多种工具和解决方案可供选择。然而，这些通常存在一些问题：要么难以部署、实施和扩展，要么无法提供有助于减轻实际安全风险的可行见解。由于每个 SDLC 在结构、流程、工具和整体成熟度方面都不同，因此没有通用的二元蓝图来实施 DevSecOps。

通过采用与框架无关的 DevSecOps 模型（专注于应用程序开发和平台安全，以确保数字社会的安全、隐私和信任），组织将能够切实地处理 DevOps 中的安全问题。该模型通过将安全嵌入到软件开发生命周期中，以满足所有在利益相关人（包括开发人员、运维人员和安全人员）连接过程中未被满足的需求。[1]

本文中的 DevSecOps 实施指南被组织成一系列实际职责和活动，旨在帮助数字安全领导者在开始进行 DevSecOps 时做出务实决策。

采用通常由软件开发和平台工程团队有机地建立，或者由领导层统一建立。无论驱动因素和利益相关者如何，组织都应将其 DevSecOps 实施视为一项迭代的持续改进工作，而不是一次性的瀑布项目。

本文的范围确定并扩展了成功的 DevSecOps 倡议的 4 个关键要素：文化、人员、流程和技术。本文涉及隐私，但没有提供该领域的完整视图（即隐私融入安全设计）。

目标

云安全联盟 DevSecOps 工作组 (WG) 在“通过反思性安全（Reflexive Security)）进行信息安全管理：安全、开发和运维集成的六大支柱”中发布了高级指南。[2] 这六大支柱被认为是实施 DevSecOps 的关键重点领域，建议的支柱之一是务实实施安全计划。务实的定义是基于实际而非理论考虑，明智而现实地做出深思熟虑的决策。[3]

对于阅读本文的数字和安全领导者来说，DevSecOps 的实用主义将通过实施产生最高投资回报的方法来推动。本指南将帮助组织成功实施 DevSecOps，将安全性嵌入到 SDLC 和 DevOps 的现有工作流程中。

读者

本文档的目标受众包括参与风险、信息安全和信息技术管理和运维职能的人员。该受众包括 CISO、CIO、CTO 以及那些领导数字化转型计划的人。

此外，应用系统、平台、安全工程师和架构师也可以使用本指南作为改进组织 DevSecOps 计划的参考。

1 概述

本文中的实施指南侧重于涵盖内部开发和第三方打包软件的安全性，无论其在何处运行（本地或云上）。

关于组织如何实施 DevSecOps 计划，通常有两种不同的观点:

1. 从DevOps的角度来看，组织应该构建安全性并将其集成到SDLC中。
2. 从安全团队的角度来看，安全控制通常是在考虑流程和技术的基础上实施的。

我们在本文中融合了这两个常见观点，并提供了全面覆盖的安全活动指导。组织应将 DevSecOps 作为一项迭代和持续改进工作，而不是一次性项目。目的是让读者思考组织内希望重点改进的领域，然后参考本文及其指南，确定每个软件生命周期阶段的优先活动。

使用 DevOps 导入安全性

从 DevOps 的角度来看，团队可以在设计和编码阶段的早期解决安全问题，也称为“左移”。其目的是尽早将安全与组织中的非安全利益相关者集成。建立默认的安全流程，以便以不安全的方式做事需要额外的努力，并且包括异常工作流程。图 1 展示了 DevOps 观点的安全性。通过 DevOps 观点导入安全的组织通常包括以下步骤:

1. 评审开发和 IT 组织当前的软件生命周期。这就是评估、构建、测试、部署和操作新技术的方式。 SDLC 通常可以采用多种方式来处理安全团队可以利用的变更（即主要是功能请求）。
2. 评审DevOps和IT组织的现有工具，特别是自动化部署和开发任务的工具。

理解开发工作流程和现有工具集有助于在软件生命周期的每个阶段制

定安全控制决策。本文详细探讨了如何实现上述两点，并结合支柱 5 自动化中详细介绍的自动化方法。

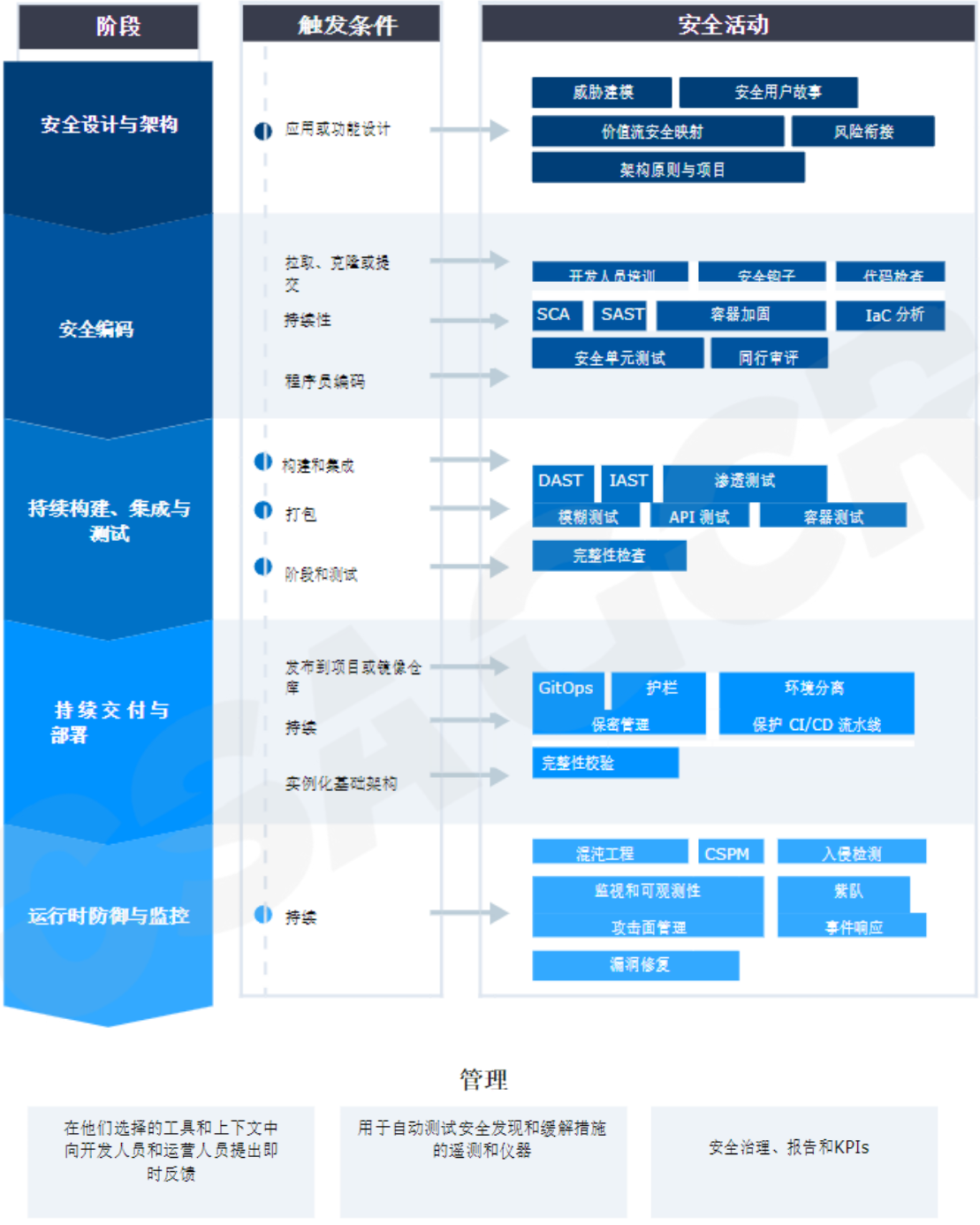

图 1：安全开发生命周期

虽然定义软件开发生命周期 (SDLC) 及其阶段的方法有很多种，但**图 1** 中引用的五个阶段的 SDLC：设计、编码、集成和测试、部署和监控为软件开发提供了通用视图。

2 软件生命周期的安全转型

从安全的角度看，寻求安全改进的组织倾向稳健的安全计划，且这些计划赖于将安全最佳实践有效集成到组织的软件开发范式中。这通常由文化和流程、安全基础设施和技术的运营以及员工意识所主导。每个软件生命周期阶段的安全都需要结合人员、文化、流程和技术:

- **人**：组织中的个人及其角色，以及安全和成功的应用程序、平台和基础设施的交付。重点关注领导人、技术利益相关者、实施者和决策者。

- **文化**：组织如何处理和管理工作和风险。文化通常是软件开发工作和安全的软性和难以衡量的领域，它直接参与了绩效和成功。

- **过程**： 常见的工作流程、手工操作和有据可查的活动。人们在自动化或两者组合的支持下执行流程。

- **技术**： 技术侧重于使用工具和自动化安全地创建和操作基础设施和应用程序，以帮助保护资产、检测漏洞、事件和事故。

这些组合之间将根据组织安全计划的规模和复杂程度而有所不同。小型组织通常主要依赖于担任多个角色的人员并管理应用程序的安全，而大型组织可能会利用技术和自动化。一个组织的文化可能会受到合规要求的影响，这会导致重大的监督和审计，而其他组织则将制衡嵌入到标准工作流程中。

2.1 人是成功实现 DevSecOps 转型的关键推动者

DevSecOps 最重要的方面是人。如果不了解如何利用现有员工的技能和知识并建立成功的跨职能团队，您的 DevSecOps 工作将变得昂贵且低效。跨职能团队和角色通常是组织在 DevSecOps 中的最大推动因素。跨职能团队是具有互补能力的混合人员群体，可降低孤岛运营的风险，并避免繁重的多阶段审批的开销。由于人是一个关键的依赖关系，我们在表 1 中确定了在 DevOps 中要考虑的不同角色配置文件以及安全性的实际应用。我们将每个角色映射到其在 DevSecOps 阶段中通常适合的位置。图 2 中显示了如何实现这一目标，每个 DevSecOps 阶段中阐述的 DevOps 和安全角色与本文中建议的安全活动保持一致。

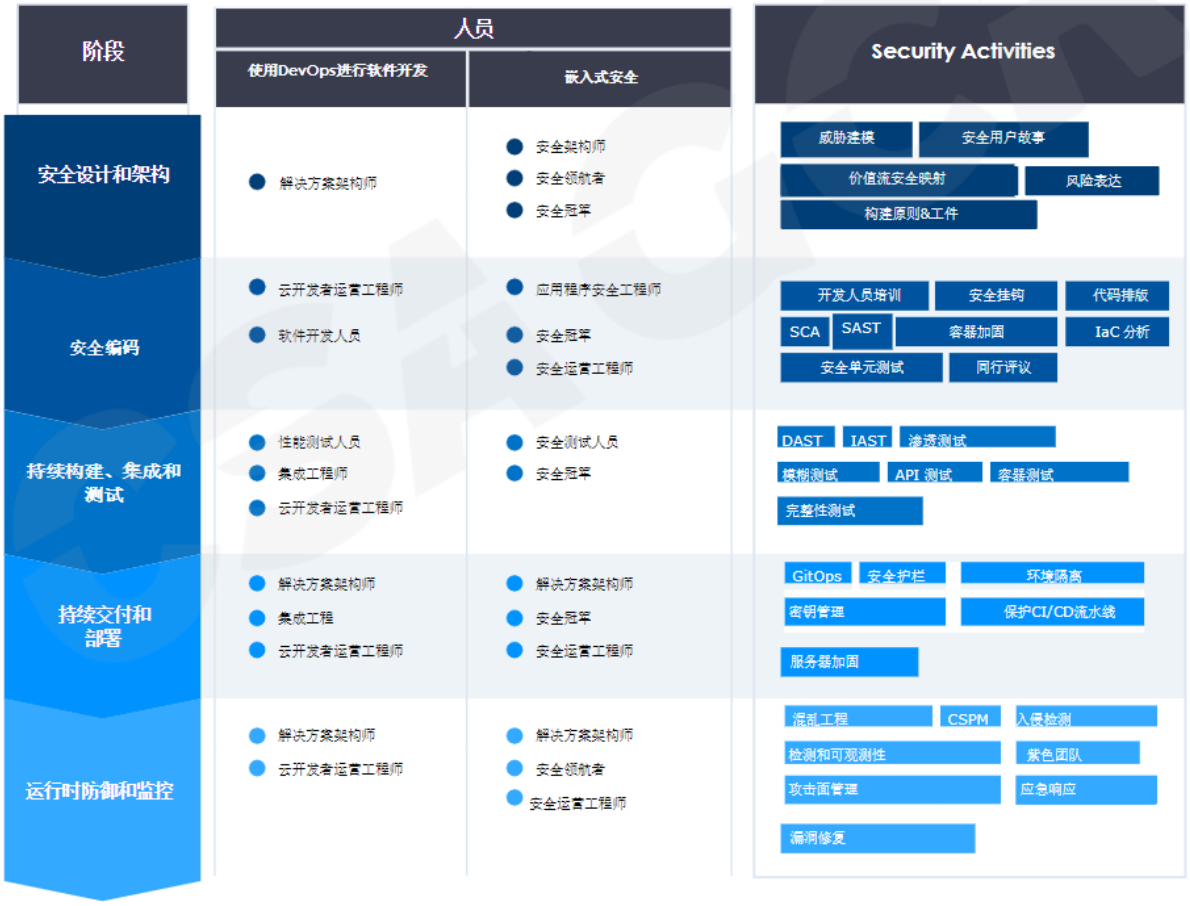

图 2：DevSecOps 结构实例

表 1：DevSecOps 角色

角色定义	
安全构架师	与业务部门合作，讨论如何设计应用程序和使用基础设施服务（包括云），以安全的方式达成于业务目标。制定方法、框架和模式，并帮助开发人员遵循安全开发最佳实践。识别未来安全方法以演进现有的实践，应对行业、监管和技术变革，并确定DevSecOps的战略和运营适应性（例如，控制措施的左移或右移）。与开发人员一起和高级业务利益相关者合作，为部署流水线和仓库的安全方式、方法和使用自动化工具提供建议和评估。
安全领航者	领导策略来沟通关键的计划和成功案例，并评估安全工具和方法在整个组织、各个团队和业务领域中的采用影响。将安全计划与业务价值和战略目标保持一致。将实现安全和业务目标的高级数字战略与低级开发人员目标连接起来。协助管理和修复对应用程序造成的安全风险、漏洞和威胁。
安全倡导者	帮助产品团队和开发人员采用与组织一致的安全计划，通常是已经从事软件或基础设施的开发人员或管理人员。持续评审其产品团队内针对威胁、漏洞和风险的安全状况。它可以作为混合角色共享。
应用程序安全工程师	识别应用程序代码中的潜在缺陷，并通过修复安全问题来缓解漏洞。应用安全编码和测试标准，并记录安全编码指南。将安全测试工具（如静态和动态应用程序测试工具）集成进源代码仓库和持续集成和交付的流水线中。对应用程序代码应用安全控制（如加密、标识和身份验证），以减少攻击面并最大程度地降低被利用风险。
安全测试人员	解决软件测试阶段的安全问题，包括风险接受、安全验收标准和安全测试方法。定期执行手动渗透测试，以暴露应用程序和基础设施的弱点，并确认安全强制执行功能的正确实施。根据测试输出提出修复建议。
安全运营工程师	提供有关云、基础设施和平台服务的安全状况的建议。在基础设施和平台上应用系统加固实践，并确保强大的安全机制。执行漏洞扫描和修复，以减少基础设施上的攻击面。

云开发者运营工程师	配置和运营云基础设施，包括服务器、存储和网络。使用持续集成和部署方法自动配置云基础设施和应用程序
解决方案架构师	开发解决方案架构和相关系统组件以满足用户需求。设计系统和系统间的接口，并确定目标环境的影响，包括但不限于安全状况。
软件开发人员	与架构师一起识别、设计和编写代码实现软件特性。他们将帮助测试、维护和更新产品，以确保所有安全、性能和功能问题得到解决。安全角色和成果使开发人员能够根据软件工程方法论、工具、实践以及安全代码指南安全地编写代码。
性能测试人员	负责编写和执行负载和压力测试基础设施、平台和应用程序的测试计划。
集成工程师	有助于确保安全活动应用于集成的软件包和集成的应用程序特性。引入影响整个应用程序的整体应用程序安全控制和过程-利用工具在运行时进行完整性校验并扫描应用程序。

现代云原生架构更加分散 - 规模较小、具有安全性的分布式团队可以更好地将端到端安全设计作为一个价值流来处理。尽管没有一种适用于每个组织的 DevSecOps 团队拓扑结构和架构的万能解决方案，但某些特征会影响效率。一个具有强大技术领导力和技术管理能力的组织将能够理解开发、运维和安全之间的差异和平衡，并认识到 DevSecOps 活动已集成到 DevOps 的每个阶段。在设计团队结构时，识别并避免一些常见错误（反模式）非常重要，例如:

- **组织孤岛**：利益相关者属于他们的团队，并且不了解彼此的工作，团队间不存在共同责任。安全在孤岛中运营，仅在部署之前作为一个单一的安全大门参与其中，这助长了一种低生产力和低效的软件开发实践。这可以从领导层决定要实施DevSecOps，并开始组建团队。这种方法成本效率低，只有在大量投资后才会开始获得投资回报，这在开发和运维团队分开并在孤岛中运营时很常见。

- **由开发人员执行DevSecOps：** 尽管通常有好的积极性，并且可以带来初步的收益。但在规模大的情况下很难保持一致，并且安全活动缺乏完整性。开发人员可能遇到的一个常见的陷阱是假设安全团队仅执行合规性和治理活动很容易低估安全构建系统所需的复杂性和知识，因为所需的技能不一定与功能开发重叠。

- **重新定义安全团队：** 从简单地招聘 DevSecOps 员工加入您的安全团队开始，以改进实践并降低成本，但他们看不到 DevSecOps 对业务的核心驱动因素。随着 DevSecOps 在行业中的发展日益明显，团队可能会在没有了解安全如何提高运营效率的情况下采纳 DevSecOps，甚至在某些情况下增加发布频率并降低安全投入。

根据现有的 DevOps 和软件开发范式了解 DevSecOps（设计、编码、集成和测试、部署和监控）的不同阶段，将有助于避免团队拓扑反模式。

2.2文化将加快速度并提高绩效

改变产品开发的思维方式是文化的基础，需要流程来确保容易采用正确的文化特征。当我们分解 DevSecOps 成功的特征时，必须在每个领域之间建立责任共担模型。开发、安全和运维角色应该协调工作，以实现快速、安全和稳定的软件目标。

商业谚语“文化在战略早餐上吃掉”[5]在将 DevSecOps 引入软件开发范式和数字化转型时具有相关性。最具创新性和普及性的软件产品通常可以不单纯依赖于大量投资和明确的战略的速度开发新功能。 DevOps 研究和评估 (DORA) 团队的研究表明[6]，重视学习的组织文化有助于通过以下方式提高软件交付绩效:

- 增加部署频率
- 缩短变更准备时间、恢复服务的时间以及变更失败率
- 强大的团队文化

高度信任并强调信息流的组织文化可以预测软件交付绩效和组织技术绩效。 DORA 研究表明，改变人们的工作方式会改变文化。团队可以通过检查 Westrum 组织文化模型[7]的六个方面，重点关注生成性文化中出现的行为，来识别有助于创建促进信息流和信任的生成性文化的有益实践:

- 高度合作
- 信使经过培训
- 风险共担
- 鼓励跨接（协作）
- 失败导致的调查
- 实施新颖性（创新）

文化指的是支持 DevSecOps 活动的原则和基础能力。这些关键实践推动了一种持续改进和作为跨职能团队的嵌入成员尽早参与安全的文化。文化可能是一个任意的概念，导致团队难以量化和衡量；我们已经认识到成功的 DevSecOps 团队的文化特征，他们可以快速将安全措施迅速。

应于满足业务需求的过程中。**图 3** 概述了这些文化特征，并用**图 1** 中 CSA DevSecOps 软件生命周期应用相应特征的程度进行着色。

文化特性	设计	编码	集成测试	交付部署	运行
加速变革	🟢	🟢	🟡	🟡	🟡
创建指南	🟢	🟡	🟡	🟡	🟢
保持沟通/透明度	🟢	🟢	🟠	🟠	🟢
安全故障准备	🟢	🟡	🟢	🟢	🟢
安全基线	🟢	🟢	🟢	🟢	🟢
继续安全学习	🟠	🟢	🟢	🟠	🟢
集体安全责任	🟢	🟢	🟢	🟡	🟠
创新	🟡	🟢	🟠	🟠	🟠
衡量工作成本	🟠	🟢	🟡	🟡	🟡
持续改进	🟢	🟢	🟢	🟢	🟢
从小处开始 从小处思考	🟢	🟢	🟢	🟢	🟡
激励	🟢	🟢	🟢	🟢	🟢

🟢 高适用性 🟡 中适用性 🟠 低适用性

图 3：DevSecOps 文化适用性矩阵

当我们思考 DevSecOps 中可衡量的文化特征时，我们还需要剖析每个特征在每个阶段通常应用的位置。只有三个文化特征适用于每个 DevSecOps 阶段。这些特征是在**表 2** 和**图 3** 中引用的安全基线和持续改进；它强调了以下方面的重要性和优先顺序:

a. 安全基线 – 了解每个阶段的最小可行安全产品（安全基线）；

b. 持续改进 – 始终在每个阶段寻求改进，无论是运营阶段还是战略阶段；

c. 激励——建立对目标和目标的奖励，以激励高绩效。

将特征映射到 DevSecOps 阶段的相关性有助于确定安全控制的适用性以及它们是否会对软件开发产生反作用。例如，“创新”特征仅在编码阶段具有较高的适用性。因此，编码阶段过度限制的安全控制可能会扼杀创新工作。然而，这应该与编码中与安全相关的文化特征相平衡：安全基线、持续安全学习和集体安全责任。

表 2 确定了优化且富有成效的文化的特征，以实现 DevSecOps 的高投资回报。每个特征都通过适用性映射到 DevSecOps 阶段；设计、编码、集成和测试、部署和监控。

表 2:DevSecOps 文化特征

定义	
加速变革	鼓励和加速持续的变革和改进。变革活动得到沟通和理解，而安全性也希望促进变革的加速
创建指南	安全设计流程和技术的使用都有详细记录，以便产品利益相关者了解安全设计工作并减少对员工个人知识的依赖。指南需要不断审视以提高效率。
保持沟通/透明度	传达关键举措和成功经验。将安全计划与业务价值和战略目标保持一致。将安全和业务目标的高级数字策略与低级开发人员目标联系起来。建设性的代码评审是常态。安全性受到重视并包含在这些评审中
安全故障准备	安全措施需要假定为会失败或被攻破的。失败是预料之中的，响应计划也已广泛制定、讨论、执行和理解。如果确实发生故障，则会对其进行评审以确定进一步的改进。
安全基线	安全需求的基线和最小可行的安全产品是被充分理解的的。不可接受的结果是众所周知的。鼓励安全且可重用的组件并将其打包为模板。
持续安全学习	有关安全的信息共享在整个组织中是持续且普遍的。共享良好的安全实践。
集体安全责任	每个人都对开发安全的产品和功能负有集体责任。安全不是别人的问题。
创新	日常运营和未来产品改进之间的工作得到了很好的平衡。时间被分配给项目目标之外的创新工作。环境旨在促进创新。
衡量工作成本	将安全左移不会让开发人员负担过重，从而对速度和创新产生负面影响。安全工作和努力是经过衡量的。
连续提高	通过增量或突破性改进对产品、服务或流程进行持续改进。
从小处开始思考	从小处开始，同时仍能实现其他特性。短期胜利/冲刺与长期构建和活动之间的区别是可以理解的。最低限度可行、安全的产品很好了解安全性是如何随着产品功能逐步发展的。
激励	是系统地建立奖励以激励参与者实现目标。当减少漏洞并获得奖励时，安全性就会得到激励。

2.3技术和流程是 DevSecOps 实务的基础

在 DevSecOps 中产生最高投资回报的活动是通过有效使用工具/技术和流程来支持的。当我们分解 DevOps 范式时，我们可以确定安全活动（技术和流程）适合的位置。**图 1** 中引用的 SDLC 的五个阶段（设计、编码、构建、部署和监控）每个阶段都有安全风险和活动，这些风险和活动在**图 4 - 8** 矩阵中进行了扩展。

本节中的安全活动并不反映解决方案设计中包含的安全资源。就像 Web 应用程序防火墙 (WAF)、防病毒 (AV) 或入侵检测系统 (IDS) 如何更好地保护应用程序/基础设施一样，它是产品设计所独有的，这就是为什么我们将其视为一种安全控制而不是工具或流程支持的安全活动。

3 DevSecOps 阶段

3.1设计与架构

安全设计和架构阶段先于编写任何代码，但在实施安全性时可能是 SDLC 中最关键的步骤。从长远来看，现阶段不安全的实施细节可能会放大未来的安全风险。相反，在此阶段识别设计缺陷和产品风险可以节省大量工作时间，并在代码编写之前完全防止漏洞引入代码库。在设计阶段可以解决的风险包括:

- 无法识别和保护数据流
- 框架和编程语言选择固有的漏洞
- 安全控制框架解决的常见风险
- 从威胁建模中发现的不良设计决策
- 与不完整或有缺陷的业务逻辑相关的漏洞

3.2开发（编码）

安全编码是基于设计阶段的详细要求进行应用开发的阶段。在这里，开发人员选择框架和库，编写代码，并创建单元测试。随着设计的决策和代码

的重写，应用程序将处于重大变化之中。这一阶段聚焦于设计和业务逻辑的安全实现，并基于安全编码实践开发应用程序后再提交到源代码存储库（SCR）。这一阶段可能导入安全脆弱性的风险包括:

- 使用不安全的协议、框架或库
- 存在漏洞的源代码和依赖关系
- 托管环境中存在漏洞且简陋的编程基础设施
- 关键代码变更缺乏透明性

3.3集成和测试

集成和测试聚焦于将应用程序的组件组装成可发布的工件，并测试新的发布以确保其满足设计要求。应该对运行中的应用程序，以及程序的组装和发布过程进行明确的安全项测试。这个阶段应重点测试产品的应用程序在运行时的安全。本阶段的风险包括:

- 构建流程缺乏完整性
- 运行时环境或源代码安全配置错误
- 运行时应用程序或基础设施的漏洞利用
- 不恰当的测试或缺乏安全测试标准

3.4交付和部署

交付和部署聚焦于形成预构建的发布制品，并将其在产品中发布。这需要确保安全检查已经完成，并使用正确的新制品和环境配置进行部署。本阶段应重点控制产品变更、变更行为记录以及产品发布的整体完整性。本阶段的风险包括:

- 发布不正确或损坏的制品
- 产品中引入不安全的环境配置
- 劫持发布流程，引入未经授权的变更

3.5运行时防御和监测

运行时防御和监测聚焦于在应用程序部署到生产环境后进行应用程序的安全管理。管理包括监测违规或其他恶意和滥用行为指标，开展持续的安全测试，并检测传统或新发掘的漏洞。本阶段的风险包括:

- 业务逻辑攻击
- 拒绝服务或其他业务连续性攻击
- 新发掘的漏洞利用
- 合规性监控

设计与架构

概述：设计部分参考了产品设计过程中可以应用的技术和流程。我们认为设计是连续的，所以可以通过设计激活新产品的特性和变化途径。在设计阶段未考虑安全设计的情况下，在部署或运行时导入安全措施将导致更高的操作影响和成本。导致检测难以度量，形成影响开发速度和发布时间线的安全瓶颈。

能力

活动	描述	技术	流程
威胁模型	一个结构化的过程，用于识别安全需求，在抽象层面上精确定位安全威胁和潜在漏洞，量化威胁并确定补救设计方式的优先级。	✓	✓
安全用户故事	安全功能需求陈述，确保软件的许多不同安全属性之一得到满足。与功能作为故事相同的方式提出安全要求。安全故事可以划分为开发人员任务，并在开发工作列表中进行优先级排序。	✓	✓
动态价值安全映射	流程图中的活动从需求到发布进行映射，以识别流程低效性和自动化机会。	✗	✓
架构规范和工件	使用和开发所有组织技术资源（包括IT、IoT和OT）的基本原则和目标。它们反映了企业各部门之间的共识水平，并构成了未来在技术、流程和人员方面决策和变更的基础。	✗	✓
风险管理（左移）	产品团队识别和有效描述产品风险（包括应用程序、组件、平台和依赖资源）并实施改进以缓解或预防可能的负面事件的过程。	✗	✓

优点	缺点	输出
• 提高评审速度 • 更易于保持规模一致性 • 减少运营影响和对人员的依赖 • 计划和衡量安全工作/需求列表 • 尽早了解风险和威胁	• 需要安全体系架构专业知识	• 以威胁为导向的体系架构设计评审 • 安全要求和需求列表的优先级 • 按工作量/时间衡量安全工作 • 形成文件的原则、模式和方法 • 记录设计风险和不可接受的风险场景

图 4:设计和建筑师矩阵

编程

概述：可以在开发应用程序时应用的功能。编码安全控制依赖于自动化，因为与人工评审相比，工具可以更好、更一致地识别代码中的弱点和漏洞。如果在编码阶段不考虑安全性，就有可能出现源代码中的漏洞并将其部署到生产环境中。在SDLC的后期阶段，修复安全漏洞和弱点的成本将更高

Capability			
Activity	**Description**	**Technology**	**Process**
开发者测试	由于需要高质量、安全的代码，基于角色的安全培训可以帮助开发人员在用各自语言编写代码时识别漏洞，而不会出现常见漏洞。	✓	✓
容器加固	通过更新包和查找不安全的默认配置和已知漏洞，解决了基础镜像和容器编排的弱点。它使新的黄金基础镜像能够在流水线中安全使用。	✓	✓
安全单元测试	采用源代码中可测试的小部分来确定安全控制的有效性。	✓	✓
安全钩子	不论是代码提交前或提交后，代码提交到git存储库（git提交操作）将触发脚本，自动识别代码中的问题，比如识别源代码中的敏感信息。	✓	✗
代码检查	集成开发环境的一种分析方法，用于标记编程错误、BUG以及风格和构造错误。	✓	✗
软件组成分析	帮助识别和修复软件产品的开源和第三方组件中的漏洞。SCA工具扫描应用程序代码库和工件存储库，以识别安全漏洞。	✓	✗
静态应用程序安全测试	分析源代码、字节码和二进制文件，找出静态代码中的安全漏洞。它也被称为静态代码分析或白盒测试。	✓	✗
基础设施即代码（IaC）分析	使工程师能够在执行安全验证检查的同时对云基础设施进行版本控制、部署和改进。这为主动改进云基础设施的架构并尽早解决安全问题提供了机会。	✓	✗
同行评审	团队成员的源代码在被实现到代码库中之前由一个或多个人员进行评审。	✗	✓

优点	**缺点**	**输出**

• 尽早并经常发现安全问题 • 增加了对代码质量的反馈 • 缩短了修复漏洞的时间 • 预防危险代码行为的安全检查	• 代码扫描仅涵盖一小部分漏洞，不包括运行时问题或设计缺陷问题 • 可能会导致误报。	• 通过SAST和SCA工具形成代码报告 • git存储库中的集成功能 • 具有安全意识的开发人员 • 开发工作流中嵌入的安全活动

图 5：编码矩阵

集成和测试

概述：用于应用程序/产品功能安全测试的工具和流程。如果没有这些工具和方法，安全漏洞和弱点将成为被利用的根源，并导致数据泄露和业务连续性问题。

能力

活动	描述	技术	流程
动态应用程序安全测试	一种黑盒安全测试形式，通过模拟针对应用程序外部接口的外部攻击来分析正在运行的应用程序以识别安全漏洞。		
交互式应用程序安全测试	一种将SAST和DAST技术相结合的方法，用于扫描运行时的应用程序，同时也扫描单独的代码行。		
API 测试	一种将应用程序编程接口（API）作为通用消息传递流水线进行测试的方法，以确保功能不存在可被利用的弱点或漏洞。		
模糊测试	一种自动化和动态的软件测试技术，使用无效、意外或随机数据作为应用程序或过程的输入，以评估其响应和揭示安全缺陷。		
渗透测试	一种针对网络、系统或应用程序组件中发现的脆弱性的手动安全测试方法，以确定是否可以利用漏洞来危害环境中的资源、整个应用程序或其数据。		
容器测试	利用攻击方法针对容器基本镜像和容器编排方法进行测试。		
完整性检查	在连续部署阶段之前，对代码制品和容器镜像进行加密签名并验证其完整性的自动化过程。		

优点	缺点	输出
• 在运行时发现安全问题 • 识别并解决集成过程中的安全问题 • 自动化流程，为开发人员提供更快的反馈	• 建立流程，使用安全测试工具编写集成成本高	• 可证明的安全态势保障 • 真实状况的可报告内容

图 6：集成和测试矩阵

交付和部署

概述：部署前的安全检查，以确保应用程序/产品部署到安全的基础设施上。如果部署中未考虑安全性，可能导致应用程序/产品存在漏洞和缺乏安全实践，发生在生产环境中被利用进行攻击的风险。

能力

活动	描述	技术	流程
GitOps	使用git存储库来管理基础设施和应用程序代码部署。 在GitOps中，git存储库是部署状态的事实来源， 而不是服务器配置文件。	✓	✗
护栏	为构建云环境提供持续治理的高级规则。护栏是实施的预防性或检测性控制措施，有助于管理资源并监控跨资源的合规性	✓	✓
环境隔离	开发、测试、预生产和生产环境的逻辑分离和管理。	✓	✓
秘密与密钥管理	用于管理数字身份验证凭证（秘密）和加密密钥信息的工具和方法。这包括用于应用程序、服务、特权帐户的密码、加密密钥、API和令牌，以及用于身份验证和加密使用的其他敏感凭据。	✓	✓
CI/CD 流水线防护	通过记录和监测变更，并根据流水线状态验证CI/CD流水线的完整性。	✓	✓
系统加固	涉及保护服务器数据、端口、组件、功能和权限的过程。	✗	✓

优点	缺点	输出
• 快速安全的部署 • 托管服务的一致配置 • 建立安全控制的基线 • 建成后自给自足	• 需要对基础架构和代码的托管服务重新架构	• 部署代码的安全配置流水线 • 安全配置的主机环境

图 7：交付和部署矩阵

运行时防御和监测

概述：应用程序/产品发布到生产中后可以应用的功能和实践。运行时安全性通过更好地识别效率低下、漏洞和弱点并启用事件响应来实现持续改进。

能力

活动	描述	技术	流程
混沌工程	测试分布式计算系统，以确保其能够承受意外中断。使用故障模式、影响分析或其他策略深入了解组织系统中的潜在故障点。		
云安全态势管理	识别云中的错误配置问题和合规风险。其重要目的之一是持续监测云基础设施的安全策略实施差距。		
性能管理	收集、分析和标记日志数据的过程，以识别和响应应用程序和基础设施上的事件和事故，同时识别指标、根本原因、趋势和性能。		
攻击面管理	E外部攻击面管理（EASM）识别并管理面向互联网的资产和系统面临的风险。这里指的是识别面向外部资产的工具。		
事后刨析	用于确定项目失败（或严重业务影响中断）的原因以及未来的预防措施。旨在构建可降低未来风险的流程改进。		
紫队	网络安全模拟是模拟网络攻击响应的一种测试技术和流程，通常需要一个仿制的环境来有效地“对抗”现实场景中的潜在攻击。		
漏洞修复	评估、处理和报告系统及其上运行的软件中发现的安全脆弱性和错误配置的过程。		
事件响应	准备、分析和响应事件的过程。		

优点	缺点	输出

• 关注事件响应改进 • 持续监测生产环境中的安全态势 • 生成合规性指标的机会 • 将数据反馈至设计阶段	• 建设流程代价高	• 基础设施托管环境情况的持续报告 • 优化事件响应准备 • 用于提升安全性和性能管理的有意义的数据和指标

图 8：运行时防御与监控矩阵

4 设计和架构

设计部分可以涉及在产品设计过程中应用的技术和工具。由于设计是持续进行的，因此通过设计可以对新的产品功能和变进行处理。如果在设计阶段不考虑安全性，那么在部署或运行时产生的安全措施将带来更高的运营和成本影响。因此，这些安全措施将难以扩展，从而导致安全瓶颈，降低开发速度并影响发布时间。

4.1 威胁建模

威胁建模是一种技术，是用来识别和概述，计划的或现有的系统或应用程序的关键威胁、攻击向量以及预防措施的技术。威胁建模的目的是:

- 识别、分析和评估安全威胁
- 制定缓解措施的优先级
- 协助并报告攻击面的分析以及降低风险

以下是《CSA 云威胁建模》中提到的核心威胁建模活动:

1. 为威胁建模活动确定威胁建模安全目标，重点关注保密性、完整性、可用性和隐私等关键方面，例如：

a. 保护公司数据库中客户或受监管的信息不受外部攻击者的影响；

b. 确保电子商务网络应用程序的高可用性

c. 选择具有最小攻击面或客户安全责任最小的云应用模式。

2. 根据提供的系统或云应用的综述，设定有关考虑中的系统和/或云基础设施的评估范围。通常涵盖各种组织资产等领域，包括所使用的技术栈、现有的安全控制、部署方案、用户类型，以及任何在威胁建模中需要解决的特定安全要求或监管要求。

3. 系统或应用程序可分解为各子系统，并检查各个组件之间的交互。这一阶段的关键活动包括：

a. 理解信任边界（面向外部和内部的、特权的、未经认证的，等等）。

b. 确认系统的输入和输出以及数据格式。

c. 绘制系统中的数据流

4. 识别和评估潜在的威胁、攻击类型以及恶意用户如何滥用给定的系统或其功能。根据OWASP Top 10的调查结果，可用于确保威胁建模活动中考虑到最关键的安全风险。一些常见的威胁是未授权访问、拒绝服务、信息泄露等。STRIDE可对威胁进行分类，DREAD框架可对威胁的严重性进行评估。MITRE的ATT&CK®框架可以帮助了解检测和缓解行动的准备情况和有效性。

5. 识别系统设计和组件中的弱点和漏洞，以帮助安全决策并定义安全测试的范围和性质。

6. 设计并优先考虑适用于预先确定的威胁的缓解措施和控制措施，并思考这些控制措施如何降低威胁或风险水平。

7. 沟通已确定的威胁、潜在影响和严重性，以及适用和建议的控制措施。提供建模数据和洞察力，并呼吁通过设计或影响来缓解威胁的行动。

组织通过威胁建模获得对设计实施早期和经常评审的好处，解决方案设计阶段初期，就可以设计富有成效的决策（包括技术采购和包括产品功能的决策）。对解决方案进行专题评审并确定控制攻击技术的机制的能力有助于优化安全解决方案，同样也使产品团队的相关利益者能在编码期间完成改善工作。

可以通过工具来实现威胁建模，该工具将威胁通过数字化的方式记录在架构图中。在某些情况下，可以自动识别威胁方法。然而，威胁建模最常见的方式，是通过对分析目标关键系统的意图，以人工评审的方式进行建模；基于对目标系统的了解，必须对数据流和数据流中组件之间的信任边界进行建模。

虽然隐私设计超出了本文的范围，但威胁建模活动是将隐私设计评审纳入产品的有效方法。根据产品类型，组织的地理位置和司法管辖范围不同，法规要求（如 GDPR）将需要组织证明其对个人数据和医疗数据的处置、使用和存储的合规性（如：数据处理协议、数据隐私影响评估）。

组织通常可以选择商业版的供应商来实现威胁模型自动化。这方面的例子有 IRIUS Risk 和 ThreatModeler，而 OWASP Threat Dragon 和 Cairis 是开源方案的例子。

4.2 安全用户故事

安全用户故事（Security User Stories）是对安全功能的陈述，确保软件的许多不同的安全属性得到满足。可以使用工具或有效的流程来确定安全需求，定义新功能或对现有功能的补充，解决特定的安全问题或修复漏洞。

为软件开发人员/工程师建立安全用户故事是将安全融入软件开发的有效方法。可以参考现有的安全合规要求、组织政策、行业准则和标准，实现则需要项目利益相关方之间的协作。像 INVEST[8] 这样的方法可以帮助支持

创建安全用户故事和任务，每个故事和任务应该具备以下特点:

- **独立性**：应该是自成一体的
- **可协商性**： 应该留有讨论的空间
- **有价值性**： 必须为利益相关者提供价值
- **可估算性**： 应该能够估计其规模
- **颗粒度小性**： 应该能够进行计划、任务和优先级排序
- **可测试性**： 开发应该能够进行测试

在安全和开发之间建立的桥梁不应该是单向的。安全政策应该被严格映射到开发人员和运维人员可以完成的技术故事范围内。软件开发人员、DevOps 工程师和架构师应该提供完成状态的反馈，以便编写信息安全政策的安全合规团队有能力了解当前软件生命周期阶段/迭代的安全状况。创建安全模型和加固指南将支持安全合规活动。

虽然安全用户故事可以更好地向业务利益相关方证明：代码的设计和配置都有组织批准的安全控制，但也可以通过工作量和时间来衡量活动，使安全工作在代办事项中得到优先安排。

安全故事通常是根据组织的政策手动编制。在不考虑政策与产品开发的情况下，OWASP 应用安全验证标准可用于应用安全的要求，CSA CCM 可有效建立云安全需求。然而，自动化安全故事流程的机会（例如，SD Elements）可能无法解决组织政策调整和工作优先级的问题。

4.3 价值流安全映射

价值流安全图（value stream security map）将描述从产品开发的想法是如何传递给到客户的。对工作的可见性代表了团队对业务到客户的工作流程的理解程度，以及他们是否对这一流程有可见，包括产品和功能的特性。

在考虑安全控制之前，了解应用程序开发生命周期中的工作流程是很重要的。VSSM 作为一个有用的工具，包括以下内容:

- **利益相关方：** 负有责任的团队和个人
- **活动：** 在阶段中执行的能用活动
- **前置时间：** 从上游流程接受工作到将该工作完成移交给下游流程的时间
- **流程时间：** 如果执行人员拥有完成工作所需的必要信息和资源，并能连续完成工作；那么完成单项工作所需的时间
- **完整和准确度（%C/A）：**从上游过程收到，可以直接使用而无需返工的比例

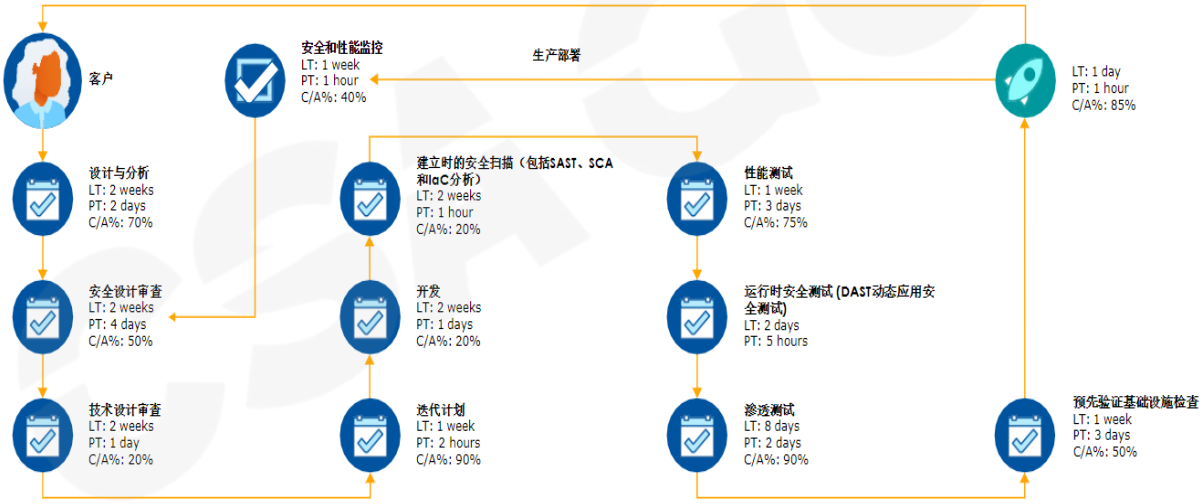

图 9：VSSM 示例

可以按照以下方式创建VSSM[9]:

1. 识别问题：从客户的角度来定义问题。记录问题。
2. 呈现问题：确保所有团队成员理解问题。将相关内容的理解和一致性记录文档。
3. 确定团队：选择具有正确知识和经验的团队。

4. 赋予团队权力：为团队提供解决问题所需的权限和资源。

5. 进行走查：对复杂的过程进行走查，确保对步骤和环境的理解，并建立共同的目标。

6. 初始映射：准备一个基本的映射作为起点。

7. 设定流程限制：建立流程的限制措施/指标，以确保和维护适当的范围。

8. 收集数据 – 收集/分析数据 – 数据包括资产、资源、循环时间、提前置时间和正常运行时间

9. 绘制当前流程图：前几个步骤代表了当前的状态，并作为一个基线。

10. 绘制未来流程图：接下来，明确需要完成的目标，并将其映射到当前的流程中。

11. 实施未来地图：未来流程图可用于指导改进过程，为团队成员提供有针对性的目标，需要采取分阶段的方法来实施改进。

价值流图提供了一系列业界广泛接受的优点。同时，安全工作的整合提供了额外的好处，[10] [11]：

- 端到端的安全性和可见性影响产品开发
- 更好地将DevSecOps与业务和时间目标联系起来，提供更好的决策信息
- 在决定加入或移除安全工具和流程时，不同的利益相关者之间可以进行更好的合作，以获得更快的反馈
- 识别安全瓶颈和限制性过强的耗时任务，可在不同的价值流阶段进行更好的优化

虽然价值流安全图活动通常是手动和本地执行（如使用 Atlassian 软件），对单个项目而言，也可以选择商业供应商（如 LeanIX, Plandek, HCL tech）

提供技术平台来创建和管理价值流图。

4.4架构原则和制品

架构原则和制品是所有组织技术资源的基础性通用原则、目标和实践。架构原则反映了某种程度的共识，并形成了对技术、流程和人员进行未来决策和变更的基础。DevSecOps 的架构原则有助于规划创建工作模式，用于构建安全和批准的解决方案的指导方针，以及对团队拓扑的有效理解。遵循这些方法和模式，可以重新设计人们构建产品的方式，以避免漏洞的引入，并检测和防止可能危及产品安全的错误。

DevSecOps 的原则因组织而异，但应该与业务目标保持一致，同时补充来自软件开发的目标价值。DevSecOps 是 DevOps 的进化，应该重视速度和自动化的价值。它通过体现注重速度、创新和协作的价值观，消除安全门和耗时的活动。以下是建立一个有效的 DevSecOps 基础的一些共同原则:

- **最小可行的安全产品：**实施安全控制和流程的基线——组织开发的每个产品都应该达到的最低门槛。最小可行安全产品提供了一个如何实现这一目标的检查清单的例子。但是，最小可行安全产品应该比检查表更多，并告知组织建立有效的基线，进一步完善和构建产品的安全功能。

- **安全是产品功能：**一种将安全功能作为产品特性引入以提高安全性的方法。产品团队和解决方案可以解决风险和威胁，而不是在开发末期的预部署阶段添加安全解决方案。安全用户故事提供了一个帮助实现这一目标的示例方法。实现依赖于关键利益相关方的支持。

- **安全是集体的责任：**在 DevOps 的文化中，开发人员和运维人员对于他们构建和管理的软件有共同的理解和共同的责任。这意味着在开发、IT/运维和"业务 "12 之间增加透明度、沟通和协作。安全不应该认为仅仅是安全团队的责任。所有利益相关者都应该承担起构建和管理服务的安全责任。有关实施的信息，请参见 CSA Pillar 1: Collective Responsibility.

上述三项原则被视为基础性原则。除此以外的原则需要与组织的目标、团队结构、预算、IT 环境和架构相关。为了解一整套 DevSecOps 原则的例子，美国国防部发布了一个 DevSecOps 参考架构指南[13]，其中概述了以下内容:

- 消除瓶颈（包括人为因素）和手动操作。
- 开发和部署活动尽可能自动化。
- 从规划和需求到部署和运维，采用通用工具。
- 倡导敏捷的软件原则，支持小型的、渐进的、频繁的更新，而不是大型的、间歇性的发布。
- 在整个SDLC过程中将应用跨职能的技能组合应用于开发、网络安全和运维等，采用并行的持续监控方法，而不是按顺序应用每种技能组合。
- 必须对底层基础设施的安全风险进行测量和量化，以了解总体风险和对软件应用的影响。
- 部署不可变基础设施，如容器。不可变基础设施的概念是一种IT战略，其中部署的组件被整体替换，而不是原地更新。部署不可变基础设施需要通用基础设施组件的标准化和仿真，以实现一致和可预测的结果。

除了基本的 DevSecOps 原则外，还应该认识到 DevSecOps 原则应该囊括业界在 DevOps 和云计算中所宣扬的内容。AWS 认可 5 个关键原则[14]，这些原则构成了其良好的云架构框架的一部分:

- **以代码方式进行运维：** 在云中，应用程序代码的工程纪律也可以应用于基础设施。整个工作负载（应用程序、基础设施）可以作为代码部署，并通过代码更新。操作流程可以作为代码实现，并通过响应事件的触发来实现自

动化，这减少了人为错误，通过以代码方式进行运维，可以实现对事件的一致性响应。

● **进行频繁的、小型的、可回退的变更：** 设计工作负载，允许组件定期更新，以小幅增量进行变更，如果变更失败，则可以回退（尽量不影响客户）。

● **经常完善操作程序：** 在使用过程中改进操作程序，随着工作负载的变化，适当地改进程序。定期评审和检验程序是否有效，以及团队对这些程序的熟悉程度。

● **故障预测：** 执行“预检”演习，以确定潜在的故障源，从而消除或减轻它们的影响。测试故障场景并验证对影响的理解。测试响应流程，以确保它们有效，并确保团队熟悉它们的执行操作。设置定期的演练，测试工作负载和团队对模拟事件的响应。

● **从所有操作失败中吸取教训：**通过从所有操作事故和故障中吸取的教训推动改进。在团队和整个组织中分享所学的知识。

谷歌云[15]还推荐了一套云原生架构的原则:

● **自动化设计：** 自动化一直是软件系统的最佳实践。尽管如此，云比以往任何时候都更容易自动化其上的基础设施和组件。尽管前期投资较高，但在工作量、弹性和性能方面，支持自动化解决方案几乎总是会在中期得到回报。自动化流程可以比人更快的修复、扩展和部署系统。

● **智能处理状态：** 存储“状态”，无论是用户数据（例如，用户购物车中的项目，或员工编号）还是系统状态（例如，正在运行的作业实例数量，在生产中运行的代码版本），都是构建分布式云原生架构的最困难方面。因此，设计系统时要故意考虑存储和设计组件的无状态性。

● **选择托管服务：** 云不仅仅是基础设施，大多数云提供商都提供丰富的托

管服务集合，提供各种功能，以解决管理后端软件或基础设施的困扰。然而，需要组织对利用这些服务持谨慎态度，因为他们担心被“锁定”在指定供应商。这个担忧是有道理的，但托管服务商通常可以为组织节省大量的时间和运营开销。是否采用托管服务归结于可移植性与资金和技能方面的操作开销。

- **践行深度防御** ：传统架构对周边安全充满信心，简单地说是一个加固的网络边界，内部有“可信的东西”，外部有“不可信的东西”。不幸的是，这一方法一直很容易受到内部攻击和外部威胁的影响，如鱼叉式网络钓鱼。此外，提供灵活和移动办公的压力越来越大，进一步破坏了网络边界。

- **架构是根本：** 云原生系统的核心特征之一是他总是在不断发展，架构也是如此。随着组织的需求、IT环境和云提供商的变化，始终寻求完善、简化和改进系统的体系结构。

虽然原则可以让人感觉像高级抽象的语句，但通过原则对业务目标进行技术和安全目标的识别和映射是有价值的。在 DevSecOps 活动中尽早定义原则，以产生以下好处:

- 难以掌握的复杂技术环境中，建立原则有助于揭开衡量成功所需完成的工作的神秘面纱
- 它们构成了一个与业务目标相关联的连接目标，并可由高级业务利益相关者评审
- 它们可被用作指导未来决策和技术、人员和流程变革的指南
- 它们可以链接到敏捷仪式和实践
- 它们可以链接到花费时间的日常运营工作团队

除了美国国防部（DoD）、亚马逊网络服务（AWS）和谷歌云引用的 DevSecOps 和云安全原则之外，还可以从 NCSC 的云安全原则和 Grafana 的云原生安全宣言中找到更多示例。

4.5风险管理（左移）

产品团队识别和有效描述产品风险（包括应用程序、组件、平台和依赖资源），并实施改进以缓解或防止可能出现的负面事件。虽然风险识别和评估应该尽早的在开发过程的设计阶段进行，但在某些情况下，随着时间的推移，可能会出现新的风险。

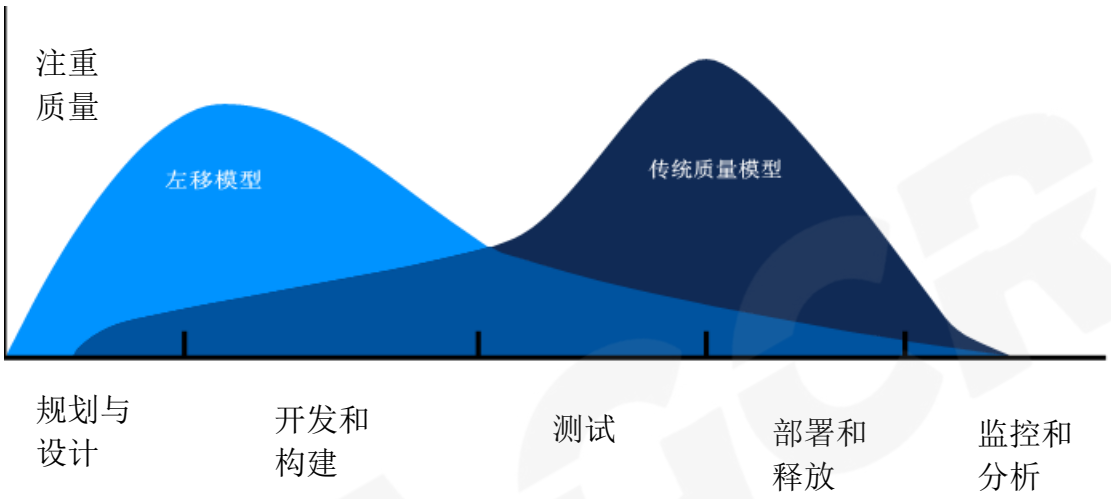

图 10：风险管理左移

形成风险声明的能力在描述安全问题和确定工作优先级方面非常重要。在 DevSecOps 和现代软件开发范式中，风险表达将在操作上有效，参考威胁（在威胁建模中识别）以及可利用的漏洞类型，请参见**图 11**。务实的决定是通过确定工作的优先级和形成风险声明的能力做出的，其中包括以下领域:

- **资产**：受保护的资产类型，范围从客户数据到知识产权（IP），以及客户是否受到影响。

- **威胁**:：威胁类型被视为内部威胁，如恶意内部人员或外部威胁，如高级可持续威胁（APT）。

- **漏洞**：应考虑评估哪些漏洞可被利用执行威胁攻击（例如，远程代码执行、代码注入、逆向工程）。

- **补偿控制**（即诱发条件）：评估和枚举现有的缓解措施，例如分割网

络的防火墙、Web框架或产品界面的输入过滤功能、二进制混淆。

- **描述风险**：影响是通过考虑受保护的资产类型和威胁（例如，知识产权和数据泄露）获得的。而可能性则考虑影响被利用的可能性（例如，通过一个漏洞）和潜在的缓解措施（例如，在内部防火墙网络的服务器端请求伪造（SSRF））。

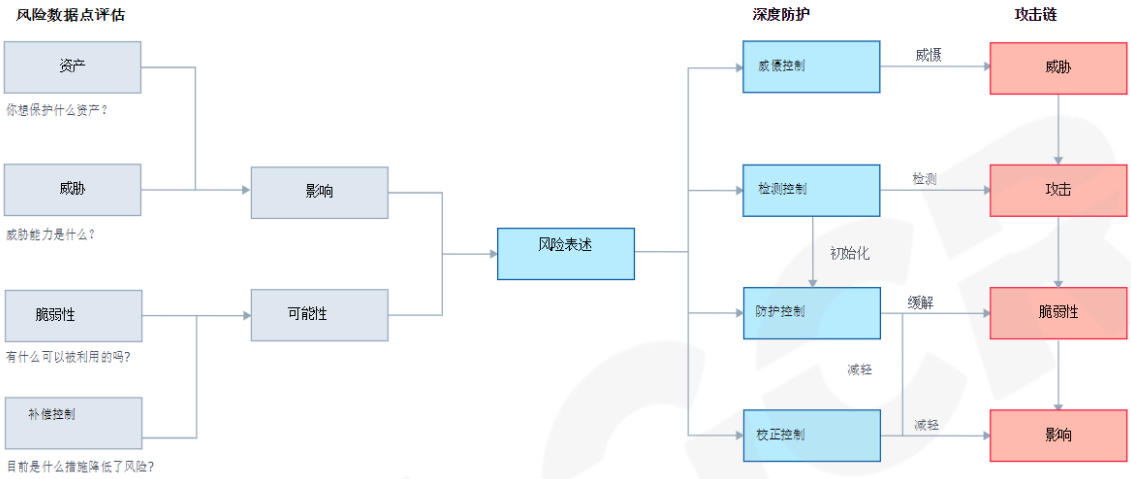

图 11：风险表达和控制

风险左移意味着解决未知风险，以减少或消除与产品设计相关的风险。在设计时而不是以后执行风险分析，可确保它有助于时间、成本和资源的管理，并更好的预测管理障碍。左移风险还能促进解决方案架构师和软件工程师等产品团队的利益相关者更容易识别和提出风险，而不是经历他们不熟悉的单一且耗时的风险管理流程。

组织的风险偏好和风险管理方法在利用人员、流程和技术能力时可影响决策。缺乏风险管理可能危及组织的运作预算，并可扰乱公司及其利益相关者的整体士气。组织在安全运营方面应该处于更加成熟的状态才能采用 DevSecOps，因为这需要长期投资和微调。

虽然举例的风险框架受组织、行业部门和地理位置的影响，但这些材料就像 CSA 云风险管理的视角一样，有助于确定和管理关键风险的主题。

5 开发（编码）

可以作为应用程序应用的功能正在开发中。与手动人工评审相比，对依赖自动化作为工具的安全控制进行编码，可以更好、更一致地识别代码中的弱点和漏洞。如果不在编码阶段考虑安全性，在源代码中的漏洞将无法识别并部署到生产环境中。安全漏洞和弱点在 SDLC 延后的修复成本将会更高。

5.1对开发人员培训

对开发人员进行培训可以为开发团队提供责任、充分的培训、工具和资源，以验证软件设计和实施是安全的。由于要求将代码安全作为安全系统生命周期的一部分，充分的安全编码培训有助于开发人员在用各自语言编写安全代码时识别错误。这将引入漏洞和设计缺陷的风险降至最低

研究表明，[16] [17] [18] 开发人员通常是自学成才的，这导致超过 50%的软件开发人员缺乏安全编码意识和技能。即使开发人员受过一些软件工程方面的教育，该课程通常也不包括安全编码实践。这种培训缺失导致 SDLC 中常出现 不安全的编码实践和常见的应用程序漏洞。

开发人员培训的有效先决条件是评估安全 SDLC 的成熟度和相应的安全编码实践。为了成功实现和实施开发人员培训，其侧重于与每个开发团队成员相关联的角色和责任。对于开发人员培训的任何情况，成功的实施是通过游戏化培训体验来为完成的培训提供奖励，从而激励实际进行培训。此外，此类培训包括与开发人员正在使用的技术和平台相匹配的指导和实例培训。培训有多种形式，主要的问题将在下文中进行讨论。

讲师主导的培训：讲师主导的培训是在实践中采用在线或面对面的为一群开发人员提供教育的方式。该培训的优势是可以实时与讲师进行反馈和讨论。这种培训在使用引人入胜的演示时尤其有效，特别是讲解如何利用特定的漏洞。这有助于开发人员了解攻击者是什么，并使他们了解为什么需要输入验证和过滤，以及如果代码未按预期运行，可能产生的影响。SAFECode 通过按需网络广播

提供在线社区资源，通过按需提供软件安全方面的讲师主导的开发人员培训。OWASP 安全知识框架（SKF）提供了一些实用的安全编码示例，用于讲师主导的培训。

在线培训： 开发人员有时间参加的一种在线培训形式。这种方法适用于更高级的形式，允许参加课程的人在需要时反复复习材料。OWASP 果汁商店（OWASP Juice Shop）是一个不安全的 web 应用程序，可用于安全培训和 CTF 意识演示。OWASP 果汁商店还包含安全编码挑战，特别是针对安全编码开发人员的培训。此外，开放安全基金会（OpenSSF）通过 Linux 培训和认证平台为软件开发人员提供了关于安全编码的免费课程 [19]。安全编码培训的一个商业解决方案是安全代码战士，它提供了一个电子学习平台，包括培训、锦标赛和基于技能的途径评估。

安全意识培训： 安全意识培训旨在向最终用户提供有关当前威胁、安全最佳实践和政策预期的信息。然而，事实证明，仅仅依靠安全意识的年度培训是无效的，尤其是在建安全编码实践方面 [20]。相反，有效的安全意识培训针对的角色是开发人员。像 SANS Web 应用程序安全意识培训这样的组织提供基于角色和循序渐进的专业培训。

指导：指导是开发人员培训的一种有效形式，因为它通常鼓励团队中的开发人员掌握安全实践。一种形式的指导是通过安全倡导者计划正式形成的，安全团队与提名的安全捍卫者切合作，就安全漏洞和安全编码实践教育其他开发人员。OWASP 安全文化项目和 OWASP 安全捍卫者行动手册提供了一个逐步建立安全捍卫者计划的过程，该计划侧重于安全团队，为开发团队中选定的安全捍卫者提供指导。

利用上文提到的对开发人员的培训组合提供了几个好处。研究 [21] 表明，开发人员所受到的安全培训水平与整个软件开发团队的安全意识呈正相关。有效的开发人员培训还使开发人员能够更快地生成安全代码，并防止在安全系统开发生命周期的设计和实施阶段引入常见的安全漏洞。

5.2 安全钩子（Hook）

Git 钩子脚本对于在代码评审之前识别代码中的常见问题非常有用。开发人员经常使用这些钩子来防止在代码库中引入轻微的错误，例如缺少分号、明文密码或尾随空格。当使用敏感信息扫描工具检测到安全凭据时，甚至在与通用软件应用程序安全测试规则匹配之后，安全钩子脚本也可以阻止提交。

Git 安全钩子可以在 Git 工作流的任何阶段设置。然而，它们通常在提交前和提交后阶段被观察到，见**图 12**。Git 工作流中可以应用钩子的其他阶段包括:

- 预提交
- 预推
- 合并前提交
- 准备消息提交
- 提交消息
- 提交后
- 检出后
- 合并后
- 后期重写

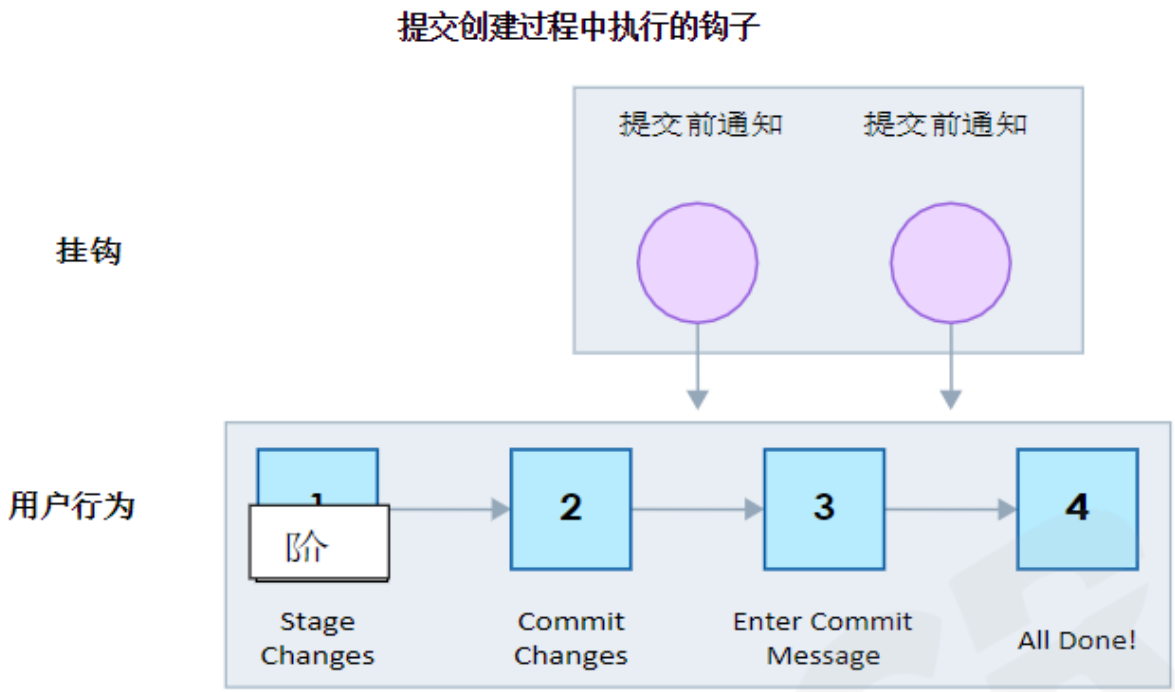

图 12：Git Hooks 钩子

需要注意的是，git 安全钩子不能替代其他持续集成安全测试方法，原因有如下两个:

1. 开发人员的时间很宝贵，他们可能一天会多次提交或推送，虽然安全钩子可以防止引入明显的安全问题，例如代码库中的凭据泄露，但它们不应该占用几秒钟到最多一分钟的时间来阻碍开发工作流程。

2. 对于非容器标准化的开发环境，不能保证钩子会像在这样的场景中那样正确配置；这种控制的设置将基于信任和开发人员的选择。

虽然安全钩子可以有效地捕捉开发人员的错误，但工具还没有成熟到可以进行完整的代码扫描。可以在 Git-scm 和预提交上找到一些安全钩子的示例。

5.3代码检查（linting）

代码检查被定义为自动检查源代码的“坏代码气味（不良代码）”，以标记代码错误、bug、风格和构造错误 [22].链接工具是静态代码分析的一种基本

方式。链接工具通常作为可配置的插件提供，用于识别和报告错误代码的误用或不良代码的模式。链接工具和高级静态分析工具之间的界限是模糊的。链接工具是多种多样的，并且是限定于特定的编程语言，特别是开发团队在集成开发环境（IDE）中作为插件或在代码提交钩子期间使用。

一种实用的方法是为编码语言仔细选择一个链接工具[23]。在选择代码链接工具时，必须找到并仅选择一个链接工具以减少冗长。此外，还必须确保链接工具充分支持使用该链接工具的编程语言。链接工具还可以独立运行，并构建允许自定义扩展和 API 支持的集成工具。代码链接是一种特别有效的应用领域，是强制执行安全编码标准、机密（隐私）检测、失效代码构造检测。另一个被证明有用的领域是检测循环复杂性，这验证了程序的复杂性，因此也查验了程序的可读性和可维护性。

当前有各种链接工具可以使用，其中大多数限定于一个特定编程语言，概要介绍如下:

- Java: CheckStyle[24] 是JAVA的一个链接器，可以用于链接分析和静态代码分析。它是一个专注于为Java开发人员遵循编码标准的工具。Checkstyle同时也是一种开发工具，可以帮助程序员编写符合编码标准的Java代码，包括安全代码。

- JavaScript: 对于JavaScript，最广泛使用的链接器是JSLint25和ESLint26，其中ESLint专注于识别不良代码，其中JSLint被用作静态分析工具，以遵守编码规则。

- Infrastructure as Code（基础设施即代码）: TFlint是一款面向Terraform的开源链接器插件。它警告开发人员已弃用的语法和未使用的声明，并在基础设施即代码中执行最佳实践和命名规则27。可以使用其他插件扩展tflint，也可以在预提交钩子期间运行。面向Kubernetes，Kubelinter28分析Kubernetes的YAML文件和Helm图表，并根据各种最佳实践对其进行检查，重点关注生产准备情况和常见的安全错误配置。

- Python: Bandit [29] 并不是一个严格意义上的链接器，它是一个高级的静态代码

分析工具，旨在检测Python代码中的常见安全问题.

● Cross-language: 跨语言静态代码分析的一个例子是PMD30，它可以检测即将出现的编程缺陷，包括未使用的变量、不必要的对象创建和空的catch博客。除了Java之外，它还支持其他语言，如Scala、Apex、Swift和PLSQL.

总体而言，代码链接器有很多好处并提高安全效率。使用代码链接器的一个主要优点是所见即所得，能在编写代码时立刻向开发人员提供反馈。链接器可以检测代码中可能会导致安全缺陷的错误，特别是它可以告知开发人员，哪些是不推荐、易受攻击的类和方法，并减少使用有漏洞软件组件的风险。链接器对于减少错误和提高整体质量很重要，它有助于加快开发速度，并通过在开发人员编写源代码时发现错误以降低成本。链接器可以扫描代码的安全缺陷、格式化或样式化问题等最佳实践，使代码更容易维护和阅读。链接器提高了代码的质量，从而减少了开发人员导入安全缺陷的可能性。由于开发人员更容易理解代码，可读的代码也能使代码更安全。当配置正确时，链接器允许强制执行安全编码标准，特别是应用于错误和异常处理的代码编写。

然而，需要指出的是，并不是每种语言都有“高质量”的标准链接器工具，因为每个框架通常都有一个或几个链接器。此外，链接器的情况不同，链接工具、版本或配置可能会导致不同的结果。代码链接还会因其冗长而导致误报和信息过载。

5.4软件组成分析

在构建现代应用程序时，获得开发速度的一个关键方法是利用开源或第三方的组件。然而，使用这些组件会带来诸如安全漏洞、许可义务和潜在代码质量问题等风险。软件组合分析（SCA）是组件分析的一个子集，是一个过程和一套工具，用于识别、纠正开源和第三方软件产品或应用程序组件所带来的风险。

SCA 工具为团队提供了整个应用程序的可视性，并能够创建和强制执行安全策略，以减轻使用这些组件所引入的风险。SCA 工具可帮助开发人员检测并修复开发环境和 CI/CD 流水线中的问题。它们通常在代码提交到源代码库（SCR）后作为代码安全扫描工具来部署，因为 SCA 不需要在运行时扫描代码。通常情况下，SCA 工具具有以下优点:

● 持续扫描代码库(例如源代码、二进制文件、包管理器、清单、容器映像、无服务函数)，并创建/维护所使用的开源组件和第三方组件的清单。该库存可以作为软件物料清单(SBOM)的起点。CycloneDX和SPDX是两种主要的SBOM格式。

● 将软件BOM清单（主要是软件名、版本号）与国家漏洞数据库(NVD)等数据库进行比对，以识别是否存在漏洞。

● 提供关于组件许可要求的信息，包括归属权。

● 使治理团队能够制定一套适用于整个公司的开源和第三方组件使用策略。

● 在整个SDLC过程中自动执行策略（例如，当违反策略时构建失败，检查开发人员IDE中的漏洞，以及触发新开源或第三方组件的审批流程）。

● SCA解决方案是自动化的，并且可以作为开源或许可产品提供。开源SCA工具的一个例子是OWASP依赖项检查（OWASP Dependency Check）、OWASP依赖项跟踪（OWASP Dependency Track）。许可证信息由Synopsys、Snyk、Dependabot和Veracode等供应商提供。

5.5静态应用程序安全测试

静态应用安全测试（SAST）是一种分析源代码以发现安全缺陷和漏洞的方法和工具。SAST 也被称为 “白盒”安全测试。SAST 帮助开发人员识别编码阶段的漏洞，为他们提供实时反馈，使他们能够在这些漏洞进入

SDLC 的下一个阶段之前修复问题。NIST 源代码安全分析工具功能规范 v1.1[31] 规定了 SAST 工具必须检测的源代码漏洞类别应包括：输入验证、范围错误、API 滥用、安全特征、时间和状态、代码质量和封装。

由于 SAST 工具被设计在编译之前扫描源代码，因此它提供了更早执行安全扫描的机会，即安全左移。SAST 扫描既可以在代码提交到 SCR 时手动启动，最常见的情况下也可以在自动流水线操作时触发。建立一个良好的 SAST 流程包括:

10. 在相应的开发环境中明确SAST工具，这些工具应该支持开发环境的编程语言、库/框架和二进制文件。它必须能够检测OWASP 的前十位 和/或NIST 规范附件A所中列举的漏洞。

11. 在各自的开发组织中部署SAST工具，作为基础设施集成在IDE和CI/CD流水线中以便访问和使用。

12. SAST扫描可以在IDE中运行，也可以在CI过程的不同阶段（例如，预编码check-ins、post-commit）运行。

13. 分析扫描结果以消除误报。如果在CI/CD流水线中检测到这些问题，则应停止下一步执行，并将这些问题记录在中央存储库中。

14. 执行快速修复变更并管理开发团队对更耗时的变更的补救。

SAST 工具可以很好地扩展；它们可以针对源代码重复执行（例如，check-ins、nightly builds）。他们可以通过反馈来识别大量的源代码漏洞，通过在 SDLC 中更早地捕获漏洞，可以节省时间和精力。这些工具可以指出源代码中漏洞的位置，并指导其修复问题。SAST 工具也有可能导致许多误报，即该工具在不存在缺陷的情况下报告缺陷。通过分析测试结果和调整工具的策略/配置可以改善误报率。

虽然 SAST 工具是有效的，但它们难以应对某些类别的漏洞(例如，身份验证和访问控制问题)，特别是与环境、配置和运行时相关的问题。这可以通过采用动态应用安全测试(DAST)来补救，因为 SAST 和 DAST 是相辅相成的。参见**图 13**。

SAST vs. DAST

静态应用程序安全测试（SAST）和动态应用程序安全测试（DAST）都是测试安全漏洞的方法，但它们的用法截然不同。以下是两者之间的一些主要区别：

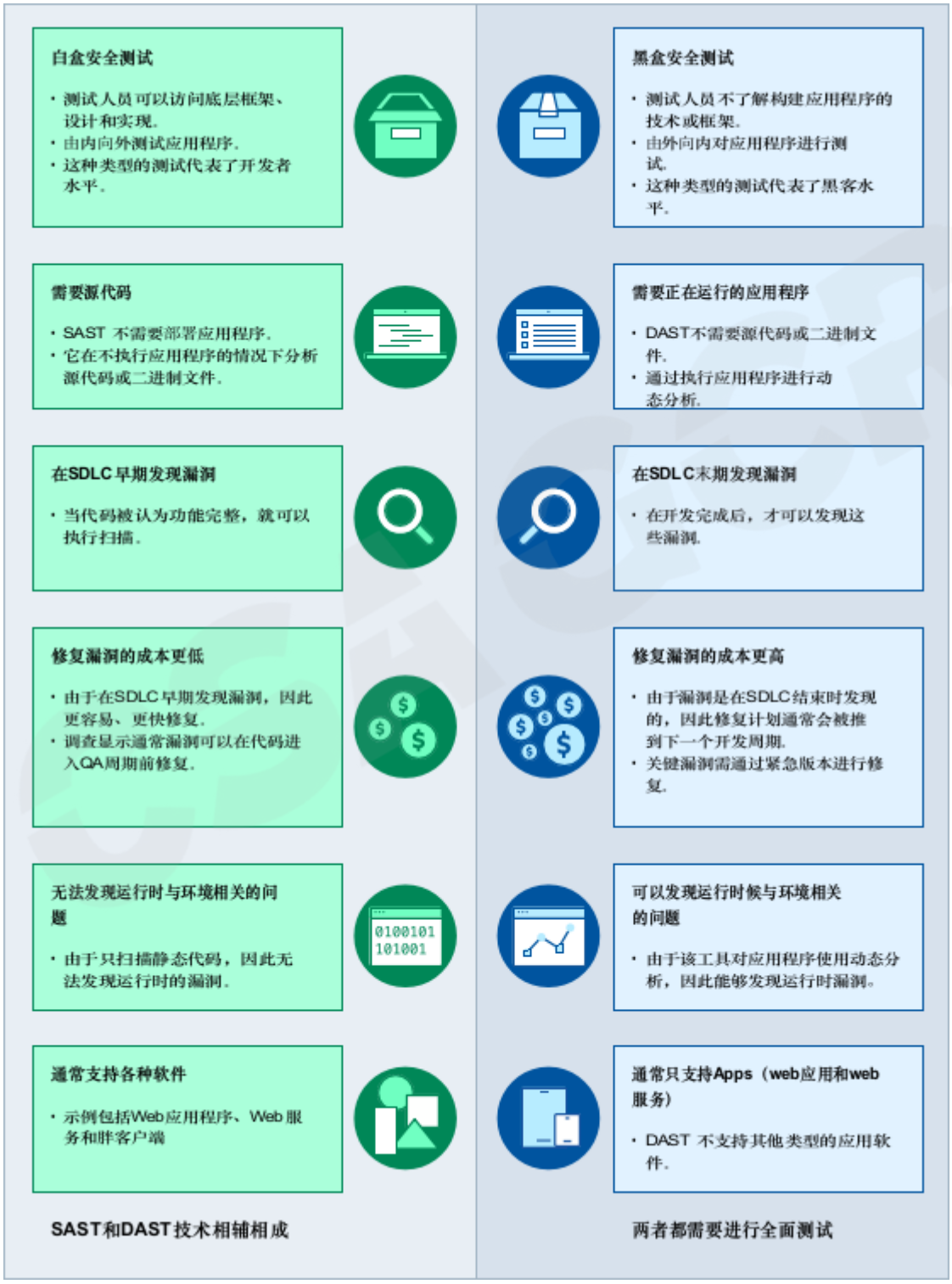

图 13：SAST和DAST的对比

虽然 SAST 可以由安全工程师手动执行，但与自动化 SAST 解决方案相比，手动评审代码的工作量太大且无法保证质量。虽然 SAST 工具的选择取决于语言支持和误报量，但 Synopsys、Veracode 和 MicroFocus 是许可产品的流行示例。OWASP 社区还提供了业界推荐的开源和商业工具列表。

5.6 容器强化

容器化应用程序和微服务架构(每个微服务都被容器化)是一种常见的组合。这是因为容器为快速和可重复的开发提供了一个平台，并促进了相同基础设施内的依赖关系管理，减少了应用程序多租户影响和潜在依赖冲突。一般来说，容器成为启用 DevSecOps 工作流的关键。

容器可以理解为与计算机中其他进程隔离的进程[33]。容器隔离与虚拟机及其管理程序提供的隔离不同，理解这一点很重要。因此，在某些情况下，容器化应用比虚拟机的攻击面可能会更大。因此，容器安全与虚拟化安全的处理方式会有很大区别

每当使用容器时，考虑容器编排的概念非常重要（例如 Kubernetes[34]、Nomad[35]、OpenShift[36]）。这种类型的软件允许控制（或编排）数百个容器，从而使配置、资源调度、网络、隔离和安全配置在某种程度上被集中起来，变得更容易管理。然而，使用编排器通常会在系统中引入额外的复杂层，这需要进行管理并考虑其安全性。安全考虑因素反映了需要加固的容器区域，以下是容器加固的安全原则:[37]

- **最小权限：** 应限定容器运行时所需的最小化权限集。
- **纵深防御：** 当应用纵深防御时，容器会成为一个额外的附加层。
- **减少攻击面：** 必须考虑容器编排层内的平衡和微服务架构的简单性。这是因为容器化微服务体系结构可以提高简单性。诸如Kubernetes这样的容器编排器增加了一层复杂性。

● **限制影响范围：** 确保被破坏（入侵）的容器不会影响到其他系统部件，这对于限制影响范围至关重要。

● **职责分离：** 需要根据特定的职责分离策略授予权限和凭证，这意味着骇掉一个容器不会让黑客获得系统内其他的资源和秘密。

有几个容器加固的指南。例如 CIS Docker 基准[38]，美国国防部容器加固程序[39]和美国国家安全局 Kubernetes 强化指南。[40]基于此，我们确定了以下常见的重点关注方面：容器隔离、容器网络、容器镜像安全和容器编排安全性。

容器隔离：确保容器正确隔离，对于确保容器不会危害主机(承载容器的机器)至关重要。这意味着要考虑以下实际因素:

● 特权容器拥有对主机中所有设备的root访问权限。这意味着如果您是容器中的root用户，那么您实际上是主机中的root访问权限。因此应谨慎对待特权容器。

● 功能[41]允许Linux对进程在系统中执行某些操作所需的不同权限进行细粒度控制。容器通常被分配了比他们需要的更多的功能。因此请确保您使用的容器时仅具有所需的已启用功能。

● 已挂载得卷允许容器访问主机内的存储。但是，挂载某些卷(例如“/”卷可以允许容器完全访问主机文件系统。

● 系统调用提供内核和应用程序之间的接口。但是,限制容器可以执行的系统调用是非常重要的。一个可以用于此操作的工具是Seccomp。[42]

● 还应该控制容器内的文件访问权限，以确保不利用高特权的二进制文件来提升权限。这样做的方法包括AppArmor43和SELinux。44

● 如果需要更多的隔离需求，开发人员可能也会感兴趣地研究KataContainers45和Firecracker.46

容器网络： 容器通常需要相互交互，这是微服务的本质。因此确保正确管理网络隔离至关重要。使用 Web 应用程序防火墙、路由表、服务网格和网络策略（在容器编排器中）可以确保不应该相互通信的服务是完全隔离的。

镜像安全： 容器通常需要相互交互，这是微服务的本质。因此确保正确管理网络隔离至关重要。使用 Web 应用程序防火墙、路由表、服务网格和网络策略（在容器编排器中）可以确保不应该相互通信的服务是完全隔离的:

- **Dockerfile 的安全性和构建过程：** 如果使用Dockerfile从头开始构建映像，那么考虑以下建议非常重要:

- 使用受信任的注册表(例如专用Azure容器注册表或弹性容器注册表)。
- 保持应用程序的安全要求要求在基本映像中使用标记或摘要的记忆。
- 使用多阶段构建。[47]
- 使用rootless容器。
- 对Dockerfile编辑访问控制。
- 检查卷的挂载，以避免挂载敏感的目录。
- 不要在Dockerfile中包含任何敏感数据。
- 避免使用SETUID二进制文件，因为它们可能允许攻击者提升容器内的权限。
- 避免在Dockerfile中使用不必要的代码。
- 确保对Docker守护进程的访问受到控制，并且不是每个开发人员都可以访问构建机器。
- 对镜像使用无守护进程构建: buildah,[48] podman,[49] and buildkit,[50]。

镜像扫描和签名： 镜像扫描旨在捕捉应用程序级别的漏洞和恶意代码，这些漏

洞和代码可能已在基础的镜像中引入。然后的目标是定期扫描镜像并对其进行签名，以确保在扫描和构建后不允许进行未经授权的变更。一些良好做法包括:

- 运行私有注册表。
- 使用Notary[51]或Harbor[52]等工具对镜像进行签名。
- 定期扫描镜像以查找应用程序级漏洞或恶意软件包。

容器编排安全性：容器编排安全主题非常广泛。因为它包括编排器底层基础设施的安全性，以及它管理和编排的软件定义的基础架构层的安全性。因此，我们的建议是使用完整的框架，如 Kubernetes[53] 的 CIS 基准或美国国家安全局加固指南，以获得系统的指导。一些一般性建议可以帮助保护诸如Kubernetes之类的编排器。确保:

- 允许控制器(例如OPAGatekeeper[54])强制在集群中创建对象。
- 所有帐户都正确定义RBAC，并且在不需要时不授予“星号”权限。
- 除非必要，否则避免节点端口暴露在公共互联网上。
- 集群中部署的所有容器都遵循上述建议。
- 所有使用的图像都遵循上述建议。
- 使用Istio[55]或Linkerd[56]等服务网格用于保证在容器之间传输敏感信息时，验证微服务之间的双向TLS身份。
- 控制平面与数据平面隔离，并且不允许在控制平面节点部署应用程序容器。

5.7 基础设施即代码分析

基础设施即代码(IaC)是通过代码管理基础设施，使用与开发人员相同的

工具，并保持基础设施配置和解除配置的一致性和可重复性。IaC 配置文件包含基础结构规范，可确保编辑和分发配置时的可用性和一致性。IaC 是 DevOps 实践的关键部分。它允许脚本自动构建和变更基础设施，从而减少了开发人员和 DevOps 工程师构建基础设施所需的工作。它使工程师能够在执行安全验证检查的同时，对云基础设施进行版本控制、部署和改进。这也是一个主动改善云基础设施状况并减轻安全和运营团队负担的机会。IaC 主要有两种类型：脚本类型和声明类型:

- **脚本类型：**这些脚本，如Bash、Go、Python或任何其他语言，使用云提供商提供的不同客户端（SDKs）；例如您可以使用AWS CLI或Azure PowerShell编写云基础设施资源的脚本。这种类型脚本因为需要管理所操纵的资源的不同状态，并编写所有步骤来创建或更新所需的基础设施，所以需要更多的代码行。[57]

- **声明类型：**在这些语言中，以配置和属性的形式编写所需系统或基础结构的状态就足够了。例如，HashiCorp的 Terraform和Vagrant、Ansible Azure ARM 模板、PowerShell DSC、Puppet和Chef就是这样。用户只需编写所需基础设施的最终状态，工具就会负责应用它。这允许使用与应用程序代码相同的源代码控制工具(即GitOps) [58]以声明方式定义基础设施并对其进行版本控制。在建立声明状态后，“漂移”被建立为一种与所定义的期望状态不对应的状态。配置管理工具在历史上一直被用来强制执行从时间点黄金镜像派生的状态。

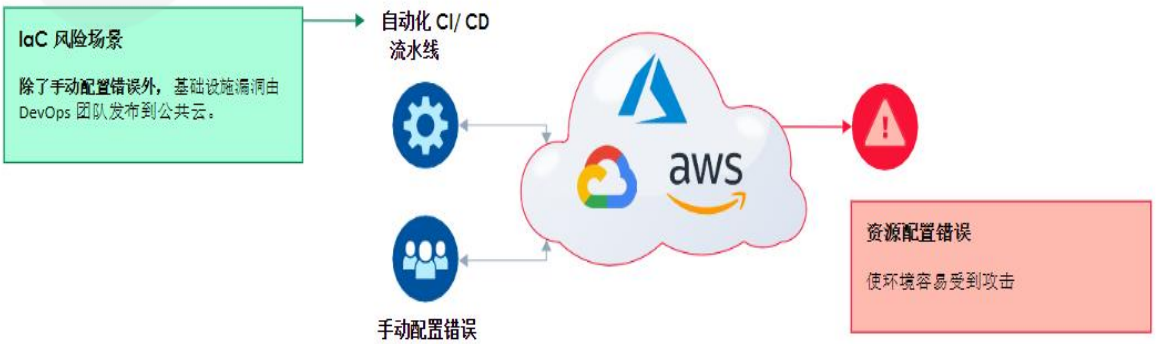

图 14:IaC 错误配置工作流程

始终如一地保持其完整性非常重要。在定义为代码的策略中定义一系列

允许的行为可以帮助检测“漂移”或偏离允许行为的所需配置。定义部署前安全配置基线检查是默认框架的安全基础。安全团队扩展多个应用程序的能力随着必要的速度而变得更加困难。

定义部署前配置安全检查是安全默认框架的基础。将安全策略定义为代码作为配置安全检查使安全性和合规性民主化。它创建了一个自助服务生态系统，工程师可以在其中安全地部署资源而不会无意中破坏服务或不安全地部署资源。在专用于分类、代码重构、跟踪和测试的时间之后重新部署资源可能需要几天或几周的时间或资源。基于 NIST[59] 的一项关于技术和测试不足的经济影响的研究,在基础设施即代码安全的范围内，定义安全策略是实现法规遵从性的前提。IaC 实践从作为代码的声明性基础设施进一步发展到所需状态将合规要求/控制编码为操作的策略作为代码(PaC)。它创建了一个自助服务生态系统，工程师可以在这里安全地部署资源而不会无意中破坏服务或不可靠地部署资源。在专用于分类、代码重构、跟踪和测试的时间之后重新部署资源可能需要几天或几周的时间或资源.

开放策略代理(OPA)--一种实现 PaC 的方法--一个策略引擎，可以自动化并统一云原生环境中的策略实现。组织使用 OPA 在所有相关组件中自动执行、监视和修正策略。OPA 可用于跨服务(如协调器、应用程序编程接口(API)网关和连续集成/连续交付(CI/CD)流水线)集中操作、安全和法规遵从性。OPA 可以通过提供强制执行策略的功能来支持创建、部署和管理 IaC。一些例子包括 [60]:

- 应用程序授权: OPA 使用声明型策略语言，可以创建和执行规则，包括为租户提供策略的权限。
- Kubernetes 准入控制: Kubernetes 具有一个内置功能，提供了强制执行准入控制策略的功能。OPA 通过将容器映像指向公司映像注册表、向 sidecar 容器中注入 pod、向资源添加特定注释来帮助扩展这些功能。

- 服务网格授权：OPA 可以通过将授权策略直接添加到服务网格中来帮助规范和控制服务网格架构。这有助于限制微服务架构中的横向移动并强制执行合规性法规。

在构建阶段，对Terraform等工具进行代码扫描(例如IaC分析)是有效的。反过来，它可以降低漏洞和安全弱点被写入代码的可能性。然而，必须理解的是，与部署后的扫描(例如云态势管理)相比，构建时的扫描并不总是能产生一套完整而全面的结果。 Tfsec,Kics, Checkov, Snyk 和 Terrascan 都是用于Terraform部署的IaC扫描工具的示例。一旦建立了安全基线，就会根据单独的策略引擎中定义的策略，部署基础设施或云资源。每个策略都要求IaC资源定义符合安全基线（请参阅Guardrails）。

尽管在基础设施构建期间建议在“编码”中进行IaC分析，但IaC安全工具在预部署和运行时等附加阶段使用时也可以提供优势。

- **预部署：**在使用VM或容器黄金映像之前，配置安全检查可以作为对云资源/服务、规模集模板的强化检查。在定期部署之前，测试Iac和安全配置非常重要。诸如Conftest[61]之类的开源工具可以用于Terraform代码、无服务器配置、Kubernetes配置或CI/CD流水线定义的结构化检查或测试。一旦在策略代码中声明了所需的状态，就可以定期在部署的云资源中检测到回归。

- **运行时：**然后，定义策略可以在云中运行的资源之间同步进行。在运行工作负载时，可以使用诸如OPA或Kyverno(专门针对Kubernetes安全环境)之类的策略工具来定义和处理策略。

安全单元测试

安全单元测试是对安全相关功能的非功能行为进行细粒度验证。安全单

元测试的实践包括单元测试和集成测试，大多数自动化测试框架都可以用于编写和执行这项安全测试。开发团队创建这些测试，用于在实现和测试阶段评估不同级别的应用程序行为。

单元测试旨在测试代码中独立组件的行为，包括类或方法。从开发人员的角度来看，安全单元测试的目标是验证代码是否符合安全编码要求。另一方面集成测试的目的是为了验证多个单元或组件之间的相互作用。对于安全集成测试，开发人员在与其他软件组件集成之前先验证自己的代码组件如函数、方法、类、API 和库。

单元测试的一种实用方法是将重点放在容易出现安全漏洞的软件组件上作为通用安全测试套件的一部分。

- **OWASP 安全测试指南** [62] 描述了一种有效的安全单元测试方法，该方法基于通用安全测试套件作为更广泛的单元测试框架的一部分。使用这种方法，开发人员可以构建从通用安全测试套件派生的安全测试用例集合。
- **InSpec**：一个可用于创建通用安全单元和集成框架的云基础设施和系统配置的测试是 InSpec[63] ，这是一个开源的基础设施测试框架，使用人和机器可读的语言来提出框架的合规性、安全性和策略要求。威胁建模还有助于识别通用测试套件的使用和误用情况。测试套件可以包括针对应用程序安全控制和 OWASP 应用程序安全验证标准等标准要求的相关测试用例。用于测试的软件组件和相关函数、方法和类通常包括 OWASP、Web 安全测试指南(WSTG)中涵盖的以下领域:

  i. **身份、身份验证和访问控制：** 身份验证、身份和访问控制是安全单元测试中可以确保访问控制中断以及识别和身份验证失败等漏洞的软件组件。身份验证的安全单元测试包括由于验证和修改不足而导致的与令牌滥用相关的故障或缺陷的测试。

  ii. **输入验证和编码：** 对于输入验证和编码，组件包括验证和清理用户提供的输入的类和方法，如用户输入表单和用户请求参数。接受用户输入

的任何组件都可能是代码注入或诸如 SQL 或跨站脚本之类的代码注入存在潜在入口路径。

iii. **加密：**用于加密的软件组件包括用于传输加密数据的组件。与加密相关的单元测试包括编码技术加密库和哈希算法的验证。这些测试包括针对弱加密控制、敏感信息的明文传输，以及证书和文件哈希值不正确的签名验证等攻击案例。

iv. **用户和会话管理：安全单元案例**：该领域的单元和集成测试包括对帐户枚举和暴力强制预防的检查。对于会话管理单元测试涵盖了每个用户在多个请求期间的会话状态和访问权限。这有助于防止访问控制中断和安全配置错误。

v. **错误和异常处理：**单元测试有关错误或异常处理检查的正确代码状态是返回的错误或错误信息不包含敏感信息。验证此类信息是否有助于防止意外故障，或验证潜在信息披露是否有可能被进一步利用。

vi. **审核和日志记录：**此类别包括与审核和日志功能相关的软件组件。对于日志记录，要测试验证日志不包括敏感信息，并且格式和编码正确。这些测试还要验证是否未记录诸如失败登录和高价值事务之类的可审核事件。这些测试降低了安全日志记录和监控失败的风险。

安全单元测试可能会带来几个好处。特别的优势是方便易用、优于扫描工具以及交付速度。[64]

- **更易于使用：** 主要好处是继承了现有的标准实践和基础条件，可有效改善软件组件的安全状况，适应极佳的同时，最大限度降低对开发人员现有工作方式的干扰。 因为单元测试不需要额外的工具，开发人员通过运行和更新测试，即可完成部分手动或自动的安全测试。 此外，单元测试也具备功能测试的优点，开发人员可以独立、持续地测试小型组件，这对于大多数开发人员来说不难，一般研发团队都具备拥有运行单元和集成测试所需的工具、技能和基础条件。

- **相对于扫描工具的优点：** 它产生假阴性的误报更少，测试用例的准确性更高。

能在创建单元和集成测试时考虑开发人员的意图，降低成本投入。 与扫描工具一样，开发人员可以通过单元测试来发现安全漏洞及问题，以便开展进一步更详细的脆弱性分析，从而便于开发人员修复程序直至漏洞不可被利用。

- **提升交付速度：** 安全的单元和集成测试将安全动作左移，使开发人员能够直接在代码层面复现安全问题，提升发现检测的效率，降低修复成本。 这有利于增加安全测试覆盖范围和测试需求的可追溯性。 测试用例可识别源代码中具有根本原因的潜在安全问题。 它还允许尽早并重复地检测安全缺陷和漏洞，作为通用测试质量保证实践的一部分。

5.8 同行评审

虽然经常被忽视，但在向存储库主分支提交变更之前，强烈建议进行同行评审和验证。鉴于，人类的时间有限且精力宝贵，应至少在评估跨信任边界的“入口点”和攻击面等关键时点，对源代码及设计的变更进行同行评审。（请参阅威胁建模）。

为了对关键代码组件实施同行评审，大多数代码存储库软件（例如 GitHub 和 Gitlab）都提供分支保护功能。例如，通过在项目主分支上启用分支保护，可以设定:

- 合并前需要评审拉取请求
- 合并前需要一定数量的批准
- 推送新提交时驳回过时的拉取请求批准
- 合并前需要通过状态检查
- 要求分支在合并前保持最新状态

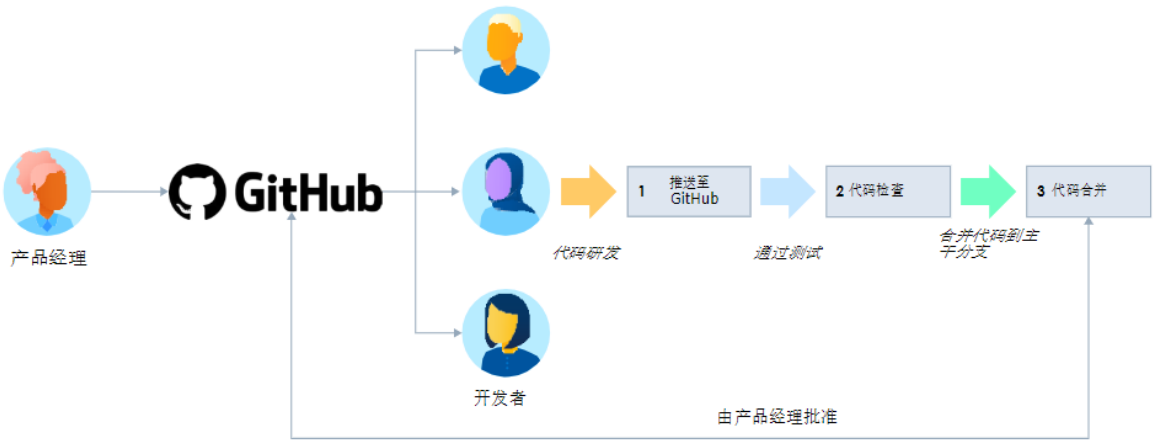

图 15：源代码同行评审

6 集成与测试

集成和测试包括对应用程序/产品的功能进行安全测试的工具和过程。如果没有这些工具和方法，安全漏洞和弱点将被利用并导致数据泄露和可用性等问题。

6.1 动态应用程序安全测试

动态应用程序安全测试 (DAST) 是一种测试方法，可分析正在运行的应用程序，通过模拟针对应用程序外部接口的外部攻击来识别安全漏洞。与 SAST 不同，DAST 工具将在不了解应用程序源代码、内部工作原理或设计的情况下扫描应用程序。 因此，它被称为“黑盒”测试。DAST 工具通常用于测试 Web 应用程序和服务，但某些解决方案可以涵盖 API 扫描、渗透测试和模糊测试。65

DAST 可以解决的最常见应用程序漏洞包括：跨站脚本、SQL 注入、不安全的服务器配置、路径泄露、拒绝服务、代码执行、内存损坏、PHP 注入、JavaScript 注入、本地和远程文件包含以及缓冲区溢出。66DAST 工具也可以被视为是包含在模糊测试中的一种测试技术。

DAST 工具独立于应用程序运行，往往可以作为测试套件的一部分持续运行。伴随新漏洞的产生及发现，公司可以在漏洞被利用之前及时发现并修补漏洞。 DAST 工具通常无关于编程语言，因为它不依赖于源代码。由于 DAST 工具无法访问源代码，因此它们无法指出代码库中漏洞的位置，导致报告的解释较为耗时。测试人员必须充分了解应用程序及其攻击面，以确保正确配置 DAST 工具。DAST 工具在 CICD 流水线中作为集成和测试阶段的一部分运行，其位置偏后，这使得修复漏洞的时间及精力成本更高。建立良好的 DAST 流程通常涉及以下部分:

确定适合您环境的 DAST 工具。Web 应用程序漏洞扫描程序评估项目(WAVSEP)[67] 和 OWASP 基线项目 [68] 有助于建立 DAST 工具的选择标准及评估。选择标准需要包括以下内容 [69]:

- 误报及漏报率低
- 可以自动生成包含高级摘要和技术细节的详细报告
- 模拟用户操作路径的能力
- 能够支持一系列脚本/编程语言、框架和所有 Web 应用程序
- 可以识别此前未知或未公开的缺陷，如 0day 漏洞
- 可以自动化的重复运行
- 了解应用程序中的用户工作流程。与软件开发人员和解决方案架构师协作沟通，了解及记录他们与应用程序的交互及操作。
- 自动化用户交互的脚本，这些脚本作为集成和测试阶段的一部分合并到 CICD 流水线中。
- 及时识别并修复问题和漏洞以防被利用。

与 SAST 工具相比，DAST 工具产生的误报更少，因为它模拟真实场景 70。 此外，与 SAST 工具不同，DAST 工具可以检测运行时及运行环境相关问题，这需要应用程序正常运行 [70]。Acunetix、StackHawk 和

OWASP ZAP 是 DAST 测试工具的典型代表。

6.2交互式应用程序安全测试

交互式应用程序安全测试（IAST）是一种结合了 SAST 和 DAST 技术的方法。与 SAST 一样，它可以识别到有问题的代码行以帮助开发人员进行精准修复；同时，IAST 还提供动态输入检测，因为 IAST 结合了 SAST 和 DAST 的能力。IAST 与 DAST 的不同之处在于它在被测系统环境中运行，因此，在部署前必须将其集成到代码库中。IAST 可以在开发期间和发布后检测漏洞，IAST 测试通常使用托管在已编译代码库和被测 Web 服务器中的代理来实现；在编译代码时，检测机制会扫描源代码中的编码错误。

与 DAST 一样，IAST 是一种在应用程序运行时进行的安全测试，通常在 QA 或测试环境中进行。IAST 在应用程序代码库中包含扫描代理和软件库，代理监视应用程序如何响应测试。IAST 使用内部应用程序流、流量和分析工具来运行测试，该工具可以执行静态源代码可见性和动态测试。代理可以查看已编译的代码，包括流量请求、第三方库以及与操作系统的通信。

在代码库中编译的检测器分析应用程序如何响应输入和单元测试。应用程序服务器在运行时托管检测组件。 IAST 以主动或被动扫描模式运行:

被动模式下: 与 DAST 工具不同，IAST 不会主动分析流量，而是使用合法流量进行分析，从而减少误报。 在被动扫描期间，IAST 运行时代理会扫描应用程序的所有传入流量并检测数据流。 根据应用程序的响应方式，IAST 工具可以通过查看 URL 和包含的请求参数来识别源代码中的漏洞。建议 IAST 在单元测试阶段启用被动交互式分析，允许该工具充当爬虫并检测易受攻击的第三方组件。

被动扫描的优点是它分析由单元测试生成的流量。另一方面，被动扫描依赖于实时流量和单元测试生成的流量，并且仅提供源代码的有限测试覆盖范围。因此，该工具可能会错过此流量之外的漏洞，尤其是当用户或测试人

员仅测试一组有限的功能时。

主动模式下：主动的 IAST 在后台使用 DAST 扫描器来创建流量并执行 IAST 工具可以检测到的主动扫描技术。 在主动 IAST 运行期间检测到的漏洞较少，因为只有来自 DAST 扫描的流量在范围内。在主动运行期间，IAST 只能检测由 DAST 扫描触发的漏洞。另一方面，这也可能导致缺乏对漏洞的检测，特别是当应用程序修改了 DAST 工具的请求或被测试的代码段时。

建议使用测试环境来执行大部分测试，以检测更多漏洞。交互式分析的优点在于它结合了 SAST 和 DAST 的功能，并且可以根据需要替换其中一种工具。 基于代理的方法允许在执行过程中跨不同环境（例如开发和生产）检测漏洞。

由于动态和静态测试的综合能力，IAST 工具还为 OWASP Top 10[71] 提供了更广泛的安全测试覆盖范围。交互式分析还可以分析实际执行情况，由此获得在运行时执行更准确的安全分析。

与 DAST 相比，交互式分析在识别和映射代码的易受攻击部分方面更加准确，并可能需要更少的精力来修复发现的问题。交互式分析有助于克服 DAST 工具的挑战，因为它有助于识别和映射 DAST 工具测试的代码段，并且修复工作量更少。

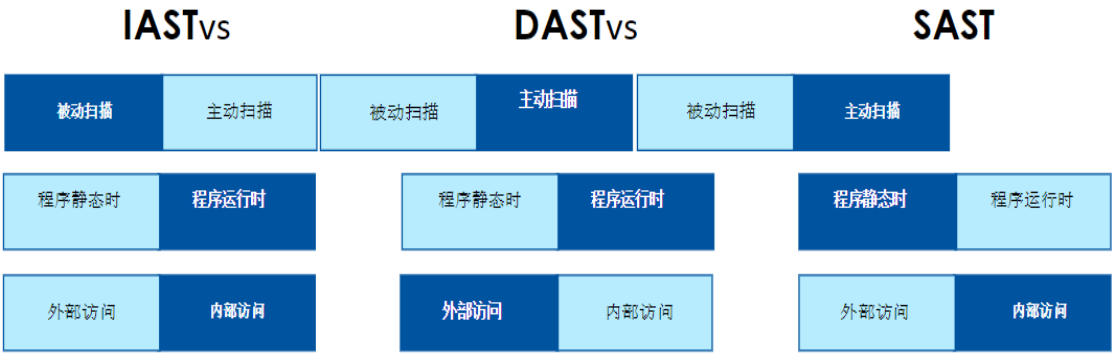

图 16：IAST 、DAST 、 SAST 的区别比较

IAST 工具大多是专有的或商业化的，也存在少量社区开源版本。开源的可用 IAST 项目，如： Contrast Community Edition，它为 Java 和 .NET Core 提供 IAST 功能。商业化的 IAST 产品可通过 Contrast、Checkmarx、Synopsys 的 Seeker 和 HCL 的 ASoC 等供应商处获取。 正如本文档中提到的，选择 SAST 和 DAST 的重要关注点应聚焦于语言支持和误报量等因素指标。

6.3 API 测试

随着企业将越来越多任务和功能打包为 API 产品或属性，应用程序编程接口 (API) 支持的数字生态系统正在激增。 金融服务行业的一个很好的例子是开放银行。 API 有多种形式，可能是内部的、外部的、REST、SOAP 或某种形式的过程调用。

安全性是一个成功 API 的关键组成部分。 API 面临数据、交易和服务等多方面风险，不安全的 API 可能导致敏感数据和 IP 外泄， 进一步因而受到损害。 常见的 API 漏洞包括缺乏有效的身份验证和加密措施、客户端凭证密钥受损、不安全的存储资产保护（如 AWS S3 存储桶/文件夹）以及 XSS、SQLi 和 DDoS 等通过 API 注入的人为干预攻击。

了解 API 生命周期有助于分解需要保护的内容，包括识别所使用的 API 并开展有针对性的漏洞测试。可参考 OWASP Top 10 API 漏洞指南对可利用的漏洞进行识别及测试:

(一) 被破坏的对象级授权：API 宽泛的对象标识符导致了广泛的攻击面，因此在使用用户所输入的访问数据源的每个函数中都应考虑对象级授权的检查。

(二) 被破坏的用户身份验证：身份验证实施不当会导致攻击者破坏身份验证令牌，从而损害系统识别客户端/用户的能力。

(三) 过度的数据暴露：对象属性通常在未被考虑其敏感性的情况下被

不当暴露。

(四) 缺乏资源和速率限制：对客户端/用户请求的资源大小或数量缺乏限制可能会影响 API 服务器性能，并进一步导致拒绝服务 (DoS) 和暴力攻击。

(五) 被破坏的功能级授权：复杂访问控制策略往往具有不同组和角色，若授权不当，管理功能和常规功能之间未明确分离，管理功能可能因此遭受攻击者的访问。

(六) 批量赋值：如果没有基于白名单的适当过滤，通常会导致批量赋值，从而允许攻击者修改他们不应该修改的对象属性。

(七) 安全配置错误：不安全的默认配置、不完整的或临时配置、开放的云存储、错误配置的 HTTP 标头、不必要的 HTTP 方法、允许的跨源资源共享 (CORS) 以及包含敏感信息的详细错误消息。

(八) 注入：当不受信任的数据作为命令或查询的一部分发送到解释器以欺骗解释器执行非预期命令时，就会发生注入缺陷，例如 SQL、NoSQL 和命令注入。

(九) 不适当的资产管理：适当的主机和部署的 API 版本清单对于缓解诸如已弃用的 API 版本和暴露的调试端点等问题发挥着重要作用。

(十) 日志记录和监控不足：这使得攻击者能够持久的保留后门，并横向在更多系统间展开篡改、提取或破坏数据等危险动作。

安全 API 网关是需要添加保护层的组织的可用选择。这类外围网关产品可提供一系列安全服务，包括 API 授权和身份验证、日志记录和监控等功能。 API 网关可以用与任何其他端点相同的方式进行测试，以进行进入公司网络的身份验证。 如果是高端的网关产品，它会采取与防火墙类似地方式运行，并较少的暴露可用的端口和服务。

几乎所有云提供商都提供 API 网关并可能执行一些身份验证，即便它们的安全功能往往有限。

API 安全测试有助于构建更安全、增强的 API 服务，以下常见的设计注意事项可以帮助进一步保护您对 API 的使用:

- **使用安全策略：** 应用 API，例如 IP 地址的速率限制、配额限制来控制 API 使用以及 API 密钥验证。
- **机器人检测：** 监控流量和识别非法请求的服务。
- **有效负载验证：** 对于入站流量，对 XML 和 JSON 等语言应用输入和请求验证。

可以使用工具来进行 API 的功能和漏洞测试，此类工具可以构成单元测试、模糊测试的一部分，或使用 DAST 进行运行时测试时的解决方案。一些 DAST 供应商通过输入恶意内容来开展 API 测试。 以下是一些可以使用特定于 API 测试的工具，例如 BurpSuite、Postman 和 Hoppscotch。

6.4 模糊测试

模糊测试[72] 是一种软件测试方法，它将格式错误/意外的输入注入系统以揭示软件缺陷和漏洞。模糊测试工具将这些输入注入系统，然后监测异常情况，如崩溃或信息泄漏，它可用于测试运行时和 API 服务中的应用程序。模糊测试最常见的形式有:

- **基于变换的模糊测试** 它对现有的数据样本进行变换，以创建测试数据。
- **基于生成的模糊测试** 在输入模型的基础上产生新的测试数据。

这种测试方法可以发现编程错误，如内存泄漏或缓冲区溢出，这些错误很少能被开发人员常规测试，如单元、集成或系统测试所发现。攻击者经常以这些错误为目标，破坏生产系统或获得对生产系统的访问。例如，密码学 OpenSSL 上的心脏滴血（Heartbleed）漏洞就可以通过模糊测试发现。通

过模糊测试通常可以发现的错误有:

- **C/C++特有的错误：**释放后重用（UAF）、缓冲区溢出、未初始化的内存和内存泄漏
- **算术错误：**除零错误、整数/浮点数溢出和无效的按位移位
- **普通崩溃**：NULL引用和异常
- **并发错误**：数据竞争和死锁
- **资源使用错误**：内存耗尽、挂起或无限循环以及无限递归（堆栈溢出）
- **逻辑错误**：同一协议的两个实现之间的差异（见示例），往返一致性错误（例如，压缩输入，解压缩回；与原始文件进行比较），以及断言失败模糊测试在使用不可信或复杂输入的软件项目中特别有用，例如文件解析器、媒体编解码器、网络协议、数据压缩、代码编译器和解释器、数据库、浏览器、文本编辑器或任何其他类型的用户界面。模糊测试为目标系统和软件的质量提供了一个良好的全貌。使用模糊测试，您可以轻松地评估系统和软件的健壮性和测试中的安全风险态势。

模糊测试通常是自动化的，可以使用 Clusterfuzz、WFuzz、OSS-Fuzz 和 OWASP ZAP 等开源工具进行。Beyond Security 和 Synopsys 等供应商是提供模糊测试的商业产品代表。

6.5渗透测试

渗透测试是一种技术评估，它模拟网络攻击场景，以评估一个组织的信息安全能力的成熟度。这些场景针对的是网络、系统和应用组件中的缺陷或弱点。识别恶意行为者是否可以利用这些弱点来破坏环境中的资源。渗透测试将技术和工具集配置在一个有限的操作范围之内，以最大限度地减少对组织的业务和运营的干扰。

渗透测试前，共享的信息量可能会影响评估的结果。有三种方法被广泛

用于进行这种类型的测试，每种方法都具有不同的目标和攻击模拟:

黑盒:模拟由无特权用户或外部人员进行攻击的测试方法。这种方法假定测试人员在评估过程中没有得到系统、网络或环境中的资源的先验知识。

灰盒:模拟由无特权用户或内部人员进行攻击的测试方法。这种方法假定了解环境的系统、网络或资源。或者作为评估的一部分，与测试人员共享了有限的信息，如凭证。

白盒:模拟由特权用户或内部人员进行攻击的测试方法。这种方法的前提是，作为评估的一部分，向测试人员提供系统、网络和环境中的资源的详细和大量实质性知识。

DevSecOps 原则加强了持续进行这些评估的必要性，以适应持续发展和不断变革的步伐。

然而，重要的是要仔细规划您的方法和目标领域，而不受合同约定条款的限制。[73] 对于托管在云中的资源，尤其需要进行规划，鉴于责任共担模型，您必须与云服务提供商（CSP）协调测试工作。CSA 云渗透测试手册提供了额外的指导和测试方法。

图 17：责任共担模型

6.5.1 渗透测试范围

云托管系统的共担所有权概念会很大地影响渗透测试评估期间的可执行活动的范围。共担所有权因采用的云服务模式（如 SaaS、IaaS、PaaS）

和属于云服务提供商（CSP）责任的安全控制而有所不同，凡是属于云服务提供商的安全控制，通常不在测试评估范围内。

- **软件即服务 (SaaS)**：软件即服务(SaaS)环境的测试范围是最小的。然而，只要排除对底层基础设施的任何测试，并获得云服务提供商的明确许可，测试人员就可以评估数据或用户访问控制，并利用过度的用户权限。

- **软件即服务 (PaaS)**：平台即服务(PaaS)环境的测试范围包括应用层，所有可用的接口，以及在SaaS部署模型所允许的基础上的一些平台配置。所有其他层都被排除在外。然而，与SaaS环境类似，必须获得云服务提供商的明确许可，特别是对于公共云部署模型。如果测试人员能够破坏访问权限，利用一个缺陷可能会影响其他租户或供应商。在这种情况下，重点是代码评审和白盒测试。

- **基础设施即服务 (IaaS)**：基础设施即服务(IaaS)环境的测试范围包括在PaaS和SaaS部署模式所允许的基础上的操作系统层。这个范围包括测试用于创建虚拟机的任何金标镜像，利用未修补的漏洞，以及滥用开放的远程管理协议来获得访问权。一般使用SSH和RDP来访问或跳转到其他虚拟机。不过，通过与各自的云服务提供商协调，确认参与规则，是可以避免这种情况的。对于白盒测试，要注意您的IP地址范围，因为它们可能随着虚拟机的配置和取消配置而定期改变。

6.5.2 渗透测试的步骤和纵深防御

随着企业采用纵深防御策略来保护他们的数据和最关键的资产，攻击者必须绕过许多安全控制措施来获得对您环境中资源的访问。实施多层防御和应对措施是一种良好的做法，可以减少单点故障的可能性，并保护系统和数据的机密性、可用性和完整性。

图 18:防御的层级

为了实现您的安全目标并保护您的数据，在对您的云托管资源进行渗透测试时，请考虑以下步骤和目标：

侦察:从开源情报（OSINT）来源收集信息。要注意可能已被硬编码并存储在代码存储库（如 GitHub）中的凭证。社交媒体通常是一种可以用来提供便利的信息来源，可以用来促进复杂和有针对性的社会工程攻击。检查云管理员在领英或其他平台上分享的内容，以确认他们没有泄露可能被恶意行为者用来破坏您的环境的信息。

扫描:许多数据泄露事件都与公开暴露的存储容器有关。存储容器，将敏感的客户和专有信息暴露在互联网上，会造成品牌和财务损失。考虑扫描凭证和错误的云安全设置，特别是 AWS S3 存储桶、Azure Blob 存储或可能被设置为 "公共 "类型的类似存储。

利用:利用漏洞获得对系统或资源的访问，需要周密计划发动一次攻击，以确认可利用的缺陷。这些将受益于特定云的技术和身份验证攻击。对暴露的管理界面、API 控制台、基于 Web 元素和实例服务进行暴力攻击和会话攻击，以产生最大的价值。以凭证和加密密钥为目标，获得访问权限并绕过安全控制。

跳板:跳板是网络攻击成功的基础。在获得初始访问权之后、恶意行为者可能需要通过网络移动来访问敏感数据或高价值资产。要注意子网的无状态访问控制列表，这些列表在默认情况下可能是过度宽松或放任的，便于恶意行为者横向移动，因为它们允许资源与环境中的其他子网通信。

渗出:数据渗出通常是恶意行为者的主要目标。窃取或加密数据通常与经济利益有关，而泄露数据是为了损害声誉。在进行黑盒或红队评估时，要考虑将数据渗出到云中的存储容器里，以减少暴露敏感信息的风险，或降低将其存储在一个您的组织可能不可见、不受控制的环境中的风险。

6.5.3 渗透测试的益处、框架、指导和战术手册

渗透测试，这种类型的技术评估通常是监管合规的要求。它被用来评估系统和应用程序的弹性，并评估一个组织的安全政策的有效性、安全意识以及检测、预防或减少安全事件影响的能力。它是任何安全计划的重要组成部分，可以提高内部部署和云托管系统的可靠性、完整性和安全性.

被识别的漏洞作为渗透测试评估的一部分，需要对更严重的威胁进行鉴定和优先处理，以便以务实和具有成本效益的方式优先处理。在建议是否应该避免、补救、减轻、转移或接受风险之前，测试人员需要评估相关的影响。正如在威胁建模部分提到的，DREAD 模型可以用来评估计算机安全威胁的严重性。相比之下，STRIDE 模型可以用来对其进行分类和定义，并回答“会出什么问题？”的问题。考虑以下渗透测试指南和框架，以尽可能多地从这项活动中获得益处:

国家网络安全中心(NCSC)的安全开发和部署指南**：** 包含旨在建立一套工作实践的高层次原则，以嵌入安全并使代码更加稳定和更加易于维护。该指南还包括一系列的安全工具，并可以转化为自动测试套件，以评估作为部署流水线一部分的构建，允许恶意行为者持续使用的一些测试场景。NCSC 还制定了渗透测试指南，该指南定义了如何有效地使用渗透测试，以获得进行这一活动的最大收益。

MITRE 公司的常见攻击模式枚举和分类: 这本手册侧重于测试公共云环境中托管的系统和服务。它为公共云渗透测试方法提供了一个基础，并促进了对云安全联盟发布的《云计算关键重点领域安全指南》的研究，以支持管理和减轻与采用云计算技术相关的风险。

云安全联盟（CSA）的云渗透测试战术手册： 这本手册侧重于测试公共云环境中托管的系统和服务。它为公共云渗透测试方法提供了一个基础，并促进了对云安全联盟发布的《云计算关键重点领域安全指南》的研究，以支持管理和减轻与采用云计算技术相关的风险。

注册道德安全测试员委员会(CREST)的渗透测试方案： 该指南提供了实用的建议，以开展范围广泛、成本效益高的渗透测试活动。它旨在帮助企业通过三个阶段的计划进行持续测试：准备、测试和后续跟进。准备阶段详细说明了测试的动机、目的和目标。测试阶段建立了一个有效的测试流程、计划和活动管理提示。后续跟进阶段提供指导，以积极修复已发现的漏洞，并提供商定的行动计划。

渗透测试执行标准(PTES)的渗透测试执行标准： 该标准提供了一个涵盖七个部分的结构，即初始沟通、情报收集、威胁建模、漏洞分析、漏洞利用、后利用和报告。本指南抓住了整个渗透测试过程，以帮助更好地了解组织，并提供支持测试执行的指导。本标准可与 PTES 技术指南一起使用，以根据场景帮助确定正确的程序和工具。

美国国家标准与技术研究院(NIST)的信息安全测试和评估： 该指南提供了进行技术测试的检查方法和技术，作为信息安全评估的一部分。它提供了准备一个强大的计划过程、根本原因分析和报告的建议。它还包括对测试人员的执行过程的洞察，以确定、验证和评估系统可利用的弱点，并验证适当的安全控制。

6.5.4 在 DevSecOps 环境中需要考虑的特殊用例

在 DevSecOps 背景下进行渗透测试时，考虑特定的场景也很重要，从评估的角度来看，这些场景可能是有吸引力的。考虑以下场景，因为它们可能会被忽视或经常不被考虑:

- **攻击持续集成/持续交付(CI/CD)编排器：** 随着自动化和使用CI/CD编排器，必须考虑对编排器本身进行渗透测试。测试的范围将是编排器本身和源代码控制应用程序。这类测试的目标如下:

确认CI/CD编排器中的访问控制，不允许可能会部署恶意基础设施的未授权作业/运行。

- 验证无法通过在编排器中运行恶意作业来提取机密。
- 确认分支策略不允许未经授权的提交和推送。
- 评估编排器内部的漏洞。

● **攻击容器编排器：**当对容器编排器进行测试时，测试的范围应包括编排器中的所有容器，并且该测试的最终目标应至少是执行容器逃逸或提取机密和敏感信息。

● **攻击云基础设施：** 在攻击云基础设施时，必须考虑到虽然提权到管理员的特权权限可以是一个目标，但考虑到云基础设施的特点，泄露敏感数据、寻找公共存储或从外部账户创建持久性也是需要考虑的变量。因此，相应地商定渗透测试的目标是很重要的。

6.5.5 渗透测试工具

众多的工具和自动化引擎可以支持识别和分析计算机系统的安全弱点。将 OWASP Top 10 的控制措施作为渗透测试评估的一部分，以保护您的系统免受最关键的安全风险的影响。云安全联盟的《云计算的顶级威胁》提供了一份相关的云安全威胁及其直接业务影响的清单。

考虑使用以下一些工具来自动化测试您的活动，并针对最常见和最严重的安全弱点评估您的 DevSecOps 计划:

● bucket_finder - 暴力破解工具，扫描Amazon S3领域以确定存储桶的名称是否存在，并标识它是公共的、私有的还是重定向的。

● Chef InSpec - 开源工具，用于自动化安全测试，并根据Chef市场的自定义或预定义测试验证合规性，以审核您的应用程序、依赖关系和基础设施。

- Gauntlt - 自动攻击平台，能够创建可操作的测试，可以连接到您的部署流水线中，使用通用渗透测试工具评估您的代码。

- pacu - 开源的AWS开发框架，旨在进行进攻性安全测试，利用AWS环境中的配置缺陷和漏洞。

- MicroBurst - 通过评估和利用薄弱配置来枚举服务、资源和凭证，用于评估Azure环境安全性的脚本集合。

- OWASP OWTF - 攻击性Web测试工具的集合，以自动化和提高渗透测试评估的效率，符合业界公认的标准，如OWASP测试指南，PTES和NIST。

- prowler - 开源的AWS安全最佳实践评估工具，用于评估各种框架的合规性，如CIS、PCI-DSS、ISO27001、GDPR、HIPAA、FFIEC、SOC2、AWS FTR和ENS。

在 Awesome/DevSecOps 的 GitHub 资源库中，提供了一个更全面的 DevSecOps 测试工具实例登记册。

6.5.6 容器测试

本节强调了在类似生产环境中测试容器和容器编排安全的重要性，因为与容器编排和容器相关的配置很容易被遗漏。因此，开发一组有助于验证特定容器和容器编排配置的测试非常重要。特别是，对于测试容器，我们建议创建测试用例，允许测试容器主机和容器编排的配置，以发现潜在的安全错误配置。我们建议的一些测试包括:

- 验证是否不能在生产集群中部署不安全的容器。Kubernetes的一个例子可以是使用坏pod的场景[74]来确保不能创建特权pod。

- 使用root用户测试容器。

运行 Docker bench[75] 和 Kubebench[76]，以确保运行 CIS 基准测试中的最佳安全实践。

- 如果对需要提升权限的容器有特定要求，例如网络或系统管理功能，请按照本文档中的测试指南确保容器中托管的应用程序不易受攻击。

- 确保在集群和容器上启用运行时监控，以了解应用程序基础架构的每个组件中运行的进程。

- 无论何时安排渗透测试，确保如果涉及容器化基础设施，则测试涉及容器逃逸的场景，以确保受损的容器无法轻易逃逸。

有效的容器测试依赖于理解容器硬化中提出的硬化要求。虽然容器强化和测试可能需要时间来实现，但在容器上使用漏洞扫描将有助于识别托管映像和库上的可利用漏洞。

容器漏洞检测程序专门针对容器映像、文件系统、Git 存储库和 Kubernetes 集群，以识别操作系统包和软件依赖关系、已知漏洞、基础设施即代码（IaC）错误配置和硬编码机密。AWS 上的云原生服务就是在其 Elastic Container Registry 上实现这一点的一个例子，像 Trivy 这样获得同等许可的供应商可以提供类似的服务。

虽然容器强化在衡量容器安全控制,有效性方面可能很难执行，也很模糊，但测试的隔离使团队能够了解其容器的可利用性级别。测试可以作为独立活动手动运行，也可以集成到渗透测试中。

6.5.7 完整性检查

完整性检查是管理、签名和验证工件（例如，容器和软件包）。作为构建输出的组件应该经过签名并可信。完整性检查向消费者确保这些组件的完整性。通过对工件和元数据进行签名使得消费者可以更有效地管理其系统中部署的组件中的漏洞，并有助于生成准确的 SBOM。组织通常管理工件、签名

容器镜像清单文件、配置文件以及包和应用程序工件的完整性。

虽然使用工件注册表和管理器不能提供直接的安全控制，但它们为管理工件的完整性提供了基础，并使团队能够有针对性地安排安全加固、漏洞扫描和修复等安全活动。例如，经过漏洞扫描和加固的实例（如容器、清单和配置文件）会作为安全包上传到工件存储库，并可以用作可重复使用的批准模板。

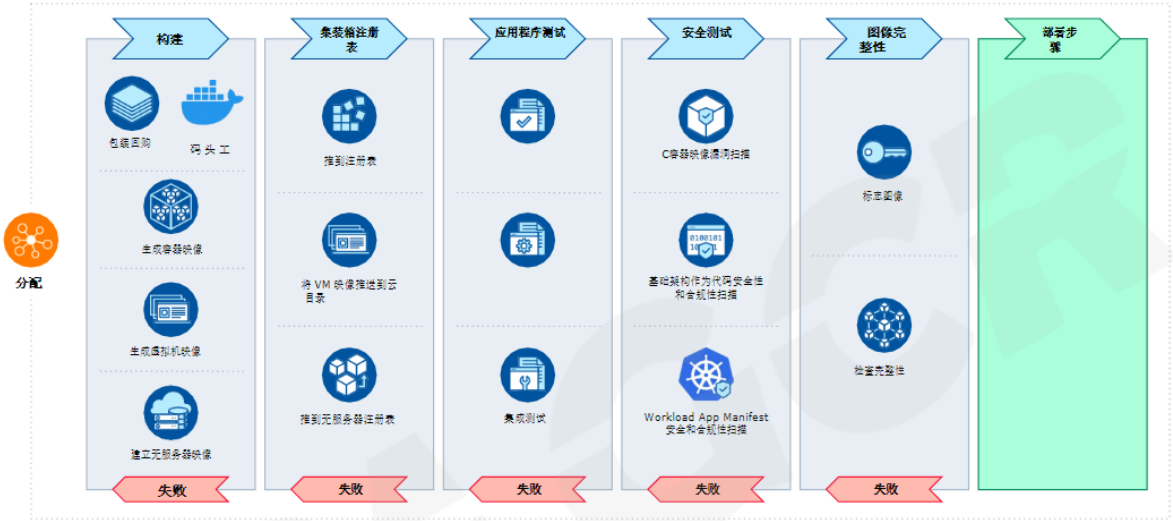

图 19：使用工件注册表的示例工作流程 [77]

Sigstore 和 JFrog Artifactory 是构建工件注册表以确保组件完整性的示例工具。Sigstore 是一个公共软件和透明度服务，类似于 SSL 证书的证书透明度，它可以使用 Cosign 对容器进行签名，并将签名存储在 OCI 存储库 [78] 中。JFrog Artifactory 对工件、二进制文件、包、文件、容器和组件进行托管和管理。

另一种替代的专有解决方案是谷歌云的 Binary Authorization 服务，该服务允许对容器镜像进行签名和准入控制，以防止未签名、过期或有漏洞的镜像在运行时进入生产环境。

7 交付和部署

在部署前进行安全检查，以确保应用程序/产品部署到安全的基础设施上。如果不在部署中考虑安全性，则存在漏洞和不良安全实践削弱应用程序/产品，并被利用来在生产环境中进行攻击的风险。

GitOps[79] 是基础设施即代码(IaC)的一种演变，它是一种实现持续部署的范例，将版本控制、协作、合规和 CI/CD 工具应用于基础设施自动化。

虽然 SDLC 的关键活动已经实现了自动化，但基础设施的相关活动很大程度上仍然是一个需要专业团队执行的手动流程。随着对当今基础设施的需求，实现基础设施自动化变得越来越重要。现代基础设施需要具有弹性，以便能够有效管理连续部署所需的云资源。

GitOps 用于自动化基础设施的配置过程。与团队使用应用程序源代码的方式类似，采用 GitOps 的运营团队使用以代码形式存储的配置文件（基础设施即代码）。GitOps 配置文件每次部署时都会生成相同的基础设施环境，就像应用程序源代码每次构建时都会产生相同的应用程序二进制文件一样。

GitOps = IaC + MRs + CI/CD

IaC: GitOps 使用 Git 存储库作为基础设施定义的唯一真实来源。

MRs: GitOps 使用合并请求（MRs）作为所有基础设施更新的变更机制。合并提交到主（或主干）分支，并用作审计日志。

CI/CD: GitOps 使用具有持续集成（CI）和交付（CI/CD）的 Git 工作流自动化基础设施更新。合并新代码时，CI/CD 流水线会在环境中执行变更。任何配置漂移，如手动变更或错误，都会被 GitOps 自动化覆盖，因此环境会收敛到 Git 中定义的所需状态。

传统流程大多依赖于人工操作知识、专业知识和手动执行的操作，但就 GitOps 而言，所有变更都是与 Git 存储库的交互。Git 存储库和 GitOps 流

程对安全至关重要。基础设施的不变性可保护团队避免在主部署过程之外进行变更。因此基于 Git 存储库中的声明性状态可更容易检测和逆转环境变更。使用 GitOps，团队减少了可以访问生产基础设施的人员和组件的数量。[80]

图 20：GitOps 工作流程

GitOps 通过以下方式提高了对基础设施安全的管理：

- **限制手动操作：** 在部署基础架构变更时减少手动操作。
- **审计变更：** 在各自的 git 存储库中跟踪变更。
- **提供单一的真实来源：** 对 IaC 的集中式管理和部署。
- **执行策略：** 针对安全要求对流程进行控制和把关。

7.1 护栏

护栏（Guardrails）是为云环境提供基线的高级规则。护栏是实施的预防性或检测性控制措施，有助于管理资源并跨资源监控合规性。在 DevSecOps 支柱 4——连接合规与发展的文件中，对护栏的概念进行了更详细的讨论。

护栏是集成的工具，在最好的情况下，在软件开发过程中是自动化的，以确保与组织的目标和目的保持一致。护栏建立了一个可以持续监控部署的基线。这些基线代表应用了检测和预防策略的高级规则。护栏可以作为合规性报告的一种手段（例如，当前批准的镜像列表中运行操作系统的机器数量）或作为一组强制执行/自动修正的控制（例如，运行任何未经批准的操作系统

的设备会自动关闭）来实现。

重点关注由紧密集成的工具和流程在 DevOps 流水线中实现的内联护栏和反馈，可以将合规方法从时间点转变为持续合规。必须考虑确保工具和技术得到实施，以符合商定的合规目标。必须可容易地提供证据，以便通过内部和外部治理实现对这些控制的外部验证。构建有效护栏 [81] 的指南如下:

1. 合理性： 令人困惑或误解的护栏可能会导致护栏被忽视或以检查表的方式遵循.

2. 透明的： 管理团队行为的标准和策略需要是明确的。未知的护栏只会在意外情况下被遵循，不会起作用.

3. 可挑战的： 护栏在其建立的背景下是有意义的。当背景发生变化时，使用护栏的团队需要能够质疑护栏的有效性/必要性。

4. 强制执行的： 违反或忽视护栏需要产生后果。护栏的存在是为了让团队与组织的目标保持一致，并引导行为朝着正确的方向发展。

5. 顶层基调： 护栏为团队提供了一个安全的决策区域，避免了微观管理。一旦建立了护栏，领导者就需要尊重这个“空间”，让团队自行决策和组织，以解决他们正在解决的问题。

《DevSecOps 的六大支柱：自动化》讨论了“实施安全自动化的框架，以及对安全控制的程序化执行和监控，以识别、保护、检测、响应并从网络威胁中恢复”，因为它与保护代码、应用程序和环境有关。这些自动化流程的输出提供了实现合规目标所需的证据。在 DevSecOps 中，部署流水线可以使用甚至构建符合合规性目标的环境。这可能是监控和强制执行控制措施的结合，可能会因环境、数据分类分级和业务风险（即开发与生产）而异。构建护栏 82 时，应考虑以下事项:

- **定义问题：**定义业务结果，防止其成为阻碍因素，然后处理技术问题。
- **设置范围：**多个云帐户现在是标准。确定护栏范围和技术规范。多个帐户可能会影响您运行护栏的位置和管理所需的 IAM 权限。
- **选择部署模式：**关于在何处以及如何运行护栏，有许多相关的技术细节。为进行

更大规模的企业级部署，需要为平衡集中化和本地化的护栏作出关键决策。

- **定义过滤器：**基本过滤器可能适用于简单的资源标签，而更复杂的过滤器可能会评估包含和排除规则的组合，以获得更大的灵活性。
- **确定触发器：**主要类别是基于时间的评估和基于事件的触发器。基于事件适用于更具确定性的事件，而基于时间的扫描则可捕捉在事件触发器上完全遗漏的条件。基于触发器收集所需的信息，并结合范围和过滤器，做出决策。
- **发送通知：**以电子邮件/文本/Slack/工单系统或导出到SIEM等形式，为护栏的创建建立告警信息。
- **验证和日志：**评审和验证护栏的有效性至关重要。记录护栏指标和事件，即使护栏在几秒钟内修复了所有问题。验证和日志记录可保持一致的记录并度量平均解决时间。

安全目标应该直接集成到部署流水线中，并在代码、模板和护栏中定义目标状态。目标应该是一个自动化的审计跟踪，以便于评审，并提供相关的安全指标，供 DevOps 和合规团队持续评审，以推动安全决策。将护栏应用于云环境的示例如下**图 21** 所示。在构建和配置帐户和环境之前应先应用云约束（检测和预防）。

根据组织的合规要求和风险偏好定制护栏。护栏的成功标准依赖于下面确定的三个基本原则，由表 2 中所示的一组控制类别所支持。有关更具体的云安全控制集，请参阅云控制矩阵 v4 [84]。

从信息/反馈、漏洞报告、或跟踪基础设施部署的合规偏差（例如为 VM 使用批准的镜像），到强制执行/阻止（护栏阻止未经批准的 VM 镜像运行），理解不同程度的保证。基于上下文（即开发或生产环境）和相关的风险偏好，这种理解有助于在安全性和开发人员灵活性之间取得平衡。

图 21: 思科的AWS护栏示例 [83]

根据组织的合规要求和风险偏好定制护栏。护栏的成功标准依赖于下面确定的三个基本原则，由表 2 中所示的一组控制类别所支持。有关更具体的云安全控制集，请参阅云控制矩阵 v4 [84]。

- 从信息/反馈、漏洞报告、或跟踪基础设施部署的合规偏差（例如为 VM 使用批准的镜像），到强制执行/阻止（护栏阻止未经批准的 VM 镜像运行），理解不同程度的保证。基于上下文（即开发或生产环境）和相关的风险偏好，这种理解有助于在安全性和开发人员灵活性之间取得平衡。

- 定义所需的目标安全状态或关键绩效指标（KPI）；例如：打补丁、服务级别协议（SLA）、漏洞评估和修复时间。所需的目标状态将取决于组织的安全和风险偏好、云和应用程序环境以及监管要求。“支柱 6：‘度量、监控、报告和行动’文件”将更详细地介绍如何定义所需的目标安全状态。

- 为生产环境建立职责分离（SoD），其中至少需要两个人对应用程序的

关键功能进行变更，以防止欺诈和错误。对于执行概念验证（即开发和测试）的环境，可以放松 SoD 控制，以允许开发人员自由创建具有较少安全控制的功能，以提高生产力。这种对开发人员的自由假设相关的环境不能访问生产数据。SoD 的实施可以在预生产和质量保证环境中开始，以确保安全控制和应用程序功能按预期工作。应用程序所有者还需要确保在发生事故时可以使用 SoD 方法恢复应用程序可用性。

虽然 Guardrails 的设计和实施在逻辑上具有挑战性，但它们具有以下优势，可以转变组织的云安全方法，[85] [86]。

Guardrails:

- 定义一系列可接受的行为。Guardrails 之间的距离允许团队和个人决定他们的路线。
- 加快决策过程。Guardrails 提供了一系列问题的解决方案和决策的起点，并缩短了做出设计或安全决策所需的时间，使开发人员能够快速操作，而无需安全团队的积极参与。
- 在不牺牲一致性的情况下，为团队和个人提供基于上下文做出业务决策的独立性。高效软件开发（所有形式）的一个标准原则是业务定义需要做什么。“什么”定义了开发人员（最广泛的用法）需要解决的业务问题；这些是 Guardrails。然后，技术团队定义他们将如何解决业务问题。技术和架构标准以及 Guardrails 提供了一个环境，允许团队在不征求许可的情况下做出决策。
- 通过自动化开发人员工作流程中的安全控制，允许产品团队进行扩展。这种自动化确保了强制执行，而不必手动执行评审和跟踪漏洞补救。
- 降低风险。架构指南为软件开发人员提供了同样的服务。Guardrails 的目标是减少基于他人进行的研究和实验的返工的可能性，以便将尽可能多的时间用于解决业务问题。

- 当控制被纳入开发过程时，降低开发团队忽略和绕过安全性的风险。

示例云本机指南可用；例如，AWS 提供了强制性、选择性和强烈推荐的 Guardrails。而 Guardrails.io 这样的授权产品有助于在应用程序开发和构建中建立防护。

7.2 环境分离

本白皮书中的环境是服务于特定目的的不同技术组件的集合。环境的示例包括不同的网段，例如，开发中使用的现有开发环境（例如开发、测试、试运行和生产）。认识到这个定义的广泛性，从云中 DevSecOps 的角度定义了一个相关的分类。

图 22. 环境层

图 22 对现代基于云的 DevSecOps 环境中的不同环境进行了分类。上述特征解释了每层的分离（或隔离）要求。例如，开发环境（开发、测试、试运行、生产、沙盒）、网络段、逻辑资源容器（例如，最常见的 CSP 中的资源组）之间所需的隔离，以及需要由支持微服务的基础设施（例如，Kubernetes、Nomad）

满足的任何多租户要求)。

按照 **图 22**,中的模型，为每一层提供了具体的分离基准和注意事项，提供了实用的指导方针和注意事项，可以帮助企业架构师、开发人员和组织实施适合云环境开发人员的环境分离策略。

组织等级：在组织等级中被认为是最外层的。这个环境在最高级别上代表组织，它的组件将是组成它的不同组织实体。实际上，这一层由云提供商实施，以确保模块化结构，在实施安全控制(如访问要求)时，允许职责分离、受控计费和灵活性。[87] [88] [89] 在组织层面，主要关注的是建立不同组织单位、子组织和第三方组织之间的界限。目标是确保每个组织单元只能访问它应该能够访问的内容（即最低特权原则）。与每个组织单位进行适当的协调并定义角色和职责非常重要。

一旦定义了这些内容，应在组织内开发和实施流程和技术手段（即自动执行的策略和访问控制），以确保定义的边界得到遵守。请注意，尽管这种隔离级别可能更适合企业架构指南，但是如果这些组件没有得到适当的定义，就不可能在较低层提供适当的隔离。利用不同云提供商提供的技术手段可以保证隔离。例如，在 AWS 中，组织单位 [90] 和服务控制策略 [91]（scp）可用于管理组织内的权限以及创建分层组织结构；以 Azure 为例，管理组 [92] 可用于建模分层组织结构，并确保根据每个组的需求应用策略。在 Google Cloud 中，folders[93] 允许组织创建层次结构并组织不同的项目。

开发团队： 开发人员将创建、托管和运行他们自己的应用程序。这一层由贯穿端到端开发生命周期中使用的不同开发环境组成。对于开发团队来说，隔离保证了对他们管理的应用程序和系统的完全所有权。这种级别的隔离是为了确保每个团队都有自己的工作区域，不受其他团队工作的影响。请记住，这个建议并不提倡团队各自为政。相反，团队环境之间不同级别的权限需要被适当地授予。与上面的层类似，这里的隔离包括为每个开发团队定义特定的角色和职责，并确保定义过程和技术手段，以确保维护和执行适当的分离级别。这些团队可以

被建模为包含特定资源的特定组织单元。

开发环境：表示用于开发应用程序的常见环境：开发、测试、试运行、生产、沙盒和共享。这些环境将具有不同的功能，使开发人员能够并确保软件开发按预期进行。这一级别的隔离旨在引入隔离，以便可以在整个 SDLC 中测试工作负载，而不会影响生产数据和基础架构。成熟的组织通常有六种不同的（共享的）环境:

开发：大部分的开发都发生在这里。在这种环境中，开发人员可以修改、测试和尝试代码变更，而不会破坏实时应用程序和系统的任何功能。新功能和更新首先在这种环境中进行尝试，以确保实时应用程序的稳定性。从安全角度来看，对此环境的变更不得影响实时客户数据或基础架构。例如，开发 web 应用程序不应该能够修改生产数据库。

测试： 大部分测试都在这里进行。虽然一些测试可以作为构建阶段的一部分在开发环境中运行（例如，单元测试），但是大多数集成和功能测试都发生在测试环境中。与开发环境一样，建议不要在此环境中存储或修改实时客户数据或基础设施。

暂存： 生产环境的副本。这个环境应该用于执行任何可能需要的附加测试（端到端测试、回归测试、负载测试和一些渗透测试）。该环境旨在提供一个尽可能接近生产的隔离环境，以确保所开发的软件工件按照预期运行。

生产： 生产是客户数据和应用程序所在的环境。这种环境的隔离要求对于确保安全性至关重要。因此，建议人员进入这种环境仅限于“打破玻璃”紧急程序。所有环境更改都是通过 CI/CD 编排器等自动化、可审核的工具来执行的。

沙盒： 侧重于实验和测试新技术和架构的特殊环境。这些环境通常是不稳定的，需要隔离以确保其他环境不会受到环境中引入的任何潜在变化的影响。

共享资源 - 托管特定需求的可共享资源的环境，例如身份验证或 web 代理、CI/CD 工具、特定安全控制、VPN 端点、内部 API 网关，或者充当其他账户的日志或事件的数据收集器账户。

可以使用不同的方法来确保保持这种分离级别。例如，一个组织单位内的不同账户或订阅可以用来表示每个环境。通过这种方式，可以保持不同环境的逻辑隔离。然而，同样重要的是要记住，分离这些开发环境也应该考虑较下层的隔离。例如，不能将生产数据库与测试 web 应用程序连接起来。

工作负载账户： CSP 账户还可以为隔离处理不同敏感级别数据的工作负载提供硬边界。使用单独的工作负载账户可确保开发工作负载资源无法访问无法访问产品工作负载资源，反之亦然，并最大限度地减少该工作负载账户可能的爆炸半径。工作负载是交付业务价值的资源和代码的集合，例如面向客户的应用程序或后端流程。[94] 这些账户可能在复杂性、范围和目的方面有所不同。通过在不同类型的环境（开发、测试、试运行或生产）之间提供账户级别的分离，将有助于支持 SDLC 流程。[95]

资源组： 在云中，资源是可以代表基础设施、服务器和托管运行时的资产。资源组是对工件、应用程序、系统和不同组件进行逻辑分组的一种方式。请注意，我们对资源组使用了一个通用的定义，尽管这些资源组是相关的，但不应该与不同云提供商（如 AWS [96] 和 Azure [97]）使用的定义相混淆。资源组之间的隔离允许跟踪相关的资源集群。这是一个逻辑分离层，但它允许跟踪和确保设计的解决方案的不同组件的可见性。与上面的开发环境一样，确保资源组是隔离的（在网络层面）是一个很好的实践.

资源：我们认为资源是应用程序和解决方案的基础层。如上所述，它们代表了数据库、无服务器功能、web 应用程序以及云中提供的所有不同类型的服务（无论是私有的还是公共的）。这一层面的隔离考虑了网络级隔离、人类和非人类账户对不同资源的访问控制，以及所实施的解决方案的租用要求等方面。

- **网络分段：**使用不同的技术（例如，防火墙、路由器等）在网络层面隔离不同系统的决策和活动。）或使用软件定义的解决方案，如 web 应用程序防火墙（WAFs）、容器网络接口（CNIs）。这一层面的安全实践包括网络层面的微分段和零信任等方法。

- **访问控制：** 确保对资源进行适当的授权和身份验证，确保职责分离。目标是遵循最小特权，确保仅在需要时授予访问权限（即，及时）。有几种方法可以确保资源层面的访问控制。从高层次的角度来看，一些方法包括基于角色的访问控制 [98]和基于属性的访问控制 [99]，前者定义和分配具有特定权限的角色，后者评估特定的主题属性以获得权限。

租赁要求： 在使用容器编排器（例如 Kubernetes、Docker Swarm、Nomad、Openshift 等）托管的容器化基础架构中。在多租户环境中，可能不相关的解决方案通常托管在同一个集群中，并使用某种逻辑隔离（例如，名称空间）。除此之外，这些应用程序通常有特定的安全需求，这可能需要与 CNI 或 Web 应用程序防火墙（WAF）相关的特定隔离技术。

应用程序： 应用程序是托管在资源中或位于第三方基础架构中作为软件即服务使用的软件程序。这一级别的分离要求应用程序不能访问无关的资源或其他应用程序，并确保只根据应用程序的需求授予访问权限。从这个意义上说，访问控制机制起着至关重要的作用。

利用过程和设计实现环境分离，不需要专有工具来专门实现。此外，借助强大的环境隔离，组织在为用户、组、资源和资产定义对这些环境的访问时，可以考虑零信任原则 [100]。虽然这有多种原因，但最令人信服的一个原因是，在大多数情况下，组织将使用商业 CSP（流量通过互联网传输并公开），从而增加了组织 IT 基础架构的受攻击面。与此同时，身份认证和授权中方面的访问控制以及开发人员和非人类账户的信任要求面临新的挑战。

目的是通过实现零信任原则来避免继承的环境信任：通过最小权限进行微分段、假设违反和连续验证。只有在基于动态策略和其他行为和环境属性的固定持续时间内进行了正面的身份验证和授权之后，才能授予访问权限。

7.3密钥和密钥管理

秘密管理： 秘密管理是管理秘密访问凭证的实践。秘密是用户和系统组件

用来验证和授权服务的键值对。秘密管理的一个主要问题是在版本控制的源代码和配置文件中暴露秘密。秘密通常是用各种密码函数生成和加密的。秘密管理系统旨在降低泄露秘密的风险。秘密管理系统管理秘密的生命周期，并提供秘密的安全存储和传输 [101]。秘密的例子包括有用户凭证和机器对机器的访问凭证。用户凭据向系统验证用户，包括密码、SSH 密钥和一次性密码。机器对机器凭证包括 API 访问令牌和 web 令牌。

密钥管理：密钥管理还包括密钥管理。密钥管理是创建、存储、使用、轮换和销毁加密密钥材料的实践。加密密钥是加密算法用来将纯文本转换为密文的一串比特，反之亦然。对称密钥用于对称加密明文数据和解密密文数据。非对称密钥（也称为公钥）使用私钥和公钥来创建、管理和撤销数字证书，签署和验证数字签名以进行传输加密，如 TLS/SSL。在云环境中，密钥管理系统提供了一种安全存储加密密钥和数字签名算法的机制。NISTSP 800-57[102] 系列为云环境提供了重要的管理指南。该出版物描述了加密算法和公钥基础设施使用的建议和最佳实践，包括使用加密技术时可能提供的安全服务的定义以及所采用的算法和密钥类型。该建议也适用于管理密钥。

密钥和密钥管理生命周期：在云环境中，密钥和秘密通常由保险库管理。保险库提供密码、数据库、密钥和证书等通用秘密的安全存储。在一个保险库中，一个秘密可能在其管理生命周期中经历一个或多个阶段，这些阶段包括:

- **创建：**生成密钥或密钥的阶段。在这个阶段，密钥管理系统确保密钥满足足够的复杂性要求.

- **激活：**密钥管理系统可以在激活之前使密钥处于预激活状态。在此阶段，它已经生成，但未被授权使用。在激活阶段，密钥随后被授权使用。

- **轮换：**在轮换阶段，密钥被更新或替换。秘密管理系统在一定的时间段内，允许自动轮转秘密，无需用户交互。

- **撤销**:当秘密被泄露或在其到期日期之前不再需要时，就会被撤销。

- **过期**: 在秘密管理系统中，可以将秘密配置为停用，以防止秘密用于身份验证。在云环境中，秘密可能是短暂的，并在定义的时间后过期。

- **销毁**: 销毁阶段是指秘密管理系统销毁和移除秘密或密钥的阶段。秘密管理工具确保对每个秘密的操作都被审计，包括销毁一个秘密。

通过秘密管理系统管理秘密有助于实践 DevSecOps（开发、安全和运营）的若干好处 [103]:

- **集中化：** 秘密管理系统支持集中管理分布在云环境中的各种秘密。秘密管理旨在保护处于静止状态以及传输过程中的秘密。

- **短期密钥和秘密：** 秘密、密钥和证书可以被配置为短期的，具有定义的到期时间，并在到期后或在秘密被泄露的情况下被撤销。

- **审计和日志：** 密钥管理确保生命周期中的所有操作都被审计和记录。存储秘密管理解决方案的审计日志有助于定位未授权的访问问题，并保留重要信息以供事件分析。

- **跨平台访问：** 秘密可以使用在不同的平台以及系统开发的所有阶段，包括本地开发、自动化构建、预上线和生产环境。使用支持多云的秘密管理解决方案可以提供一个完整的托管解决方案，来处理秘密的整个生命周期以及它们对用户和机器的分配。

有效的秘密和密钥管理的实用方法是使用云服务提供商（CSP）提供的秘密管理解决方案；然而，授权服务可以帮助降低 CSP 管理和托管秘密的风险。云服务提供商提供秘密和密钥管理系统，或者作为独立的解决方案，用于多云目的。秘密管理系统允许秘密的集中化，并提供基于角色的细粒度访问控制和仅对可信身份配置秘密访问策略的功能。一些例子包括:

- **GCP秘密管理器：** 在GCP中，秘密管理器服务允许管理API密钥、密码、证书和其他敏感数据。

- **AWS秘密管理器：** 在AWS中，秘密管理器使用AWS密钥管理服务来存储和加密密钥的受保护文本。

- **Azure密钥保险库：**对于Azure，Microsoft提供Azure密钥保险库服务，用于在Azure中管理和轮换秘密、证书和加密密钥。Hashicorp的Vault是一个流行的独立解决方案，用于跨多个云提供商和内部部署基础设施管理机密。Vault特别目标是在多云部署中管理和轮换机器对机器的秘密。

  HashiCorp Vault：HashiCorp Vault 是一个多云秘密和密钥管理解决方案，组织可以使用它在混合和多云环境中集中管理秘密。HashiCorp Vault 的一个主要功能是管理动态秘密，这些秘密在被访问时生成。Vault 可以管理整个秘密管理生命周期中的动态秘密和密钥。

  云HSM服务：云硬件安全模块（HSM）是密钥管理服务，它允许在FIPS 140-23级认证的HSM集群中托管加密密钥，并进行加密操作。这些服务允许在各自的云提供商内创建和使用加密密钥。

总的来说，对于云环境使用秘密和密钥管理系统是高效的 DevSecOps 实践，它让我们从手动安全操作转向更自动化的文化。想要了解更多关于密钥管理的信息，可以参考云安全联盟（CSA）关于云服务上的密钥管理的发布。

7.4 保护 CI/CD 流水线

持续集成、交付和部署是产品交付和部署的方法。这些方法允许定期进行代码提交，触发构建并运行测试，同事部署最大限度地减少对手动流程的需求。

CI/CD 流水线允许更快、更多样化、更灵活的交付。然而，将 CI/CD 流水线加入技术栈会引入可以被利用的攻击面，例如:

- 从第三方源导入到CI/CD流水线中的不安全代码
- 未经授权访问源代码库

- 破坏开发/测试环境的使攻击者禁用安全测试
- 流水线内的秘密管理不安全

如果需要保护代码及其构建的系统的完整性，则此流程的安全性就至关重要。然而，安全应该与这个过程协同工作，而不是阻碍它。CI/CD 的安全风险可以在 4 个领域得到解决；治理、身份管理、部署配置和 git 仓库管理。

7.4.1 治理

对第三方服务的治理需要包括流水线依赖审批、管理和退役。对于审批，对第三方的评审过程应包括资源访问权，而对于管理和停用，应保持对集成方式、权限、秘密访问和所有集成到 CI/CD 系统中的第三方的进出网络连接的可见性。

潜在的漏洞可能涉及参与 CI/CD 过程的任何系统，包括 SCR、CI、Artifact 仓库、包管理软件、容器注册表、CD 和协调引擎。通过以下方式可以进一步提高 CI/CD 流水线的管理和治理能力:

- 通过签名资源（软件和基础设施）进行流水线证明，用于开发和生产环境中使用，根据签署机构验证资源完整性 - 参阅完整性检查。Artifact 验证工具可以作为阻止未验证的软件通过流水线交付的一种方式。
- 通过可视化功能理解应用生态系统可以帮助熟悉所有涉及的系统。一个完全映射的应用生态系统可以在设计阶段帮助安全评审和威胁建模。
- 确定所有相关系统的日志源可以确保相关日志被启用和集中。这应包括人员和程序访问。日志的聚合和集中（如SIEM）应是支持检测异常和潜在的恶意活动的基础。

治理也可以通过有效的 Git 仓库管理实现，方法是为 Git 存储池连接到流水线部署的开发人员应用 Git 限制和参与规则:

- 限制自动合并规则并管理生产和源代码仓库之间的漂移。参阅 GitOps。目标在于检测配置漂移的措施（如，云环境中未使用签名的 IAC 模板进行管理的资源），这可能表明资源是由不受信任的源或进程部署的。

- 检测上传/推送到代码仓库和存储在代码仓库上的秘密。请参阅安全挂钩。

- SCM解决方案提供了使用每个贡献者的唯一密钥签名提交的能力。这个措施可以用来防止未签名的提交流入流水线。

7.4.2 身份管理

对 CI/CD 流水线和依赖服务（如源代码仓库）的身份管理，支持程序和人员访问的最小权限原则。为了保护 CI/CD 流水线，应考虑采用基于角色的访问控制（RBAC）设计，以免向所有用户授予系统中的基本权限。确保后续对网络资源和流水线节点的访问反映了 RBAC 设计，并符合最少访问原则。作为 RBAC 的补充，CI/CD 流水线应解决:

- 移除在SCM中可以将未评审的代码推送到仓库，并随后在CI/CD流水线上部署流水线的用户账户。

- 移除每个工具的本地用户账户，每个账户有不同级别的访问权限。相反，应使用单点登录（SSO）通过集中的身份提供程序来管理身份。

- 考虑将凭证绑定到预定的条件（如IP地址、FQDN、地理位置或VPN绑定），以确保受损凭证不能在环境外使用。

7.4.3 部署配置

流水线管理、设计和配置对于如何利用 CI/CD 流水线也很重要。Cider 的 Top 10 列出了 CI/CD 流水线中的主要安全风险。对于每一项风险，我们

都提供了一系列建议。我们将这些建议分为了以下几种，可以在您的流水线配置中进行变更。

■ 包:

● 启用对拉取的包进行校验、验证和签名验证。

● 阻止直接从互联网或不信任的源获取/拉取代码包。

● 配置所有客户端从包含预先评审过的软件包的内部存储库中提取软件包，并建立一种机制来验证并强制执行此客户端配置。

● 确保客户端被强制从组织内部注册中心获取软件包。

● 评估从外部贡献者的公共软件源触发流水线的需求。

● 避免配置客户端为拉取包的最新版本。

■ 流水线机密:

● 对于暴露给秘密的流水线，确保在CI系统中配置触发流水线的每个分支在SCM中都有一个相应的分支保护规则。

● 确保在CI/CD系统中使用的秘密仅限于所需的秘密。

● 确保从任何工件中移除了秘密，如容器镜像、二进制文件或Helm图表层。

● 监控流水线日志中敏感数据的暴露情况，如秘密、密钥和环境变量。

■ 逻辑和网络分割:

● 确定并配置托管生产代码的分支上的分支保护规则。

● 运行未评审代码的流水线应在隔离的节点上执行，不暴露给机密和敏感环境。

● 不要使用共享节点用于需要访问不同资源的、具有不同敏感级别的流水线。

● 共享节点应仅用于具有相同机密级别的流水线。

● 为了防止操作CI配置文件在流水线中运行恶意代码，每个CI配置文件在流水线运行之前都必须被评审。或者，CI配置文件可以在远程分支中管理，与流水线中构建的代码包含的分支分开。远程分支应被配置为受保护的。

● CI和CD流水线作业在控制器节点上应具有有限的权限。在适用的情况下，应在单独的专用节点上运行流水线作业。

● 确保网络分段配置为允许执行节点仅访问所需的资源。在可能的情况下，不要授予构建节点无限制地访问互联网的权限。

确保安装脚本作为包安装的一部分被执行（这些脚本有单独的上下文），该上下文不能访问构建过程中其他阶段可用的秘密和其他敏感资源。

7.5 系统加固

系统加固是任何用于直接提高系统安全性的过程、方法论、产品或组合。[104] 系统加固是限制使系统易受攻击的潜在弱点的过程。这个过程旨在通过保护数据、端口、组件、功能和权限来减少系统的攻击面。安装后，常见的服务器操作系统通常会由于缺少补丁和不足的安全配置而存在漏洞。系统加固的过程大大减少了已知漏洞的风险。然而，确保系统在加固后仍能正常运行也很重要。系统加固包括几项任务，以减少系统的攻击面，包括以下内容[105]：

● **初始系统设置：** 任务包括文件系统加密、配置安全引导机制、文件完整性检查和登录警告标语。对于Linux，此任务还可能包括启用内核安全模块

Security-Enhanced Linux (SELinux)和高级入侵检测环境 (AIDE)进行文件完整性检查。

- **服务配置：** 不需要的系统服务被停用，以最小化操作系统服务和应用程序的攻击面。额外的任务可能包括安装端点保护和防病毒软件。

- **网络配置：** 任务包括配置非必需的网络服务，如禁用ICMP重定向和非必需的网络接口。一项主要任务是通过配置iptables，添加默认拒绝策略和禁用不使用的IPv6，在操作系统层面配置防火墙。

- **日志和审计：** 日志和审计加固措施包括配置系统日志，如Syslog，审计数据保留，以及对访问控制的修改。登录尝试，文件访问和使用特权操作也需要被记录。

- **访问，认证和授权：** 确保按照最小特权原则，最大限度地减少操作系统上的权限分配。其他任务包括用户帐户配置，密码复杂性要求，以及 RDP 和 SSH 等身份验证服务的配置。

- **系统维护：** 系统维护任务包括配置文件和目录上配置文件权限和组设置，以及对特权管理操作进行多因素认证。此外，还有更新和打补丁的任务，以确保应用了最新的补丁。

系统加固有各种标准和推荐[106]

- **CIS 基准：** 一个实用的系统加固方法是使用信息安全中心（CIS）发布的系统安全基准。CIS已经发布了CIS基准，这是一套系统加固的安全建议。目前有超过100个CIS，涵盖超过 25 个产品系列。这些包括操作系统，Web服务器，和云平台到容器编排环境的建议。

- **STIG和SCAP:** 安全技术实施指南(STIG)[107]是另一套系统加固的推荐。STIG最初是为美国国防部开发的，以提供系统和应用的配置标准。这些标准增加了攻击者访问系统的难度。为各种产品定义了标准，如操作系统，网络设备，数据

库和移动设备。安全内容自动化协议（SCAP）支持自动执行STIG指南。SCAP是一种使用特定标准的方法，帮助组织实现自动化漏洞管理和政策合规性评估。SCAP和STIG支持自动化扫描包括云环境在内的各种操作系统的过程。

- **黄金镜像：**在云环境中建立一套不变的加固镜像的一种实用方法是创建金镜像。黄金镜像比开箱即用的镜像更安全，它是一个预配置的虚拟镜像模板。黄金镜像为云环境的部署提供了一个安全的配置基线。创建黄金镜像的一个广泛使用的工具是Hashicorp的Packer。Packer允许使用单一源模板为云基础设施创建相同的镜像[108]。CIS提供了一套加固的服务器镜像，组织可以将其用作黄金镜像。在公有云平台中，也可以直接从相应的云提供商的市场中购买CIS加固的镜像。[109]

系统加固通过删除所有非必要的软件程序和实用程序[110]来降低系统的攻击面和安全风险。特别是使用 CIS 或 STIG 等安全基准和加固的黄金图像，可以加强基于行业指导的安全态势，而无需花费时间和精力来微调云系统和应用程序中的特定安全设置。此外，行业指导方针使加固过程变得稳健和全面，而不会降低适应特定用例和场景的需求和目标的灵活性。

总体而言，系统加固减少了攻击面，从而减少了攻击者进入系统的机会。它还降低了因系统受损或未经授权访问而导致声誉损失的风险。

8 运行时

应用程序/产品发布到生产环境后可以应用的功能和实践。通过识别低效率、漏洞、弱点以及启用事件响应等持续改进运行时安全。

8.1混沌工程

测试计算系统（例如：微服务架构）确保其能够承受意外中断的过程。混沌工程的核心思想是实验驱动的测试。混沌工程是指用具体的假设来定义特定的场景。然后在实验过程中对这些假设进行测试。如果出现失败，则会

记录经验教训，并实施改进以进行进一步的实验迭代。 简而言之，目标是找到可能完全破坏组织系统业务的潜在故障点。

尽管混沌工程被认为是一种测试潜在可用性风险（如：拒绝服务或关键系统宕机）的工具。 同样的概念可以应用于安全控制的测试。通过这种方式，安全控制故障/弱点被认为是由于随着时间的推移而被假定，并在受控环境中被主动发现的。团队养成持续透明和小组分析的习惯，减少对有限和时限的根本原因分析的依赖。这种类型的实验通常称为安全混沌工程 (SCE)。[111]

通过主动实验来识别安全漏洞或安全控制侵蚀。主动实验可以通过在 CI/CD 流水线或持续验证流水线中引入意外变量或“故障注入”。SCE 用例也可以通过在运行的工作负载中执行故障注入来执行。

对于需要每个服务正常运行的复杂系统，这些服务之间的交互可能会导致不可预测的结果。 这些不可预测的结果，以及影响生产环境的破坏性事件，使得分布式系统本质上是混乱的。 当服务不可用时，混乱事件可能源自不正确的后背设置；下游依赖系统接收到过多流量时发生的中断；单点故障崩溃时的连锁故障。一种基于系统经验的方法在一定程度上解决了分布式系统中的混乱问题，并建立了对这些系统承受现实条件能力的信心。

将 IT 系统的稳态和正常行为定义为可测量的输出。在短时间内对该输出的测量结果可以代表系统的稳态状态。整个系统的吞吐量、错误率和延迟百分比都可以表示相关的稳态行为指标。通过在实验过程中关注系统行为模式，Chaos 验证系统是否工作，而不是试图验证它是如何工作的。

引入反映服务器停机、硬盘故障、网络连接问题、软件故障和流量峰值等事件的变量。

通过采样实际流量，在生产中进行混沌测试。为了真实性和相关性，chaos 倾向于直接在生产流量上进行实验。如果无法在生产中进行测试，则应在尽可能接近生产且具有实际流量的环境中进行测试。

限制范围和规模以避免和减少测试的影响。 确保将实验的影响降至最低并加以控制。

作为流程利用文档的 Chaos 实验模板应包含:

- 实验名称、作者和修订版本的标识
- 通知的人员和团队
- 实验范围
- 实验假设
- 实验中需要观察的指标
- 在生产中进行实验的风险评估
- 减少上述风险并将实验爆炸半径最小化的防护栏

虽然混沌工程（chaos engineering）可能很难实现，但它是 DevOps 成熟度的指标，其具有以下方面的益处:

- 对已发现弱点的应用程序和基础设施的弹性、设计、流程和配置改进可以更好地保护您的产品。
- 提供工作实践以持续改进组织内使用的系统。
- 能够测试复杂体系架构的混乱和恢复活动。

工具可以使混沌工程体验自动化，而无需定义系统稳态或假设实验。一些可用的工具包括 Chaos Monkey[112], AWS 故障注入模拟器 [113], Azure Chaos Studio[114] 和 Chaos 工具包 [115]。

8.2 云安全态势管理

云安全态势管理 (CSPM) 主要用于监控云帐户或订阅中的云合规性、漏洞和资源配置错误。 云配置错误是导致数据或资源遭受攻击的典型错误，

这在云服务的设置过程中很常见。 许多大型数据泄露都是由于云资源配置错误而造成的，例如：存储桶暴露，数据暴露。

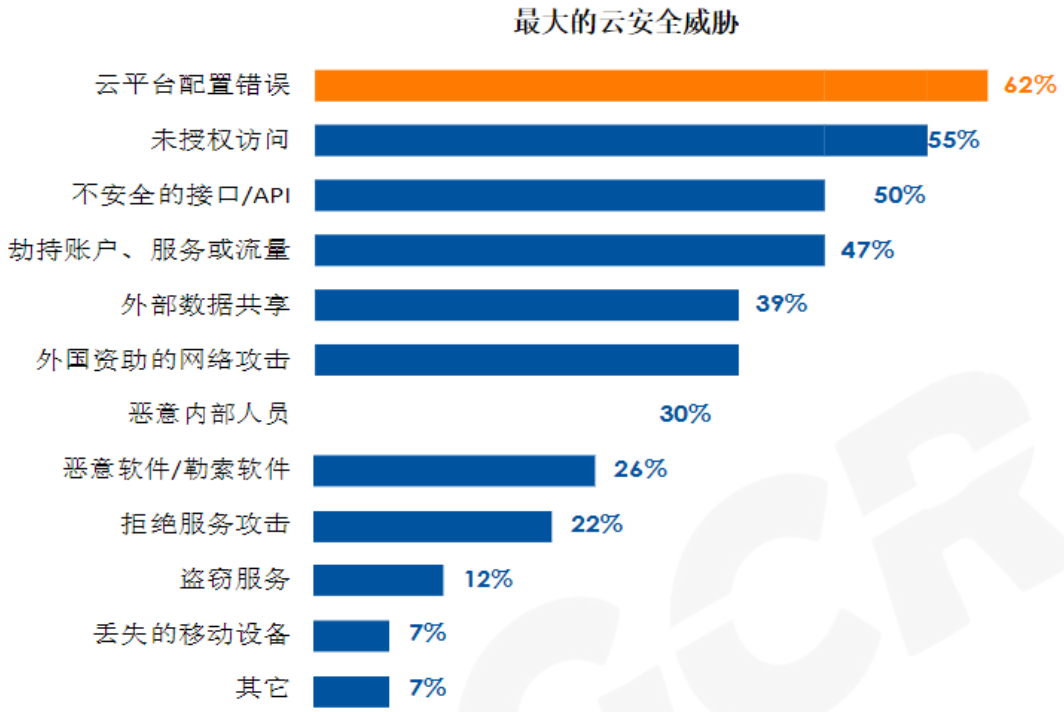

图 23：关于云安全威胁的调研结果[116]

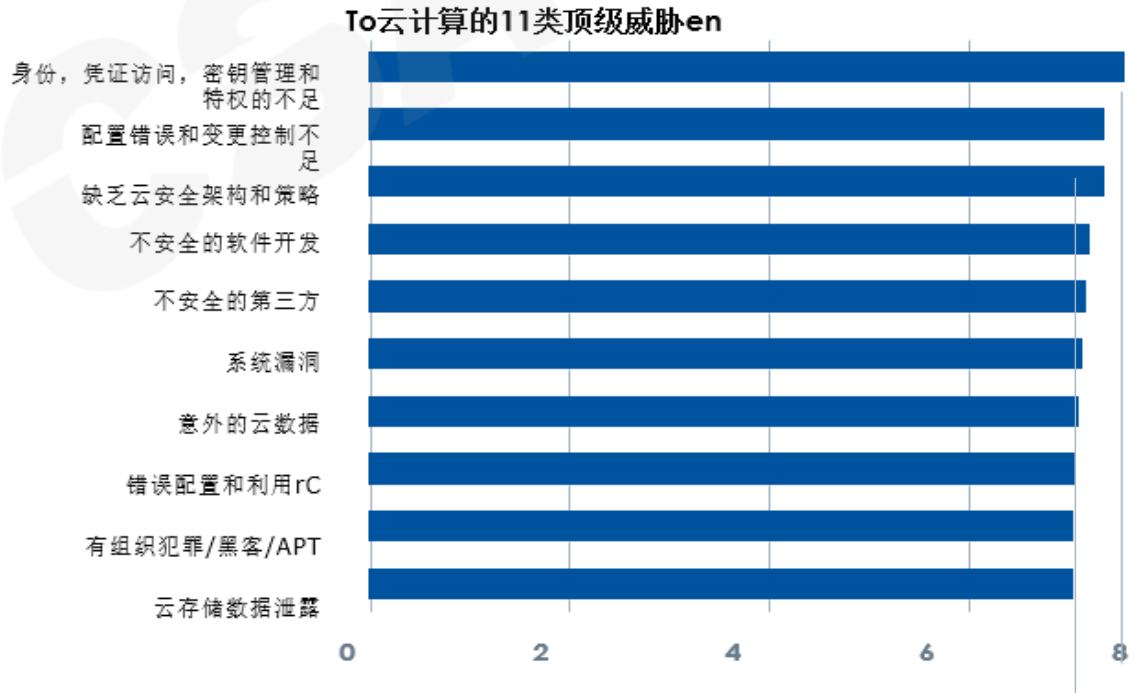

图 24: CSA 关于云计算的 11 类顶级威胁

CSPM 的核心目的是通过一系列功能来持续监控云资源的安全策略执行缺口、安全威胁和数据安全。 CSPM 通过对云服务的 API 调用、访问 CSPM 事件触发警报的日志记录来实现此目的。CSPM 是构建安全基线（例如：WAF，防火墙规则）、检测安全错误配置（例如：有风险的访问和权限）以及应用自动纠正控制（例如：自动修补，图像刷新）的辅助工具。

组织需要遵守保护数据和控制数据访问的严格要求。受监管的行业必须遵守特定的基准； 例如，零售业的 PCI DSS、医疗保健的 HIPAA、金融服务的 FFIEC、NIST、GDPR 和 ISO27001。 CSPM 自动扫描并检测违规行为，这有助于组织更好地遵守法规。

虽然可以通过使用 CSPM 来实现合规收益 (请参阅 支柱 4：连接合规与开发)，但可以应用控制措施来补充云工程的运营效率 - 结合本文推荐的以下活动，可增加 CSPM 的效益:

- 基础设施即代码分析：在构建阶段修复配置错误和易受攻击的云资源，以减少部署云服务后 CSPM 识别的安全问题数量。

- **GitOps：** 同步从 IaC 上的 CSPM 识别的安全变更，以自动部署为实时云服务。

- 护栏： 使用预设规则和 CSPM 调查结果的反馈，在多云环境中定义一致的防护轨迹。

- 系统加固： 使用 CSPM 指导云资源的基础架构加固决策。

- 尽管 CSPM 的采购和实施成本高昂，但其带来的好处不仅仅是提高云安全性：

- **云服务的安全基线配置：** 基于合规框架映射（例如： CIS 基准），开发人员可以采用一套行业标准的护栏，或者可以调整该框架以符合公司和监管政策，并适当调整控制和监控。

- **实时云资产清单：** 无论组织是否采用 DevSecOps，发现和盘点所有软件

和云资源对于成熟的安全能力都至关重要。 CSPM 解决方案定期扫描云环境并维护该服务和云资源清单。

- **数据资产存储：** CSPM 解决方案还可以清点云对象和服务，并列出存储在存储容器、数据库和其他数据存储库（例如： PII）中的数据对象。在了解系统处理或持久化的数据类型后，组织可以 选择调整适用于该系统的安全控制并/或完善该对象或组件的风险计算，从而形成与风险相称的不断发展的安全控制措施。

一般可以通过许可的提供商获得云安全态势管理工具。市场上的一些 CSPM 产品包括 Wiz, Palo Alto's Prisma Cloud, Checkpoint, Crowdstrike, 和 Zscaler。

8.2.1 监控和可观测性

监控是收集、聚合和分析数据以跟踪应用程序和基础设施的过程。监控有助于:

- 指导业务决策
- 提供反馈以尽早快速发现并解决问题
- 将系统信息传达给企业的其他部门
- 检测和响应可疑活动、网络事故和事件

监控是捕获和显示数据，而可观测性可以通过分析系统的输入和输出来识别系统的健康状况。例如，我们可以积极地观察单一的指标变化，来判断是否存在问题 —— 这就是监控。

监控可以包括底层硬件资源、网络传输、应用程序/微服务、容器、接口、正常和异常端点行为以及安全事件日志分析。**表 3** 是来自美国国防部 DevSecOps 参考架构的监控工具和阶段摘要 [117]。

表 3：监控工具和阶段摘要

活动/工具	描述/功能	输入	输出
日志记录	记录所有用户、网络、应用程序和数据活动的事件	所有用户网络、应用程序和数据活动	日志
日志分析与审计	分析和审计以检测恶意威胁/活动； 用于响应的自动警报和工作流程； 用于损害评估的取证功能	日志	警报和补救报告
系统性能监控	监控系统硬件、软件、数据库和网络性能； 建立系统性能基线； 检测异常情况	运行系统	性能KPI衡量标准； 建议采取的行动； 警告或警报
系统安全监控	监控所有系统组件的安全性； 安全漏洞评估； 系统安全合规性扫描	运行系统	漏洞； 不合规调查结果； 评估和建议； 警告和警报。
资产清单	盘点IT资产	IT资产	资产清单
系统配置监控	系统配置（基础设施组件和软件）合规性检查、分析和报告	运行系统配置； 配置基线	合规报告； 建议采取的行动； 警告和警报
数据库监控和安全审计	基线数据库性能和数据库流量；执行用户访问和数据访问审计；从事件相关性中检测异常；检测SQL注入；生成警报	数据库流量、事件和活动。	日志； 警告和警报

如果一个系统发出有关其内部状态的有用数据，那么它就具有可观测性，这些数据对于确定根本原因是至关重要的 [118]。可观测性是根据系统生成的数据（例如：日志、指标和跟踪）测量系统当前状态的能力。可观测性的目标是了解所有环境中发生的情况，以便检测和解决问题，以保持系统高效和可靠的问题。许多组织还采用可观测性解决方案来帮助他们检测和分析事件对其运营、软件开发生命周期、应用程序安全和最终用户体验的重要性 [119]。日志、指标、分布式跟踪和用户体验的测量是实现可观测性成功的关键支柱:

- **日志：** 在特定时间发生的独立事件的结构化或非结构化文本记录。
- **指标：** 这些值表示为计数或度量，通常在一段时间内计算或汇总。指标可以来自多种来源，包括基础设施、主机、服务、云平台和外部来源。
- **分布式追踪：** 当事务或请求流经应用程序时的活动，并显示服务如何连接，包括代码级别的详细信息。
- **用户体验：** 通过在应用程序上添加特定数字体验的由外而内的用户视角，扩展了传统的可观测性遥测技术，甚至在预生产环境中也是如此。

谷歌云的 DevOps 研究与评估 (DORA) 发布了一组指标，组织可以利用这些指标来定量评估和衡量绩效。

- **部署频率：**部署频率决定了成功生产的速度。提高部署频率支持每天多次的快速部署变更，而不是每周或每月发布一次。组织可以通过持续交付，以及允许自动化测试和快速反馈的持续集成流水线（Pipeline）来提高部署速度。
- **变更交付时间：**此指标定义将变更提交给生产环境所需的时间。自动化性能测试和小规模迭代变更是改进总体交付周期的实用方法，使组织通过加快反馈能够更快地修复缺陷。基于主干的开发也能缩短交付周期，通过在基于功能的独立分支上工作来自动化测试。

- **变更失败率：**部署到生产环境失败的比率，以百分比表示。缩短交付周期的方法同样适用于变更失败率，例如自动化测试、基于主干的开发以及小批量发布变更。提高部署频率可实现高效的缺陷检测和更快的修复。故障报告和通知可以修复错误并确保新版本符合安全要求。

- **恢复服务的时间：**这一指标被定义为从生产故障中恢复所需的时间，使企业能够在故障发生后不久就迅速从故障中恢复。。缩短恢复服务的时间还允许更快地回滚导致故障的任何变更。通过云原生监控和告警进行持续性能管理，可以在发生故障时通知利益相关者。

以 Dora 指标作为组织绩效的基础，有助于提高 IT 运营的效率和效益。将 DORA 等衡量标准应用于云应用程序和基础设施，可以帮助组织了解云系统的性能和弹性，组织可以通过围绕这些指标的报告获得更多的可见性和洞察力。组织受益于以指标为中心的绩效管理战略，此策略可进行量化评估，以深入了解工作负载事务和业务工作流的效率。指标使组织能够了解质量缺陷并改进工作负载的发布周期，包括其整体弹性和可用性 [120]。

实现监控和可观察性系统允许跟踪内部指标来确定性能。虽然我们将在“支柱 6:测量、监控、报告、行动”中探讨可观察性指标的安全场景和用例，但谷歌云识别的一些示例指标提供了如何进行此分析的指示 [121]:

- **对监控配置所做的变更：**每周对包含监控配置的存储库进行多少次拉取请求或变更？

- **非工作时间”告警：**夜间处理告警的百分比是多少？虽然一些全球企业采用了“全天候 ”的支持模式，使这一问题变得无关紧要，但这可能表明对导致失败的主要指标没有给予足够的重视。定期的夜间告警可能会导致对告警疲劳和团队疲惫。

- **团队告警平衡平衡：** 如果团队位于不同位置，这些告警是否由所有团队进行公平分配和处理？

- **误报：** 有多少告警最后没有任何操作，或被标记为“按预期工作-正常情

况”？

- **漏报：** 有多少系统故障发生时没有发出告警或告警晚于预期？事后检查包括添加新的（基于症状的）告警的频率如何？
- **告警创建：** 每周创建多少个告警（总数，或按严重性、团队分组）？
- **告警确认：** 在约定的截止时间（如 5 分钟、30 分钟）内确认的告警百分比是多少？
- **告警静默和静默持续时间：** 每周有多少告警处于静默或抑制状态？有多少被添加到这个池中，有多少被删除？
- **无法采取行动的告警：** 被视为“无法采取行动”的告警的百分比是多少？也就是说，发出告警的工程师不能立即采取一些行动，要么是因为无法理解告警的含义，要么是因为已知的问题。无法采取行动的告警是众所周知的麻烦来源。
- **可用性：** 告警、操作手册、仪表板。有多少图表和仪表板？每张图有多少行？团队能理解这些图表吗？是否有介绍文档来帮助新工程师？人们必须查找和浏览大量内容才能找到他们需要的信息吗？工程师能否有效地关联从告警到操作手册再到仪表板？告警的命名方式是否为工程师指明了正确的方向？随着时间的推移，这些可以通过对团队的调查来衡量。
- **平均检测时间，平均响应时间，影响：** 底线是检测时间、解决时间和影响。考虑测量业务中断影响客户的时间乘以受影响的客户数量的“曲线下面积”。这可以用工具更精确地估计或完成。

对于监控和指标收集，组织可以集中使用 AWS CloudWatch、GCP Operations Suite 和 Azure Monitor 等云原生解决方案来收集、访问和关联性能指标可见性，并了解性能问题。通过应用云原生监控指标，可以对云应用程序和平台的控制流进行细粒度监控，包括单个业务的时间控制。

通过跟踪指标，可以更好地了解监测和可观察性。按产品、运营团队或其他方法对这些衡量指标进行细分，将有助于深入了解产品和团队的运行状况和安全状况。实现监测和可观察性的组织可以使用开源工具，如OpenTelemetry、Jaeger 和 Zipkin。组织可以利用商业工具根据定义好的指标来改进绩效管理。DataDog、DynaTrace 和 Splunk 是具有可观察性、提供安全相关事件、包括容器在内的云集成和资产发现的示例商业解决方案。

DevSecOps 工作组（WG）将在“ 第六支柱:测量，监控，报告，以及行动 ”中提供一份关于 DevSecOps 监控指标主题的白皮书。该白皮书描述了在 DevSecOps 环境中要监控的一些最关键的指标。它将包括衡量修复漏洞和从事件中恢复的平均时间、软件和基础设施可重复性、安全控制量和深度防御控制平均值的指标。

8.2.2 攻击面管理

攻击面管理是一门新兴的网络安全学科，是指发现、识别和管理组织的IT 系统所带来的风险所必需的流程和技术。攻击面管理包括外部攻击面管理（EASM）和内部攻击面管理（IASM），前者涉及在组织面向互联网的资产上执行攻击面管理活动，后者包括在只能从组织内部访问的 IT 资产上执行攻击面管理活动。攻击面由四个主要组件组成[122]:

- 本地部署资产： 服务器和硬件.
- 云资产： 云资源，如服务器、工作负载、SAAS 应用程序或云托管数据库.
- 外部资产： 存储和处理数据或与内部网络集成的外部供应商资产.
- 子公司网络： 由多个组织共享的网络（例如，在合并或收购的情况下）

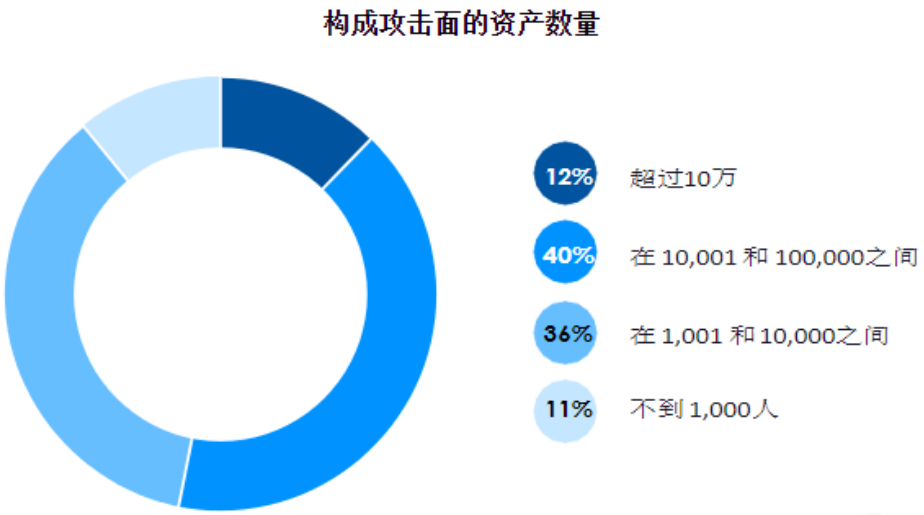

图 25: 攻击面上的体量 [123]

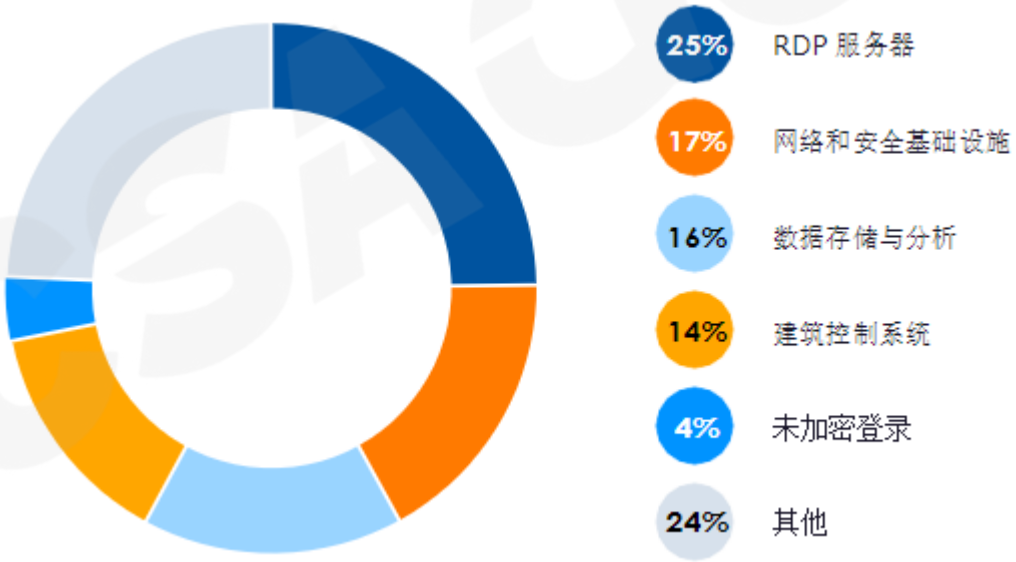

图 26: PaPalo Alto，全球攻击面的风险 [124]

攻击面管理通常包括以下功能:

- **发现：**枚举所有基础架构、网络连接、应用程序、API、凭证、数字证书和域，它们是组织技术堆栈的一部分，包括与第三方服务、组件和 API 的依赖关系。

- **清单和分类：**建立组织 IT 资产的清单，并根据各种标准（包括资产类型、技术属性、业务关键性和合规性要求）对其进行分类和优先排序。
- **风险分析和评分：**将资产信息与威胁情报信息和资产分类相关联，以计算风险评分。
- **持续安全监控：**资产的持续监控包括监控和分析来自资产的日志和安全事件，以检测可疑和恶意行为以及漏洞。关键是不断评估解决方案的所有方面，而不仅仅是执行定期分析。
- **补救：**补救风险资产和暴露资产中的弱点。重要的是与产品团队和技术负责人分享调查结果和补救措施，以提高对问题的理解和认识，从而不断改进设计。

将安全调查结果与可能丢失的数据进行优先排序，对安全调查结果进行优先排序，有助于将人员集中在风险较高的调查结果上，并确定部署流水线和流程中可能需要改进的领域。通过假设攻击者的思维方式并模仿他们的工具集，组织可以提高对所有潜在攻击向量的可视性。有效的攻击面管理工具的优势可以使组织能够:

- 自动执行资产发现、评审和补救
- 识别并禁用影子 IT 资产和其他以前未知的资产
- 修复已知的弱点，如弱密码、错误配置和过时或未修补的软件

组织可以通过观察组织网络边界以外的资源，在网络罪犯发现时持续监控资产并识别漏洞。可以使用可用于攻击面管理的许可工具和服务，示例包括 CyCognito 和 Palo Alto Cortex Xpanse。

攻击面管理解决方案定期持续监控组织的外网 IP 地址空间和连接组织的 IP 地址空间。这包括识别创建为影子 IT 形式的恶意资产，如具有对组织构成风险的使用默认账号密码的公开服务器（例如，Jenkins 服务器、AWS

S3 存储桶）。虽然攻击面管理是通过工具实现的，但需要注意的是，组织的攻击面将随着设备的不断添加、新用户的引入和业务需求的变化而不断发展。

8.2.3 事后刨析

项目事后刨析是一种仪式，用于确定项目失败、时间或重大业务损害停机的原因，并评审未来的预防措施。项目的事后刨析旨在为流程改进提供信息，以减轻未来的风险并促进迭代最佳实践。

事件事后刨析将团队聚集在一起，对事件进行更深入的研究，并找出发生了什么，为什么会发生，团队如何应对，以及可以采取哪些措施来防止事件重复发生并改进未来的应对措施。整个过程应避免指责个人，避免陷入互相指责的游戏中。在事后刨析过程中，应假设每个团队和员工都根据他们当时所掌握的信息，以最好的意图行事。事后刨析并不是找出并惩罚人为错误的根源，而是专注于不断提高绩效。

Atlassian 的事故管理手册

当事情出错时，寻找替罪羊是人类的自然倾向。避免这种情况符合 Atlassian 的最大利益，所以当你进行事后刨析时，你需要有意识地克服它。我们认为员工的出发点是好的，从不责怪别人的过错。事后刨析需要诚实和客观地检查导致故障的情况，这样我们才能找到真正的根本原因并减轻它们。

事后刨析包括事件影响、缓解措施和根本原因，关键部分是防止事件再次发生的后续行动。事后刨析的目标是通过确保充分了解所有根本原因和相应的预防措施来不断学习和改进。团队有一定的内部灵活性，但常见的事后触发器包括:

- 用户可见的停机时间或降级超过某个阈值
- 任何类型的数据丢失
- 随叫随到的工程师干预（释放回滚、流量重新路由等）

- 高于某个阈值的分辨率时间
- 监控失败（通常意味着手动事件发现）

Google 的事后刨析策略:

- 将事后刨析简化纳入工作流程。除了有助于确定什么情况下应该启动事后刨析外，几个完整及成功的事后刨析测试期可能有助于证明它们的价值。
- 通过前面提到的社会方法以及个人和团队绩效管理，确保撰写有效的事后刨析报告是一种可以被公开奖励和庆祝的做法。
- 获取高层领导的认可和参与。

尽管事后刨析流程确实存在时间或精力方面的固有成本，但它提供了持续改进工作方式的好处，培养了一种积极的文化，有助于激励透明度、创新和变革，并改善了团队之间的沟通渠道。

谷歌提供了一个事后处理的示例输出，而在 danluu/post-mortem Github 页面的链接中可以看到停机和真实的事后事件。 其他可用的事后刨析框架包括 Atlassian 的无指责事后刨析，它也可以作为 突发事件管理流程的补充。

8.2.4 紫色团队

网络安全模拟是一种测试应对模拟网络攻击的技术和流程的方法。这通常需要一个复刻的环境来有效地“战争游戏”，以应对现实场景中的潜在攻击。这就产生了所谓的不同颜色的团队。在这一部分，重点是红队和蓝队，以及他们的组合，这产生了一个紫色的团队。在这种情况下，红队是负责进行渗透测试、漏洞评估和全面模拟网络攻击的进攻团队。有时，这些模拟不会通知蓝队，即防守专家小组。该小组由组织的安全运营中心的成员组成，是安全事件的第一级响应和识别。相反，紫色团队是一种方法，在这种方法中，

两个团队的成员聚在一起，以确保两个团队可以相互学习，增加练习对组织的价值。

紫队演习可以在两个不同的阶段进行 [125]:

- **在测试时间紫色演习**

在这种情况下，如果红队变成紫队演习，可以进行不同类型的紫队演习:

- **捕获和释放：** 这种类型的测试是测试结束阶段的最佳选择，在这种情况下，红队已经多次被蓝队捕获，并且不再可能变更 TTP。在这种情况下，在蓝队和红队之间建立了一个沟通渠道（由负责管理红队演习的白队主持）。利用这个沟通渠道，蓝队将在确定红队活动时传达 IOC 被攻破指示。假设这些是正确的，如果特定的计算机或帐户已被阻止，则可以将其作为测试的一部分进行解锁。
- **共同协作的 POC 方法验证：** 当无法进行特定测试或场景时， 例如，由于系统超出范围或在关键系统中具有高风险的影响，协同方法可以允许红队和蓝队在考虑所有进攻和防守方面的情况下进行攻击的概念验证。
- **“战斗状态”：**特定类型的紫色团队测试，其中蓝队是完全知道红队的存在和他们的目的。在这些场景中，特定的标志被分配给目标资产，红队必须捕获这些标志，而蓝队必须保护系统。

- **测试结束时的紫色团队**

每当在红队训练结束时再进行紫队训练时，这有几种可能性:

- **桌面讨论：** 在这种情况下，蓝队、红队以及管理层（可选）进行潜在场景的讨论和解决。进行这种讨论的方式可以是特定场景的角色扮演、理论性讨论以及业务连续性讨论。
- **测试场景的重新探索：** 在这种情况下，红队分析过的一些场景可以作为蓝队的演练重新访问，以了解安全漏洞存在的位置以及如何制定和实施潜在

的缓解措施。

- **探索新的场景：** 如果红队在测试结束期间有新的想法，与蓝队合作探索这些新场景可能会带来有趣的结果。这个想法是与蓝队一起走一遍潜在的新场景，填补起初测试未解决的空白。

通过有效的团队结构和流程成功实现紫队合作，并不意味着取代红队行动——测试结果保密，蓝队不被告知以增加模拟的真实感。然而，在特定情况下，两个团队相互学习可能更有益。例如，当红队无法逃避被识别和理解时，了解 SOC 结构可以帮助他们改进方法以获得更成功的模拟测试，或者当蓝队根本无法捕捉到红队的活动时。紫队合作行动将帮助通过理解红队的战术、技术和程序（TTP）来提高蓝队的检测能力。

8.2.5 漏洞管理（识别后）

随着网络攻击的增长和与网络相关的泄露事件的公开，尤其是在持续快速实施变化的 DevSecOps 环境中，有了对这些实践采取的进一步需求。为了应对这些挑战，请考虑以下作为您漏洞修复流程的一部分的步骤。这些步骤需要根据您的环境进行微调，并通过吸取教训不断改进:

- 识别：通过基础设施和应用测试以及持续漏洞扫描来识别漏洞。考虑监控公共和供应商的安全通告以及网络威胁情报，以识别与组织相关的其他漏洞。
- 评估： 评估、确定和分类漏洞，以确定其适用性和严重性。这将使组织能够优先处理漏洞，并将治疗工作集中在更重要的风险上。
- 整改： 通过修复、配置变更、代码修改或流程变更来修复影响系统的漏洞。对于可以减轻风险的情况，制定相应的对策，以便在无法完成修复的主动攻击事件中快速部署。
- 报告： 收集数据指标，并将其与用于管理报告的既定指标进行比较。通过评估

您的技能能力和可用的技术解决方案，持续评估漏洞修复流程的有效性。

图 27：CSA 漏洞整改框架

虽然编码和测试阶段的安全活动可以帮助识别应用程序和基础设施中的漏洞，但漏洞修复会评估、优先处理在网络、系统和应用程序组件中找到的缺陷和弱点。

修复漏洞可能具有挑战性，并且可能被视为敏捷开发方法的反模式。传统的补丁方法通常是手动的、耗时的和昂贵的，需要大量的专业知识来评估修复对系统的影响。DevSecOps 原则通过提供持续测试软件和嵌入式安全性以及质量的方式，彻底改变了这些实践。修复操作旨在减轻漏洞被利用的风险和可能性。它是更广泛的漏洞管理流程的重要组成部分，该流程从初始识别到风险分类、优先级、修复和报告中管理漏洞的整个生命周期。这些漏洞可能在开发活动中产生，并表现为需要补丁的软件缺陷或需要解决管理操作的配置问题。

漏洞修复与其他实践相关联，例如攻击面管理。在整体攻击空间和利用可能性的背景下，应优先考虑修复漏洞的工作。团队应分工并专注于网络层、基础设施（虚拟机、容器、操作系统）和应用层（应用程序、API、微服务）。如本文其他部分所述，补救措施尽可能实现自动化。

并非所有漏洞都是相同的。通过向左倾斜安全性，团队可以及早而频繁地识别和修复漏洞，并在同一开发迭代中使用 SAST 等工具加以解。

同样，通过云安全委员会（CSPM）、动态应用安全测试（DAST）或渗透测试等手段识别漏洞时，需要在评估发现和确定风险概况和可利用性时进行更多的分析和严谨处理。在分类之前评估漏洞，然后就针对每个已识别的漏洞采取行动之前，以下措施至关重要:

- **修复：** 根据漏洞的严重程度、复杂性和依赖关系，确定是否应该修复漏洞，以评估修复的影响和可行性。对于无法修复的严重漏洞，请考虑进行系统重设计或更换，以消除与漏洞相关的风险。

- **接受：** 确认存在漏洞，并评估使用风险接受方法是否适用于该漏洞。影响这一决策的因素可能包括成本、补丁可用性或技术资源。

- **调查：** 进一步调查漏洞的适用性、严重程度和被利用的可能性，以更好地理解与漏洞相关的风险以及可能的修复方案和技术复杂性。

一个健壮的漏洞修复流程对于降低组织的风险概况至关重要。实施和执行强大的漏洞修复流程的关键驱动因素包括:

- **安全优势：** 允许高效评估、优先处理和修复所在环境中发现的漏洞，减少您的组织被网络攻击入侵的风险，避免业务中断、财务损失和声誉受损。此实践还可以识别其他改进领域，如补丁管理和资产管理，推动技术改进。

- **合规性：** 作为大多数安全最佳实践行业标准、指南和框架（包括 CIS、PCI-DSS、ISO27001、GDPR、HIPAA 和 FFIEC）的一部分的强制性要求。一个强有力的流程对于避免因未履行义务而面临财务处罚，并降低受到安全漏洞影响的可能性至关重要，因为可能会在声誉受损的基础上带来额外的罚款。

对漏洞的管理和修复被视为减少受到威胁的可能性的关键措施，因此是

大多数监管机构的常见合规要求。在当今环境中，通过零信任的安全方法，修复漏洞变得更加重要，该方法旨在将焦点从网络边界转向资源的主动保护。DevSecOps 实践的持续变化和敏捷开发要求采用强大的漏洞修复流程以降低引入漏洞的风险。请考虑以下框架和指南来管理、高效进行分类和修复漏洞，并最小化运营中断:

- **CISA** **CRR 第 4 卷漏洞管理：** 该指南由美国国土安全部的网络安全评估计划（CSEP）制定，旨在帮助组织管理运营风险并实施成熟的网络韧性评审（CRR）流程。漏洞管理活动采用了基于 CERT-RMM 的方法，并侧重于识别、分析和管理漏洞的过程。这包括定义漏洞解决策略的步骤，并将其转换为具有规则和指导方针的结构性计划，以减少暴露和缓解漏洞。

- **Hyperproof** **漏洞管理指南：** 该指南强调了漏洞管理对企业安全和合规性的重要性。它鼓励组织采用有组织、协调和循序渐进的方法有效地管理整个组织的漏洞。指南包括七个组成部分：建立基本准则、资产、配置和补丁的管理，以及通过扫描和渗透测试识别和评估漏洞。文章还提供了增强补丁管理流程的九个技巧，并列出了一些可以帮助您识别漏洞的漏洞扫描器。

- **NCSC** **漏洞管理：** 该指南旨在帮助组织评估和优先处理漏洞，并根据可用资源和资金确定最相关的补丁。该指南侧重于管理广泛可用的软件和硬件中的漏洞，主要是部署补丁和修改弱配置，而不是解决小众的软件问题。其中包括指标和影响漏洞分类和优先级方法的指标和因素。

- **NIST** **企业补丁管理计划指南：** 该指南提供通过企业补丁管理进行预防性维护的建议。它提供了简化和操作化打补丁过程的建议，以提高风险降低和帮助防止系统受损、数据泄漏和中断。该过程包括识别、确定优先级、获取、验证和监控补丁、更新和升级的步骤。推动保护业务资产，避免昂贵的故障，并可靠地实现组织的使命。

- **OWASP** **漏洞管理指南：** 该指南将漏洞管理分解为三个可管理和可重复的功能：检测、报告和修复。这些功能由四个主要流程组成，可以视为包含待办事项的任务清单。该文档提供了建议，逐步不断完善每个任务以适应您的

目标，并使您的组织更具弹性。

8.2.6 事件响应

预防性安全控制无法完全消除关键数据在网络攻击中被破坏的可能性。因此，组织必须确保建立可靠的事件响应流程和策略。事件响应是一种有组织的处理和管理安全漏洞或网络攻击后果的方法。其目标是以减少损害、缩短恢复时间和降低成本的方式处理情况。

事件响应 126 可以定义为在云环境中管理网络攻击的流程，包括四个阶段（如**图 28** 所示):

- 阶段 1: 准备
- 阶段 2: 检测和分析
- 阶段 3: 封堵、清除和恢复
- 阶段 4: 事后总结

图 28: CSA 事件响应

云安全事件响应系统与非云安全事件响应系统有几个关键方面的区别，尤其是在治理、责任共担和可见性方面。

- **治理：** 云中的数据存储在多个位置，还可能使用了不同的云服务提供商。将各个组织聚集在一起调查事件是一个重大挑战。这也消耗了拥有庞大客户端池的大型云服务提供商的资源。

- **责任共担：**云服务客户、CSP 和/或第三方供应商在确保云安全方面都承担着不同的角色。通常情况下，客户对其数据负责，CSP 则对其提供的云基础设施和服务负责。云安全事件响应应该始终在各方之间进行协调。根据所选的云服务模型，例如软件即服务 (SaaS)、平台即服务 (PaaS) 和基础设施即服务 (IaaS)，CSP 和 CSC 之间的责任共担领域也有所不同，这个概念必须充分理解。例如，在 IaaS 中，操作系统 (OS) 的管理由 CSC 承担。因此，操作系统的事件响应责任也由 CSC 承担。组织应具有一致而明确的多云战略/框架，与 CSC、CSP 和/或第三方

云服务提供商进行协作。任何采用单一 CSC、CSP 或第三方云服务提供商的“一体化”(all-in)战略的组织，在服务提供商端发生中断的情况下，都会间接引入单一故障点。提供云服务的单一 CSP 方式可能会导致 CSC/CSP 一旦出现组织无法控制的故障，组织的业务会遭受持续中断。这种情况将严重影响业务运营，并增加业务连续性计划(BCP)策略无法恢复的可能性，从而导致系统性 CIR 事件。从 CIR 视角考虑服务提供商的多样性时，还应该鼓励组织在其计划中考虑数字服务主权方面的问题（例如数据驻留、数据主权等）。

- **可见性：** 云云中缺乏可见性意味着本应快速解决的事件没有得到及时解决，并存在进一步升级的风险。如果利用得当，云可以提供更快、更便宜和更有效的 IR。CSP 及其合作伙伴已经提供了许多内置的云平台工具、信息源、服务和能力，来显著增强检测、反应、恢复和取证能力。在利用云架构而不是传统数据中心模型开发 IR 流程和文档时，必须小心谨慎。CIR 必须是积极主动的，并且在整个流程中都要抵御故障。

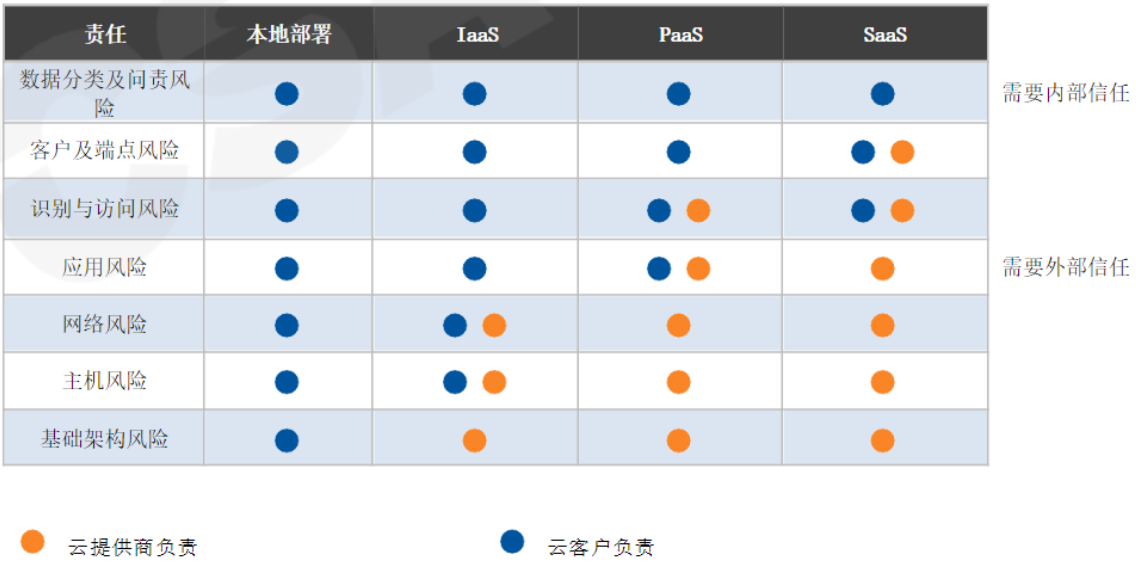

图 29：云提供商 / 客户

参考文献

文中标记的 1-126 参考文献链接请访问英文原稿链接：

https://cloudsecurityalliance.org/research/working-groups/devsecops

其他文献：

Cloud Security Alliance, The Six Pillars of DevSecOps: Pillar 4, Bridging Compliance and Development, February 2022, Available @ https://cloudsecurityalliance.org/artifacts/devsecops-pillar-4-bridging-compliance-and-development/

Cloud Security Alliance, The Six Pillars of DevSecOps: Pillar 5, Automation, July 2020, Available @ https://cloudsecurityalliance.org/artifacts/devsecops-automation/

Cloud Security Alliance,The Six Pillars of DevSecOps: Pillar 1, Collective Responsibility, February 2020, Available @ https://cloudsecurityalliance.org/artifacts/devsecops-collective-responsibility/

Cloud Security Alliance, Information Security Management through Reflexive Security: Six Pillars in the Integration of Security, Development and Operations, August 2019, Available @ https://cloudsecurityalliance.org/artifacts/information-security-management-through-reflexive-security/

Cloud Security Alliance, The Six Pillars of DevSecOps, Achieving Reflexive Security Through Integration of Security, Development and Operations, August 2019, Available @ https://cloudsecurityalliance.org/artifacts/six-pillars-of-devsecops/

Cloud Security Alliance, Cloud Penetration Testing Playbook, July 2019, Available @ https://cloudsecurityalliance.org/artifacts/cloud-penetration-testing-playbook/

Cloud Security Alliance, Cloud Threat Modelling, July 2019, Available @ https://cloudsecurityalliance.org/artifacts/cloud-threat-modeling/

Patel, S.: 2019 Global Developer Report: DevSecOps finds security roadblocks divide teams (July 2020), https://about.gitlab.com/blog/2019/07/15/ global-developer-report/, [Online; posted on July 15, 2019]

Assal, H., Chiasson, S.: 'Think secure from the beginning' A Survey with Software Developers. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. pp. 1–13. CHI '19, Association for Computing Machinery, New York, NY, USA (2019)

Gasiba, Tiago Espinha, et al. "Awareness of Secure Coding Guidelines in the Industry-A first data

analysis.” 2020 IEEE 19th International Conference on Trust, Security, and Privacy in Computing and Communications (TrustCom). IEEE, 2020.

Blomqvist, Markus, Lauri Koivunen, and Tuomas Mäkilä. “Secrets Management in a Multi-Cloud Kubernetes Environment.” (2021).

Barker, E., Barker, W., Burr, W., Polk, W., Smid, M., & Gallagher, P. D. (2020). NIST Special Publication 800-57 Recommendation for Key Management–Part 1: General.

HashiCorp, Unlocking the Cloud Operating Model: Security - What is Identity-based Security, May 2019, Available @ https://www.hashicorp.com/resources/unlocking-the-cloud-operating-model-security“

Rose, Tina, and Xiaobo Zhou. “System Hardening for Infrastructure as a Service (IaaS).” 2020 IEEE Systems Security Symposium (SSS). IEEE, 2020.

Mourad, Azzam, Marc-André Laverdiere, and Mourad Debbabi. “Security hardening of open source software.” (2006).

GitHub, Chef InSpec: Auditing and Testing Framework, August 2022, Available @ https://github.com/inspec/inspec

OWASP Foundation, OWASP Web Security Testing Guide (WSTG), The OWASP Testing Project, July 2022, Available @ https://owasp.org/www-project-web-security-testing-guide/latest/2-Introduction/

Gonzalez, Danielle. The State of Practice for Security Unit Testing: Towards Data-Driven Strategies to Shift Security into Developer’s Automated Testing Workflows. Diss. Rochester Institute of Technology, 2021.

Planning, Strategic. “The economic impacts of inadequate infrastructure for software testing.” National Institute of Standards and Technology (2002): 1.

Tómasdóttir, Kristín Fjóla, Mauricio Aniche, and Arie Van Deursen. “The adoption of javascript linters in practice: A case study on eslint.” IEEE Transactions on Software Engineering 46.8 (2018): 863-891

OWASP Foundation, OWASP DevSecOps Guideline Project, June 2022, Available @ https://owasp.org/www-project-devsecops-guideline/

GitHub, Checkstyle, August 2022, Available @ https://github.com/checkstyle/checkstyle

Douglas Crockford, JSLint, The JavaScript Code Quality and Coverage Tool, July 2022, Available @ https://www.jslint.com/

Nicholas C. Zakas, ESLint, Find and fix problems in your JavaScript code - eslint - pluggable JavaScript linter, August 2022,
Available @ https://eslint.org/
GitHub, Terraform-Linters/tflint: A pluggable terraform linter, August 2022,
Available @ https://github.com/terraform-linters/tflint

Kubelinter, KubeLinter analyzes Kubernetes YAML files and Helm charts and checks them against various best practice, August 2022,

Available @ https://docs.kubelinter.io/

PyPI, Bandit - Security oriented static analyser for python code, August 2022,
Available @ https://pypi.org/project/bandit/

PMD, PMD - An extensible cross-language static code analyzer, July 2022,
Available @ https://pmd.github.io/

Mateo Tudela, Francesc, et al. “On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications.” Applied Sciences 10.24 (2020): 9119.

Brunnert, Andreas, et al. “Performance-oriented DevOps: A research agenda.” arXiv preprint arXiv:1508.04752 (2015).

Schulz, H., Okanović, D., van Hoorn, A., & Tůma, P. (2021, April). Context-tailored Workload Model Generation for Continuous Representative Load Testing. In Proceedings of the ACM/SPEC International Conference on Performance Engineering (pp. 21-32).

Datadog, Performance Monitoring (APM), August 2022,
Available @ https://www.datadoghq.com/product/apm/
Tom Hall, Atlassian, DevOps metrics - Why, what, and how to measure success in DevOps, August 2022,

Available @ https://www.atlassian.com/devops/frameworks/devops-metrics

缩略词

AI	Artificial Intelligence 人工智能
CISA	Cybersecurity and Infrastructure Security Agency 网络安全与基础设施安全局
CSA	Cloud Security Alliance云安全联盟
CSA CCM	Cloud Security Alliance Security, Cloud Control Matrix 云安全联盟，云控制矩阵
CSA GSD	Global Security Database 全球安全数据库
CI/CD	Continuous Integration, Continuous Delivery 持续集成，持续交付
CIO	Chief Information Officer 首席信息官
CIS	Center for Internet Security 互联网安全中心
CISO	Chief Information Security Officer 首席信息安全官
CSC	Cloud Service Customer 云服务客户
CSP	Cloud Service Provider 云服务提供商
CTO	Chief Technology Officer 首席技术官
DAST	Dynamic Application Security Testing 动态应用程序安全测试
DORA	DevOps Research and Assessment

	开发运维研究与评估
EASM	External Attack Surface Management外部攻击面管理
FFIEC	Federal Financial Institutions Examination Council 联邦金融机构评审委员会
GDPR	General Data Protection Regulations通用数据保护条例
HIPAA	Health Insurance Portability and Accountability Act of 19961996年健康保险流通与责任法案
IaaS	Infrastructure as a Service基础设施即服务
IAC	Infrastructure as Code基础设施即代码
IAM	Identity and Access Management身份与访问管理
IDE	Integrated Development Environment 集成开发环境
IDS	Intrusion Detection System 入侵检测系统
INVEST	Independent, Negotiable, Valuable, Estimable, Small, Testable 独立的、可协商的、有价值的、可估算的、小的、可测试的
ISO	International Standards Organization 国际标准化组织
KPI	Key Performance Indicator 关键绩效指标
ML	Machine Language 机器语言
NCSC	National Cyber Security Center 国家网络安全中心
NIST	National Institute of Standards and Technology 国家标准与技术研究院
OPA	Open Policy Agent开放策略代理
OS	Operating System操作系统
OWASP	Open Web Application Security Project开放式Web应用程序安全项目
PaaS	Platform as a Service平台即服务
PCI-DSS	Payment Card Industry Data Security Standard 支付卡行业数据安全标准
PII	Personally Identifiable Information个人身份信息
RBAC	Role-based access control基于角色的访问控制
REST	Representational state transfer表述性状态转移

SaaS	Software as a Service软件即服务
SAST	Static Application Security Testing静态应用程序安全性测试
SBOM	Software Bill of Materials软件物料清单
SCA	Software Composition Analysis 软件组成分析
SCM	Supply Chain Management供应链管理
SCR	Source Code Repository源代码仓库
SDLC	Software Development Life Cycle软件开发生命周期
SLA	Service Level Agreement服务水平协议
SIEM	Security information and event management安全信息与事件管理
SOAP	Simple Object Access Protocol简单对象访问协议
SOAR	Security Orchestration, Automation and Response 安全编排、自动化和响应
SOC	System and Organization Controls 系统与组织控制
SOC 2	System and Organization Controls Two系统与组织控制二
SOD	Segregation of Duties 职责分离
SPII	Sensitive Personally Identifiable Information敏感个人身份信息
VPC	Virtual Private Cloud虚拟专用云
VPN	Virtual Private Network虚拟专用网络
VM	Virtual Machine虚拟机
XDR	Extended detection and response扩展检测和响应

CSA GCR cloud security alliance
GREATER CHINA REGION ®

Cloud Security Alliance Greater China Region

扫码获取更多报告

云安全联盟大中华区
官网：https://c-csa.cn 电话：0755-86548359
地址：上海市浦东新区东育路255弄5号3FB30（前滩国际经济组织集聚区）/
广东省深圳市南山区粤海街道北京航空航天大厦1号楼905