

AI驱动的SOLC及软件供应链 安全技术应用指南

——立足AI原生，构建安全可信的AI开发生态



版权声明

本报告为北京谷安天下科技有限公司（以下简称“本公司”）旗下媒体平台安全牛研究撰写，报告中所有文字、图片、表格均受有关商标和著作权的法律保护，部分文字和数据采集于公开信息，所有权为原著者所有。未经本公司书面许可，任何组织和个人不得将本报告内容作为诉讼、仲裁、传媒所引用之证明或依据，不得用于营利或用于未经允许的其他用途。任何未经授权的商业性使用本报告的行为均违反《中华人民共和国著作权法》及其他相关法律法规、国际条约。未经授权或违法使用者需自行承担由此引发的一切法律后果及相关责任，本公司将依法予以追究。

免责声明

本报告仅供本公司的客户或公司许可的特定用户使用。本公司不会因接收人收到本报告而视其为本公司的当然客户。任何非本公司发布的有关本报告的摘要或节选都不代表本报告正式完整的观点，一切须以本公司发布的本报告完整版本为准。

本报告中的行业数据主要为分析师市场调研、行业访谈及其他研究方法估算得来，仅供参考。因调研方法及样本、调查资料收集范围等的限制，本报告中的数据仅服务于当前报告。本公司以勤勉的态度、专业的研究方法，使用合法合规的信息，独立、客观地出具本报告，但不保证数据的准确性和完整性，本公司不对本报告的数据和观点承担任何法律责任。同时，本公司不保证本报告中的观点或陈述不会发生任何变更。在不同时期，本公司可发出与本报告所载资料、意见及推测不一致的报告。

本报告所包含的信息及观点不构成任何形式的投资建议或其他行为指引，亦未考虑特定用户的个性化需求或投资目标。用户应结合自身实际情况独立判断报告内容的适用性，必要时应寻求专业顾问意见。报告中涉及的评论、预测、图表、指标、理论等内容仅供市场参与者及用户参考，用户需对其自主决策行为负责。本公司不对因使用本报告全部或部分内容所产生的任何直接、间接、特殊及后果性损失承担任何责任，亦不对因资料不完整、不准确或存在任何重大遗漏所导致的任何损失负责。

引言

随着大模型与智能编码工具在研发流程中的深度渗透，软件开发正迈入“AI 协同开发”的新阶段，传统的指令、语法驱动的代码开发范式，开始向低代码、自然语言驱动转型——用聊天的方式写代码。各类 AI 编程助手、AI 编程智能体以及 AI 原生集成 IDE 等低代码开发工具（如 Cursor、OpenAI Codex、Claude Code、Gemini CLI、AutoML 平台等）逐渐成为软件生命周期的核心载体，其能力已经从代码补全，对话式代码生成，发展为自主写代码并调整修复的 Coding Agent。据 Gartner 预测，到 2028 年，90% 的企业软件工程师将使用 AI Code Assistants。

但在代码开发效率显著提升的同时，模型的原生漏洞、幻觉依赖、供应链投毒、影子 AI 治理等新风险也在集中爆发。2026 年以来，Cline CLI 供应链攻击、Claude Code 51.2 万行代码意外曝光等事件接连爆发，恶意 npm 包窃取密钥、Prompt 注入攻击等隐患持续蔓延，AI 开发环境已沦为网络安全的高危攻击面。

政策标准层面，国家陆续出台人工智能、智能体、软件安全与供应链治理相关标准规范。《人工智能代码生成服务安全要求》国家专项标准也已于 2026 年 5 月立项，其内容直接覆盖提示词注入、代码安全与幻觉检测、插件沙箱、最小权限、供应链完整性、SBOM、命令执行控制以及 Agent 自主行为的人机介入与鉴权等问题。推动 AI 开发安全治理从原则性要求向可执行、可评估、可验证、可审计的标准体系演进。

AI 开发环境下，软件供应链边界已被重新定义。企业该如何保障代码安全？如何管控开源依赖风险？AI 生成代码是否可审计、可追溯？为帮助开发者厘清 AI 开发的风险体系，沉淀可落地的治理方案，安全牛携手行业厂商，正式发起《AI 开发环境下的代码及供应链安全技术应用指南》报告调研和研究活动。本报告将立足合规要求与行业实践，聚焦 AI 开发场景，系统拆解代码安全与供应链管理的痛点、难点与解决方案，为行业和企业 AI 供应链安全提供技术参考。

关键发现

- **开发范式：**随着大语言模型（LLM）的快速发展，应用软件开发开始从传统“人工开发”演进为“AI 开发（AI Coding）”。根据产业发展，AI Coding 大致可以划分为：智能代码补全、AI 辅助开发、智能体自主开发、AI 原生软件工程四个典型发展阶段，每个阶段对应不同的技术能力、代表产品和研发模式。
- **安全风险：**AI 开发模式改变了传统软件供应链的安全边界，使软件供应链风险进一步向模型、数据、Agent 和开发环境扩展。为方便分析，报告将其分为 AI 编码原生漏洞、开发环境、供应链依赖组件三个典型风险类型。
- **关键技术：**AI 开发安全的技术体系正在从传统的软件安全检测能力，扩展至覆盖模型、数据、代码、Agent 和供应链的全过程安全保障能力。技术上，AI 开发过程安全、代码可信共识验证、AI 供应链安全是保障 AI 开发与供应链安全的重要技术基础。
- **市场规模：**AI 编码与供应链安全市场正处于快速发展阶段。相关研究显示：全球 AI 代码工具市场规模预计从 2025 年的约 78 亿美元增长至 2034 年的 700 美元，年复合增长率超过 27%。AI 代码生成、智能补全、自动测试和代码优化等能力正在成为软件研发基础设施的重要组成部分。
- **产业格局：**当前的产业结构和竞争格局也发生了明显变化。典型地形成了代码安全厂商、综合网络安全厂商与 AI 大模型厂商融合发展的三大竞争格局，覆盖 AI 辅助编码、AI 代码审计、AI 供应链治理、Agent 安全的四大核心产品方向。
- **监管层面：**全球范围内，围绕 AI 开发和供应链的安全监管体系都在持续完善。整体趋势表现为从单纯的“已知漏洞风险防御”，演变为“涵盖代码、数据、模型、工具链的全生命周期透明度的 AI-SBOM 与合规审查”。
- **趋势预测：**AI 正在重塑软件研发模式，也正在重新定义软件供应链安全边界。未来，安全体系将从传统 DevSecOps 演进至 AI DevSecOps，并进一步迈向 Agentic SDLC 时代，形成覆盖代码、模型、数据、Agent 和供应链的统一可信开发安全体系。

目 录

第一章 AI 驱动的软件开发变革与治理挑战

1.1 AI 协同开发范式的演进与应用现状	1
1.2 AI 开发范式带来的安全治理挑战	6

第二章 AI 编码及软件供应链的风险特征

2.1 AI 生成代码的核心风险与攻击特征	10
2.2 AI 开发环境引入的新风险影响	15
2.3 隐性依赖与组件来源不透明风险	18
2.4 2026 年典型安全事件复盘与启示	19

第三章 AI 开发与供应链安全技术

3.1 AI 开发过程的关键安全技术	24
3.2 AI 代码可信共识验证机制	27
3.3 AI 时代的软件供应链安全技术	29

第四章 产业格局及国内外代表性厂商

4.1 市场规模与产业格局变化	33
4.2 国外代表性厂商及产品方案	37
4.3 国内代表性厂商及产品方案	39

第五章 企业 AI 驱动 SDLC 安全建设指南

5.1 AI SDLC 安全建设的六项基本原则	46
5.2 企业 AI 开发安全成熟度评估模型 (AI-DSMM)	49
5.3 企业自评指南与建设优先级建议	53
5.4 AI 开发平台安全建设	54
5.5 AI DevSecOps 建设	57

第六章 行业实践及厂商案例分享

6.1 三大典型行业 AI 代码安全分析与实践	62
6.2 安普诺 (悬镜安全): 依托灵脉 CODE AI 落地数字供应链 AI 原生代码全生命周期闭环治理实践	63

第七章 趋势预测

7.1 监管政策趋势分析	70
7.2 市场发展趋势预测	73

参考资料	76
------------	----

第一章 AI 驱动的软件开发变革与治理挑战

2025 年以来，Vibe Coding（氛围编程）作为 AI 开发领域的热门概念开始风靡全球，AI 原生 IDE 和 Coding Agent 加速进入研发环境。传统的软件开发体系已被深度集成了大语言模型（LLM）与自主智能体（Agentic AI）的全新范式所取代。相关统计数据表明，在先进的开发场景中，已有高达 65%~70% 的生产环境代码由 AI 自主生成或辅助编写。AI 不再仅是提供语法补全的辅助工具，而是演变为具有自主决策能力的协作主体。

这一现象背后，是生成式人工智能驱动的软件开发范式革命。标志着软件开发开始进入“人类负责决策、AI 负责执行”的新阶段，软件生产主体从开发人员扩展为“人+模型+Agent+工具链”。但同时，也带来了代码可信性、生成质量、安全可控性等新的挑战。

1.1 AI 协同开发范式的演进与应用现状

1.1.1 开发范式重塑：从人工编码到智能体协作开发

过去二十余年，软件开发经历了从传统瀑布式开发、敏捷开发、持续集成/持续交付（CI/CD）到 DevOps、DevSecOps 的持续演进，其核心目标始终围绕提升研发效率、保障软件质量以及缩短产品交付周期展开。然而，这一阶段的软件开发仍然以开发人员作为主要生产主体，自动化工具更多承担编译、测试、部署等辅助角色。

随着大语言模型（LLM）的快速发展，以代码生成、代码理解、代码补全、自动测试和代码审查为代表的新一代 AI 能力逐渐融入软件研发流程，软件开发开始迈入“AI 开发”时代。

在 AI 开发模式下，大模型不再只是简单回答问题，而是作为研发过程中的“智能协作者”，能够参与需求理解、系统设计、代码编写、单元测试、文档生成、安全修复以及代码重构等多个研发阶段。开发者逐渐由传统的代码编写者转变为 AI 的任务规划者、结果审核者和工程决策者，人机协同成为新的研发范式。

AI 开发的发展并不是一次性完成的，而是伴随着大模型能力、软件工程方法论以及 Agent 技术的发展逐步演进。从发展阶段来看，大致可以划分为四个典型阶段，每个阶段对应不同的技术能力、代表产品和研发模式。

AI Coding 发展路线图

AI Coding 并非一蹴而就，而是伴随着大模型、软件工程方法论及 Agent 技术逐步演进，划分为四个典型阶段。

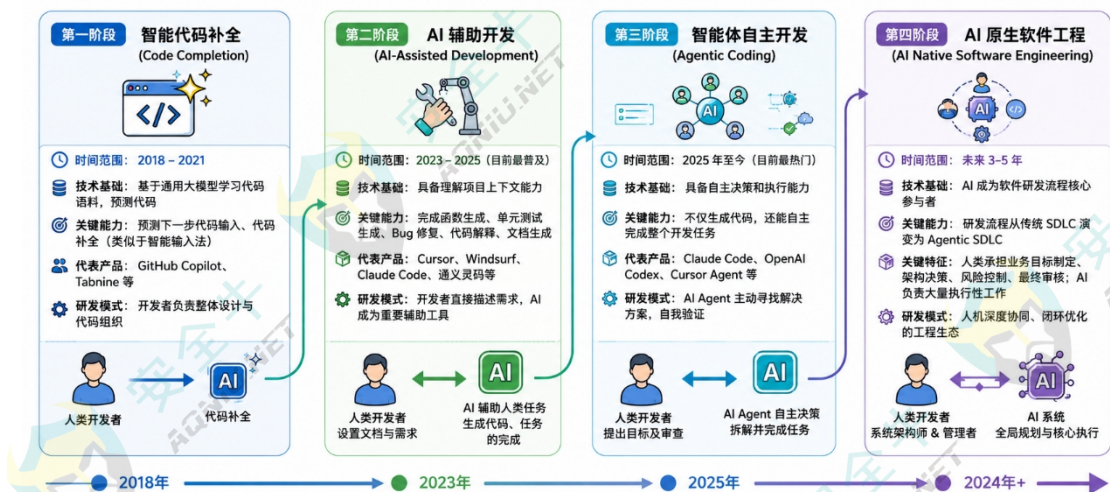


图 1 AI Coding 演进

第一阶段：智能代码补全 (Code Completion, 2018—2021)。这是 AI Coding 最早进入开发者视野的阶段。以 GitHub Copilot、Tabnine 等工具为代表，基于通用大模型学习大量代码语料对开发者下一步要输入的代码进行预测、代码补全，类似于智能输入法。开发者仍然负责整体设计和代码组织。

第二阶段：AI 辅助开发 (AI Coding, 2023—2025)。这一阶段也是目前最普及的阶段。以 Cursor、Windsurf、Claude Code 等开发工具为代表，开始具备理解项目上下文的能力，完成函数生成、单元测试生成、Bug 修复、代码解释、文档生成等任务，逐渐成为开发者的重要辅助工具。开发者可以直接描述需求。

第三阶段：智能体自主开发 (Agentic Coding, 2025 年至今)。这是目前行业最热门的发展方向。以 Claude Code、OpenAI Codex、Cursor Agent 等工具为代表，不仅能够生成代码，还能通过模型上下文协议 (MCP) 和 API 连接，自主进行需求分析、调用外部编译和部署工具、自动发起拉取请求 (PR) 乃至独立提交代码库，自主完成整个开发任务。例如，一个智能体可以根据需求自动创建项目、编写代码、执行编译、修复错误、提交代码并发起代码评审，实现从“代码生成”向“任务执行”的能力跃迁。

第四阶段：AI 原生软件工程 (AI Native Software Engineering, 未来 3~5 年)。AI 原生软件工程，AI 不再只是一个开发工具，而成为软件研发流程中的核心参与者。研发流程将从传统 SDLC 演变为 Agentic SDLC。人类更多承担业务目标制定、架构决策、风险控制、最终审批，AI 负责大量执行性工作。

Agent Coding 的出现意味着 AI 角色由“代码助手”进一步演进为“开发执行者”，开发流程开始由“人操作工具”逐渐转向“AI 操作工具、人负责监督”。

1.1.2 产业生态变革：AI 开发工具类型与生态结构

生成式 AI 不仅改变着软件开发人员编码范式，也重新塑造了软件开发工具链和生态结构。

根据技术架构不同，当前市场上的主流的 AI 开发工具，可分为：AI 增强型 IDE 平台、AI 原生 IDE 平台、Coding Agent、低代码/Agent 开发平台四大类型。

AI 开发工具类型

根据技术结构不同，当前市场主流 AI 开发工具可分为四大类型



图 2 AI 开发工具类型

(1) AI 增强型 IDE 平台

AI 增强型 IDE 平台是在传统 IDE 基础上，通过集成大模型能力提升开发效率，其核心特点是“IDE + AI 助手”。该类工具通常提供代码补全、代码解释、智能问答、代码重构、测试生成、漏洞提示等辅助能力，但整体开发流程仍由开发人员主导。

典型代表性产品，包括：GitHub Copilot、VS Code 的 AI 扩展、Amazon Q Developer 等。

(2) AI 原生 IDE 平台

AI 原生 IDE 不再简单叠加 AI 插件，而是突破传统 IDE 形态，在产品架构层面围绕大模型能力重新设计开发流程，形成以 AI 交互为核心的新型开发环境。AI 原生 IDE 代表了软件开发工具的重要演进方向，其目标并非简单提高代码输入效率，而是将 AI 能力深度融入软件工程全过程。

典型代表性产品：Cursor、Windsurf、Trae AI IDE 等。

(3) AI 代码智能体 Coding Agent

Coding Agent 是基于大模型和 Agent 技术构建的新型软件开发智能体，其目标是从辅助开发转向自主完成软件工程任务。能够自主理解软件开发任务、进行任务规划、调用开发工具、修改代码、执行测试并完成软件工程流程。

典型提供者是云厂商和大模型厂商，通常采用云端 SaaS 化交付模式。代表性产品，如 Claude Code、OpenAI Codex、Qoder、CodeBuddy。

(4) 低代码/Agent 开发平台

低代码/Agent 开发平台主要用于开发基于大模型能力的 AI 应用。该类平台通过可视化流程编排、行业知识库、预置 Agent 模板等方式，使非专业开发人员也能够快速构建智能客服、知识助手、自动化流程等 AI 应用。与传统 IDE 相比，低代码/Agent 开发平台关注的已经不是单个程序员的编码效率，而是企业级智能应用的构建、部署和运营能力。

代表性平台如，Dify、Flowise、Coze 等。

从产业生态来看，AI IDE 工具提供者已经不再局限于传统 IDE 软件厂商，而是形成由大模型厂商、云计算厂商、开源社区、开发者平台以及行业 AI 企业共同参与的新型生态体系。

AI IDE工具产业生态体系

由大模型厂商、云计算厂商、开源社区、开发者平台和垂直行业企业共同构建的新型生态



图 3 产业生态结构

一是 大模型厂商成为 AI 开发工具的重要推动力量。随着代码大模型能力不断提升，模型提供商开始直接进入开发工具领域，通过构建编码 Agent、AI IDE 以及模型服务生态，占据软件开发入口。例如，OpenAI、Anthropic 等企业均推出面向软件开发场景的 AI Agent 产品，通过模型能力直接参与代码生成和软件工程任务。

二是 云厂商通过“云+模型+开发平台”构建完整 AI 开发生态。云厂商拥有基础设施、模型服务、开发工具链和企业客户资源，因此正在成为企业 AI 开发的重要承载平台。其发展方向已经从提供计算资源，扩展到提供覆盖模型训练、应用开发、Agent 编排、安全治理和运行管理的一体化平台。

三是 开源社区推动 AI 开发工具生态开放化发展。开源模型、开源 Agent 框架以及开发工具生态的发展，加速了 AI IDE 能力普及。开发者可以基于开源模型、插件体系和工具协议快速构建定制化 AI 开发环境。例如，围绕代码 Agent、MCP（Model Context Protocol）、RAG 框架等形成的新型开发生态，正在降低企业构建 AI 应用的技术门槛。

四是 垂直行业厂商推动 AI 开发能力行业化落地。金融、电信、制造、能源等行业由于业务复杂度高，对行业知识、安全合规和定制化能力提出更高要求。因此，越来越多行业厂商开始建设面向自身业务场景的 AI 开发平台，通过领域模型、行业知识库和智能 Agent 实现业务流程智能化。

1.1.3 市场应用现状：工具采用与应用热度

AI 开发在从“工具创新”迈向“工程重构”的同时，AI 编程工具的市场渗透速度同样令人瞩目。

- 微软披露数据显示,截至 2025 年 7 月,GitHub Copilot 累计用户已突破 2000 万,90%的 Fortune 100 企业已使用该工具。随后,截至 2026 年 1 月, GitHub Copilot 付费订阅者达到 470 万,同比增长 75%,进一步反映出其企业商业化规模的快速扩张。
- 第三方咨询机构 Gartner2025 年 7 月预测,到 2028 年约 90%的企业软件工程师将使用 AI 代码助手,高于 2024 年初不足 14%的基线。Stack Overflow 2025 年度开发者调查报告也显示,全球 84%的专业开发者已在工作中使用或计划使用 AI 编程工具,其中超过半数 (51%) 开发者每天都在使用 AI 辅助编码。放眼国内,2025 年中国 AI 代码生成市场已进入应用爆发期,开发者覆盖率迅速提升,主流厂商在产品能力上的差距正在逐步缩小。



图 4 国内外主流开发工具对比

从市场应用热度看，海外主流 AI 开发工具 GitHub Copilot，是全球应用最广泛的 AI 开发工具；其次，Claude Code，支持超长上下文，擅长百万行级项目重构，后端开发者优选，企业市场份额快速增长；Cursor AI，是 AI 原生独立编辑器，支持 GPT-4o/Claude/Gemini 多模型切换，具备全项目智能 Agent，独立开发者热门选择。

国内 AI 模型厂商、智能体平台厂商都陆续推出 AI 开发工具。其中，成品 IDE/智能体工具：Qoder、CodeBuddy、Trae，开箱即用，国内企业场景覆盖率更高；开源底层代码大模型，如 GLM-5.2、Kimi-K2.7-Code，以 API 对外输出能力，企业需自研封装，终端直接覆盖率相对较低。

1.2 AI 开发范式带来的安全治理挑战

AI 驱动的软件研发模式改变了传统软件供应链的安全边界，使代码生成主体、依赖组件来源、开发交互过程以及运行环境面临新的安全挑战。一方面，AI 生成代码可能引入未知漏洞、不安全编码模式以及隐藏逻辑缺陷，开发人员对 AI 输出的过度依赖可能进一步放大安全风险；另一方面，AI 模型、训练数据、提示词上下文、插件工具、Agent 能力组件以及第三方服务均成为软件供应链的新组成部分，攻击者可能通过模型污染、提示词注入、恶意依赖投毒、Agent 权限滥用等方式实施供应链攻击。具体体现在以下六个方面：

1.2.1 传统软件供应链边界正在被极大扩展

相比传统软件供应链，AI 开发环境最大的变化之一，是 AI 大模型及生态组件开始成为软件生产流程中的关键参与者。

在 AI 开发环境中，一个完整的软件开发任务通常需要多个 AI 组件协同完成，包括：

- 大模型：GPT、Claude、Gemini、Qwen、DeepSeek 等第三方模型 API 和云平台提供推理和代码生成能力；
- IDE 插件：Cursor、Copilot、Claude Code 等负责上下文管理；
- MCP Server：负责连接数据库、GitHub、Jira、Kubernetes 等外部工具；
- Agent 框架：负责任务规划、工具编排及自动执行；
- RAG 知识库：提供企业内部代码和文档上下文；
- 第三方模型 API 和云平台提供推理能力。

这意味着软件供应链已经由代码组件和 IDE 平台组成的“传统软件供应链”扩展为覆盖开发者、AI 模型、Agent、工具链、代码仓库、开源组件及构建环境的全新“AI 开发供应链”。组件之间普遍采用 API、MCP 协议、插件机制以及 Agent 调用链进行互联互通，形成高度动态、跨平台的依赖关系。任何一个环节被攻击，都可能影响最终生成代码的安全性和可信性。

开发人员可能直接调用第三方大模型 API，使软件供应链边界不断外延，企业难以准确识别所有依赖对象，也难以建立完整的软件物料清单和模型物料清单，进一步增加了供应链资产管理和风险治理难度。

1.2.2 动态依赖链难以被现有 SCA 工具检出

在 AI 开发环境中，大量依赖关系并非来源于开发者手工引用，而是由大模型根据 Prompt 上下文实时推荐、自动补全甚至自主生成，其生命周期具有即时性、隐式性和动态变化等特点。例如，当开发人员要求 AI 实现某项业务功能时，AI 可能自动生成新的第三方库引用、自动引入 SDK、自动推荐 Docker 镜像、自动调用 GitHub 开源项目，甚至通过 MCP 协议动态连接外部工具服务。这些依赖关系很多不会立即写入 Manifest 文件，而是在代码生成、Agent 执行、CI 流水线运行等多个阶段逐步形成，导致传统 SCA 只能识别“最终落盘”的组件，却无法识别 AI 生成过程中真实发生过的依赖关系。

AI Agent 正在逐渐具备自主规划、工具选择和多轮任务执行能力。在一次复杂开发任务中，一个 Agent 可能连续调用多个 MCP Server、代码仓库、模型服务、数据库及外部 API，其真实依赖关系已经由传统“组件依赖树”演变为“动态执行依赖图”。这种运行时形成的供应链关系具有明显的动态特征，同一 Prompt 在不同时间、不同模型版本或不同上下文下可能生成完全不同的依赖路径，使传统静态 SCA 分析方法难以实现完整覆盖。

此外，AI 推荐还容易引入“幽灵依赖”问题，即 AI 生成代码中调用了开发人员并未主动选择或充分评估的第三方组件，部分组件甚至来源于长期无人维护、存在已知漏洞或许可证冲突的开源项目。由于这些组件往往缺乏人工审核，其进入企业软件供应链的速度远快于传统治理流程，进一步扩大了供应链攻击面。

1.2.3 研发人员对 AI 生成结果过度信任

随着 Cursor、Claude Code、GitHub Copilot、Trae、CodeBuddy 等 AI 开发工具逐渐成为研发人员的“默认开发入口”，开发模式正在由“人工编写代码”转向“AI 生成+人工确认”。这一变化显著提升了开发效率，但同时也带来了典型的自动化偏见问题，即开发人员倾向于认为 AI 生成内容天然可信，而降低了独立验证和安全审查的意识。

大量实践表明，AI 生成代码通常具有良好的语法正确性和较高的可运行性，但并不意味着其符合安全编码规范。由于大模型训练数据来源复杂，模型可能继承历史代码中的安全缺陷、过时 API 调用、不安全设计模式甚至已知漏洞。例如，AI 可能继续推荐存在 SQL 注入风险的字符串拼接方式、弱随机数生成算法、硬编码密钥、禁用 TLS 证书校验或存在缓冲区越界风险的函数调用。这些代码在功能测试阶段往往能够正常运行，但却埋下了严重的安全隐患。

更值得关注的是，AI 生成结果往往具有较强的“权威感”。相比传统搜索引擎提供多个参考答案，AI 通常直接生成完整实现方案，使开发人员更容易形成“模型已经完成安全判断”的心理预期，从而减少代码 Review、安全测试和人工验证环节。一旦模型受到 Prompt Injection、数据投毒或上下文污染攻击，其生成内容可能被恶意操控，引导开发人员主动引入恶意依赖、危险命令或存在后门的代码片段，而研发人员却难以及时察觉。

对于具备自主执行能力的 AI Agent，这种风险进一步扩大。当 Agent 能够自动修改代码、自动提交 Pull Request、自动执行 CI/CD 流水线甚至自动部署生产环境时，开发人员往往只关注最终执行结果，而忽略了 Agent 在执行过程中调用了哪些工具、访问了哪些资源、生成了哪些代码以及是否引入新的供应链对象。

研发人员缺乏 AI 安全意识培训，企业缺乏透明的执行日志和可信审计机制，一旦 AI 行为异常，企业将难以准确定位问题来源。

1.2.4 缺乏统一的 AI 供应链准入与退出机制

AI 开发环境中，企业引入的 IT 资产已经扩展至基础模型、AI 编程工具、Agent 框架、MCP Server、Prompt 模板、插件生态及外部模型服务等多种新型资产，而多数企业尚未建立统一的 AI 供应链准入与退出机制。

当前，研发团队通常根据开发效率自主接入各类 AI 工具。例如，不同团队可能分别使用不同的大模型、代码助手、Agent 框架及插件市场，同一企业内部甚至可能同时存在多个模型版本和大量未经统一评估的 MCP Server。这种“自由接入”模式虽然降低了技术使用门槛，却导致 AI 开发环境快速形成“影子 AI (Shadow AI)”，企业安全部门无法准确掌握 AI 资产清单，也难以及时识别其中存在的漏洞、许可证风险、数据合规问题及供应链安全风险。

企业目前缺乏统一的 AI 供应链退出机制。当某个模型停止维护、插件曝出高危漏洞、MCP Server 存在安全缺陷或外部模型服务停止支持时，企业往往无法快速识别受影响的业务系统，也难以统一完成替换、下线及风险隔离。这导致存在风险的 AI 组件可能长期滞留在研发环境中，形成持续性的攻击入口。

1.2.5 跨组织、跨平台代码资产追溯断层

AI 辅助开发正在打破传统研发边界。一次代码生成任务可能同时涉及 IDE 插件、本地 Agent、企业知识库、GitHub 仓库、模型服务平台、MCP 工具、CI/CD 流水线以及云端部署环境，代码资产在多个组织、多个平台及多个服务之间持续流转，其形成过程远比传统软件开发复杂。

但目前多数企业的代码追溯能力仍停留在 Git Commit、版本控制和 CI 日志层面，只能记录代码“最终提交”的信息，却难以回答更加关键的问题，如：某段代码由哪个模型生成，使用了哪个 Prompt，引用了哪些上下文知识，调用了哪些外部工具，经过哪些 Agent 修改，最终由谁审核并发布，等等，这些关键证据往往分散在不同平台的日志系统中，缺乏统一关联机制。

一旦发生供应链攻击、代码泄露或安全漏洞事件，企业很难完整还原 AI 生成链路。例如，当生产环境发现存在恶意依赖时，安全团队往往无法快速判断该依赖是开发人员主动引入、AI 推荐生成，还是 Agent 自动调用外部仓库获取；当模型输出存在版权争议或许可证冲突时，也难以准确追溯其训练来源、引用数据及生成过程。这种追溯断层不仅增加了事件响应成本，也影响了安全审计、责任认定及合规管理。

1.2.6 DevSecOps 不能满足 AI 开发的安全特性

AI 正在推动 DevSecOps 向 AI DevSecOps 演进，但同时也使传统安全左移体系面临新的治理难题。

首先，AI 生成代码具有规模化、自动化特点。一个开发者每天可能通过 AI 生成数千甚至上万行代码，远高于人工编码效率。传统静态代码分析（SAST）、人工代码评审及安全测试流程难以满足如此高频、高速的代码生成节奏，安全检测能力逐渐成为研发效率提升的瓶颈。

其次，AI 生成代码具有非确定性特征。同一需求在不同时间、不同模型或不同 Prompt 下可能生成完全不同的实现方式，增加了漏洞复现、缺陷定位和代码追溯的难度。传统基于版本管理和代码 Diff 的审计方式，难以完整记录 AI 生成过程及决策依据，影响软件供应链可追溯性。

再次，AI 开发流程引入了大量新的安全检查对象。除代码本身外，还需要持续评估模型版本、Prompt 模板、知识库内容、Agent 权限、工具调用链以及外部模型 API 的安全状态，实现从“代码安全检测”向“AI 开发过程安全治理”转变。

最后，AI Agent 具备自主执行能力，能够自动调用 Git、Shell、Docker、CI/CD 流水线及云资源接口，一旦 Agent 权限配置不当或遭受 Prompt Injection 攻击，攻击者可能利用 Agent 完成代码提交、仓库篡改、凭证窃取、恶意部署等操作，导致供应链攻击由传统的软件漏洞利用演变为面向开发流程的自动化攻击。

因此，未来 DevSecOps 需要进一步融合 AI 安全能力，在现有代码安全、依赖治理和 CI/CD 安全基础上，引入模型安全评估、Prompt 安全检测、Agent 权限控制、MCP 工具可信验证、AI 生成代码审计、AI 行为监测以及供应链可信度评估等能力，实现覆盖 AI 开发全生命周期的 AI DevSecOps 治理体系。

第二章 AI 编码及软件供应链的风险特征

AI 开发独有的技术特性和开发环境，打破了传统软件开发的安全边界。其风险既包括传统代码缺陷在 AI 生成代码中的复现与放大，也包括 Prompt/Context、Agent 自主执行、Tool/MCP、非人身份和模型/数据供应链带来的新增攻击面，以及可追溯性和治理缺失。

为系统厘清 AI 编码场景下的软件供应链安全隐患，精准定位风险根源、梳理风险特征，本章节立足 AI 软件开发全链路，从 **AI 生成代码缺陷**、**开发环境**、**供应链依赖组件** 三个核心维度对 AI 开发风险进行具体分析。

2.1 AI 生成代码的核心风险与攻击特征

AI 生成代码的安全缺陷是根植于大语言模型底层的技术原理的系统性风险。多项大规模实证研究表明，AI 生成代码存在固有且持续的安全弱点，这些弱点不会随着模型规模的扩大而自动消失。

2.1.1 十类典型的 AI 生成代码风险

AI 生成代码的底层逻辑是基于预训练代码数据，在已有上下文基础上预测并生成符合语法与逻辑约束的代码序列，并不具备真正意义上的安全意识或工程判断能力。

AI 生成代码通常具有以下特点：

- **模型更倾向于生成功能正确而非安全正确的代码。**当 Prompt 仅描述业务功能时，模型通常优先满足业务需求，而不会主动增加身份认证、安全校验、日志审计等非功能性安全代码。
- **AI 倾向于复现训练语料中的历史漏洞。**如果训练数据包含大量存在安全缺陷的开源项目，模型可能学习并重复这些不安全的编码模式。
- **模型偏向输出最短路径实现。**为提高代码可读性和执行效率，模型经常省略异常处理、权限控制、输入验证等安全逻辑，导致代码存在天然缺陷。

因此，AI 生成代码在提高研发效率的同时，也会直接生成新的代码缺陷。这些缺陷即包含 AI 原生代码漏洞，也包括存量代码缺陷重构产生的次生风险。

AI 原生代码漏洞指由于大模型代码生成机制、训练数据特征、上下文理解能力不足或 AI 开发流程缺陷，在 AI 辅助编码、AI 生成代码以及 AI 驱动软件开发过程中引入的新型代码安全漏洞。特别是当模型缺乏足够上下文或训练语料存在缺失时，容易产生典型的模型幻觉现象，生成“看似正确、实际上不存在或错误”的代码内容。

这类漏洞并非传统软件代码缺陷的简单延伸，而是伴随着 AI 系统的模型能力、数据依赖、智能决策和自主执行能力产生的新型攻击面。通常发生在模型、提示词、上下文、工具调用、数据流和 Agent 行为链路中。

除直接生成新的安全漏洞外，AI 还可能将企业存量代码中的历史缺陷进一步传播和放大，形成“存量代码缺陷重构的次生风险”。

大模型训练语料来源广泛，其中包含大量历史项目、开源代码及社区示例，不同版本的代码质量和安全水平差异较大。特别是当企业内部积累了大量历史代码，而这些代码又被作为 AI 上下文输入或纳入企业私有知识库后，大模型往往会将其中的实现方式学习为“最佳实践”。当模型学习了已废弃的 API、不安全加密算法、弱随机数生成方式或过时开发框架时，可能继续沿用这些不安全的代码和编码模式。导致漏洞模式持续复制、不安全编码模式被继承、漏洞在重构过程中隐蔽传播、企业知识库污染等，并在后续开发过程中不断复用和扩散。

因此，AI 生成代码虽然符合语法要求，但并不意味着符合安全编码规范。企业不应将 AI 生成代码默认为可信代码，而应将其纳入统一的软件安全开发生命周期（SSDLC），通过 SAST、SCA、IaC 扫描、Secret 扫描及 AI 代码安全检测等工具进行自动化审查，确保生成代码满足企业安全基线要求。

结合近几年 AI Coding 工具快速普及过程中暴露的问题，安全牛从 SDLC 角度总结了十大核心风险。重点覆盖“AI 生成代码、AI IDE、Coding Agent、Agentic SDLC”带来的新型安全风险。

表 1 AI 生成代码十大核心风险

编号	风险类型	风险描述	影响
1	AI 生成代码漏洞	大模型生成代码可能存在逻辑缺陷、不安全函数调用、错误安全配置等问题，开发人员容易因“AI 可信”而直接采用	SQL 注入、XSS、越权访问、弱密码算法、敏感信息泄露
2	提示词注入风险	攻击者通过恶意 Prompt、代码注释、README 文件、Issue 内容等方式影响 AI 编码助手，使其生成恶意代码或执行非预期操作	导致代码生成失控，生成后门代码、绕过安全规则、泄露上下文信息
3	恶意依赖包与软件供应链污染	AI 编码工具可能自动推荐、生成或安装不存在或恶意的软件包，攻击者利用 AI 推荐机制投毒供应链	恶意代码植入、开发环境感染、生产系统入侵
4	幻觉代码与虚假依赖风险	AI 模型可能生成不存在的 API、函数、库或错误参数，开发人员未经验证直接使用	引入恶意组件、编译失败、安全漏洞
5	敏感信息泄露风险	开发人员向 AI 工具提交源代码、配置文件、密钥、业务逻辑等敏感信息，或者在 AI 输出中泄露训练数据和上下文信息	API Key 泄露、源码泄露、商业秘密泄露
6	不安全代码审查与错误安全建议	AI 辅助代码审查可能遗漏真实漏洞，或者错误判断安全问题，导致开发人员产生安全错觉	漏洞进入生产环境、错误修复、合规风险
7	Agent 过度权限与自主执行风险	Coding Agent 具备修改代码、调用工具、执行命令、访问仓库能力，一旦被操控可能执行危险操作	自动植入漏洞、删除代码、泄露数据、横向移动

8	上下文污染与代码库投毒	AI Coding 依赖项目上下文、文档、Issue、代码库历史，攻击者可通过污染上下文影响 AI 生成结果	AI 生成恶意提交、错误架构设计、安全策略绕过
9	身份与权限管理漏洞	Coding Agent 通常使用开发者身份访问 Git 仓库、CI/CD、云资源，如果身份权限过大，会形成新的非人身份风险	代码仓库泄露、CI/CD 供应链攻击、云资源滥用
10	缺乏可追溯性与安全审计风险	企业难以知道哪些代码由 AI 生成、使用什么模型、经过哪些 Prompt、依赖哪些组件，导致责任追踪困难	合规困难、漏洞溯源困难、安全事件响应困难

2.1.2 四类值得关注的新型攻击向量

AI 编程环境的独特架构和操作模式也催生了一系列前所未有的新攻击向量。这些攻击不同于传统的软件供应链攻击，它们针对的是 AI 编码工具的工作机制本身，利用 AI 对自然语言指令的解释特性、对上下文的过度信任，以及工具链配置的安全弱点。

值得关注的四类典型攻击向量，分别是：提示注入攻击、AI 幻觉依赖、过度代理权限攻击以及 MCP 特有攻击。

AI编程环境的四大新型攻击向量

AI编程环境的独特架构和操作模式，催生了一系列前所未有的攻击向量，带来新的安全挑战。



提示注入攻击 (Prompt Injection)：是当前最普遍且危害最大的 AI 编码环境攻击向量。OWASP 2025 年 LLM 应用程序十大安全风险将提示注入列为首要风险。直接注入发生在用户输入中嵌入恶意指令以覆盖模型的原始指令；间接提示注入则通过模型读取的网页、文档、电子邮件、票据、代码仓库或知识库中的不可信内容实现。**在编码环境中，间接注入尤为危险**——攻击者可以在开源项目的 README 文件、代码注释、Issue 讨论中植入隐藏指令，当 AI 编码工具读取这些内容作为上下文时，恶意指令被执行。2025 年，GitHub Copilot Chat 被发现存在可通过间接注入泄露私人对话数据的漏洞。

幻觉包名抢占攻击 (Slopsquatting)：是 2024-2026 年间出现的一种新型供应链攻击技术，专门针对 AI 生成代码的“幻觉”特征。USENIX Security 2025 年发表的研究分析了 57.6 万个 AI 生成的代码样本，发现约 20% 的样本引用了不存在的 Python 或 JavaScript 包。更严重的是，43% 的幻觉包名在相似提示下被一致复现，58% 在十次运行中至少重复出现一次。攻击者可以提前预测这些幻觉包名，在开发者遇到这些名称之前将其注册为恶意包。已确认的恶意 Slopsquatting 包“unused-imports”被观察到执行旨在窃取凭证和 API 密钥的安装后脚本。一个实验发现，仅上传一个无代码、无 README 的幻觉包名，三个月内就积累了超过 3 万次下载。Sonatype 2026 年报告显示，2025 年新发现的恶意开源包数量超过 45.46 万个，同比增长 75%。

过度代理权限攻击：针对的是 AI 编码智能体被授予的广泛操作权限。OWASP 2025 年将过度代理列为 LLM06，定义为 LLM 应用被赋予过多自主性、过多权限或访问强大工具而未设置适当限制。在 Agentic 编码环境中，AI 智能体通常被授予文件系统访问、代码执行、网络请求、数据库操作等广泛权限。攻击者可以通过操纵自然语言输入改变 AI 智能体的目标，导致数据外泄、输出被篡改或工作流程被劫持。OWASP 2026 年发布的 AI 代理十大安全风险将 Agent Goal Hijack 列为 ASI01，即 Agentic AI 的首要风险。

MCP 特有攻击向量：MCP 生态系统特有的攻击向量正在快速演化。2026 年 5 月，MCP 生态安全风险进一步集中暴露，公开披露的漏洞和攻击案例涉及远程代码执行、命令注入、SSRF、工具投毒、权限绕过及数据泄露等多类风险。其中，多起漏洞可通过恶意 MCP 服务器或工具调用链最终实现远程代码执行，可见 MCP 已经成为 Agent 连接外部工具和系统过程中重要的新型攻击面。

CSA 的评估指出，MCP 安全危机不是任何单一产品的编码错误，而是嵌入在每个官方 MCP SDK 中的设计默认值，通过供应链传播到所有信任参考实现的下游项目。

典型 MCP 攻击方式，如：

- **Rug Pull 攻击：**攻击者利用 MCP 工具的动态发现与更新机制，在智能体首次连接时提供看似可信的工具，待建立信任后，再恶意修改工具描述、参数、执行逻辑或可用权限，使原本可信的工具在后续运行中执行恶意操作。
- **沙箱逃逸攻击：**攻击者通过恶意 MCP 工具或利用 MCP Server 执行环境中的漏洞，突破 Agent 或工具运行时的沙箱隔离机制，获得超出授权范围的系统权限，并进一步访问主机文件、凭据或其他系统资源。

- 资源窃取攻击：攻击者通过恶意或被攻陷的 MCP Server 滥用 Sampling 机制，诱导客户端或 Agent 代表其频繁调用 LLM，消耗受害方的模型推理、Token 配额、计算资源或 API 调用额度，从而造成资源滥用、费用增加甚至拒绝服务。

2.1.3 攻击链视角的典型攻击路径

根据 AI 开发环境看，攻击者可能通过操控输入内容影响 AI 行为，通过污染模型生成结果植入漏洞，通过供应链渠道传播恶意组件，通过滥用 Agent 权限实现自动化攻击，并最终利用企业治理缺陷实现长期潜伏。

因此，攻击链路可以进一步归纳为：**输入侧攻击、模型生成侧风险、软件/AI 供应链侧攻击、Agent 执行层风险、安全治理层风险**五个核心层面。这五大攻击面可共同构成从“攻击者→污染输入→影响 AI 生成→引入恶意依赖→操控 Agent 执行→绕过治理控制→形成安全事件”的完整攻击链路。



图 5 五大攻击面

(一) 输入侧攻击：通过操控上下文影响 AI 代码生成结果

输入侧是 AI 代码生成链路的第一环，也是攻击者影响 AI 行为的主要入口。在传统软件开发中，用户输入通常只影响业务逻辑，而在 AI 开发环境中，自然语言提示词 (Prompt)、代码注释、Issue 描述、需求文档以及知识库内容，都可能成为影响模型决策的重要上下文。

攻击者可以通过操控输入内容，引导 AI 生成不安全代码、泄露敏感信息或执行非预期操作。

典型风险包括：提示注入/间接提示注入、上下文污染、恶意 Issue/代码注释；

（二）模型生成风险：AI 生成代码自身存在安全缺陷

大模型本质上是基于概率预测生成代码，其目标是生成“符合上下文的代码”，而不是保证代码一定安全。因此，即使输入环境可信，AI 生成代码仍可能存在安全缺陷。

典型风险包括：幻觉代码、不安全代码模式、错误安全建议。

（三）供应链攻击面：AI 扩大软件供应链风险边界

AI Coding 环境下，软件供应链已经从传统的“代码依赖关系”扩展为“模型→数据→插件→Agent→第三方组件→开源依赖→应用软件”。攻击者可以通过污染供应链环节，使 AI 生成或集成存在风险的软件。

典型风险包括：恶意依赖包、AI 推荐组件投毒、开源生态污染。

（四）Agent 执行面：自动化能力带来的权限滥用风险

Coding Agent 区别于传统代码助手的重要特点，是其具备自主执行能力，包括：修改代码、调用 API、执行 Shell 命令、操作 Git 仓库、访问云资源、自动部署应用。但这种能力提升研发效率的同时也扩大了攻击影响范围。

典型风险包括：权限过大、工具调用滥用、CI/CD 自动化攻击。

（五）治理攻击面：AI 代码缺少可见性和责任追踪

技术攻击往往通过治理缺陷实现长期潜伏和扩大影响。安全治理攻击面关注企业在 AI 开发过程中建立的安全管理、权限控制、审计监控和合规机制是否有效。

典型风险包括：AI 代码不可追踪、缺少 AIBOM、权限管理不足、审计困难。

2.2 AI 开发环境引入的新风险影响

AI 开发环境的出现，不仅改变了代码生成方式，也重新塑造了软件供应链结构。传统的软件包、依赖库、构建流水线等“软件组件”，逐渐扩展到模型、插件、知识库、Agent 和 MCP Server 等“AI 能力要素”组成的新型智能开发体系。

在这一体系下，攻击者的目标已不再局限于传统软件漏洞，而是开始针对模型输入、上下文管理、知识来源及智能体执行链路发起新型攻击。任意环节受到攻击，均可能通过 AI 生成能力向下游传播安全风险，形成跨组件、跨项目、跨组织的供应链攻击。

AI开发组件引入的风险影响

AI开发过程中引入的各类组件与资源，可能在模型、工具链、数据和权限等层面带来多种安全风险，对企业应用的安全性、合规性和业务连续性产生深远影响。



图 6AI 开发组件的四大风险影响

2.2.1 基础大模型自身的脆弱性与已知后门

作为 AI 开发环境的核心能力，大语言模型已经成为新的基础设施，其安全性直接决定生成代码和开发建议的可信程度。然而，目前大量开源模型和部分商业模型仍存在模型安全验证不足、训练来源不透明、安全审计缺失等问题。

首先，基础模型可能存在训练阶段遗留的安全风险。例如，攻击者可以通过数据投毒 (Data Poisoning) 在预训练或微调阶段植入特定触发条件，使模型在特定 Prompt 下输出恶意代码、错误配置或包含后门逻辑的程序片段。这类攻击通常具有较强隐蔽性，难以通过传统漏洞扫描发现。

其次，部分公开研究表明，模型可能受到模型后门攻击影响。当输入满足特定触发条件时，模型将偏离正常行为，输出攻击者预设结果。例如自动生成危险系统调用、绕过身份认证逻辑或推荐存在漏洞的代码实现。

此外，模型本身还可能受到 Prompt 注入、上下文污染、模型偏见等攻击影响。使攻击者能够操控模型推理过程、窃取模型能力或污染模型上下文，从而间接影响软件开发质量。

因此，基础模型已成为 AI 时代供应链的重要组成部分，企业不仅需要关注模型能力，更应建立模型来源验证、安全评测、版本管理及持续风险监测机制。

2.2.2 工具链、开源 Agent 框架的漏洞及供应链风险

AI 开发过程中，大量 IDE 插件、Agent 框架及开发增强工具承担着模型调用、代码执行、文件访问及自动提交等重要功能。这些工具虽然显著提升开发效率，但也成为新的供应链攻击入口。

目前主流 AI IDE 插件通常拥有较高系统权限，包括读取源码、访问终端、调用 Git、执行 Shell 命令以及连接企业知识库等能力。一旦插件自身存在漏洞，或更新渠道遭受供应链攻击，攻击者即可利用插件获得开发环境访问权限，实现代码窃取、恶意代码注入或敏感信息泄露。

与此同时，开源 Agent 框架大量采用插件化设计，通常依赖数十甚至数百个第三方组件共同运行。这些组件更新频繁、维护质量参差不齐，一旦其中某个依赖包被污染，将通过依赖链快速影响整个 Agent 生态。

此外，部分 Agent 支持自主调用浏览器、终端、GitHub、云平台 API 及办公系统，其权限范围远高于传统开发工具。如果缺乏最小权限控制及运行隔离，攻击者可利用 Agent 执行横向移动、数据窃取甚至自动化攻击。

2.2.3 数据污染对企业定制化微调模型的影响

随着越来越多企业采用私有大模型、垂域微调、持续训练以及 RAG 知识增强等技术构建专属 AI 开发平台，模型训练数据逐渐成为新的攻击目标。

在 AI 开发环境中，数据污染主要体现在以下几个方面：

(1) 污染企业代码仓库。企业微调模型通常会学习 Git 仓库中的历史代码。如果攻击者向代码仓库提交存在安全漏洞或恶意逻辑的代码，并通过代码评审进入训练数据，模型可能将其学习为推荐模板，在后续开发过程中持续生成类似实现，形成漏洞传播。

(2) 污染知识库内容。企业越来越多地采用 RAG 技术连接内部开发规范、设计文档、接口文档及最佳实践。当攻击者能够修改知识库内容或植入伪造文档时，模型检索到的信息将成为其回答依据。

(3) 污染模型微调数据。企业私有模型通常会基于内部代码进行增量微调。如果训练样本中混入大量不安全编码模式、已知漏洞代码、弱加密算法、错误权限模型.....，模型将在参数层面固化这些错误模式，使后续生成结果持续受到影响。相比 RAG 污染，这类风险更难修复，因为一旦模型参数被污染，通常需要重新训练或重新微调才能恢复。

此外，训练数据还涉及版权、隐私保护及数据合规问题。未经授权的数据集可能违反开源协议、商业许可或数据保护法规，导致企业在模型商业化过程中面临法律风险。

2.2.4 Prompt 及 Agent 权限导致敏感数据泄露

AI 开发工具为了提升代码生成质量，通常会向模型提供大量上下文信息，包括当前文件、关联代码、项目结构、配置文件、历史提交记录、知识库文档及用户历史对话等。这种“上下文工程”虽然显著提升了模型能力，但也扩大了敏感数据暴露面。

当前，敏感数据泄露风险主要体现在代码上下文反向推导和知识库越权访问两个方面。

首先，上下文代码泄露。AI IDE 插件通常会自动上传一定范围内的代码上下文，帮助模型理解项目结构。攻击者可通过连续设计具有诱导性的 Prompt，逐步引导模型输出原本不应暴露的代码内容，例如：项目目录结构、接口定义、配置参数、数据库模型、内部算法实现、私有函数逻辑。即使模型没有直接返回完整源码，也可能通过多轮对话逐步泄露关键业务逻辑，最终实现对核心代码的逆向重构。特别是，对于采用长上下文窗口的大模型，这类风险将进一步放大，因为模型能够访问更多历史上下文信息。

其次，知识库越权访问。企业 AI 平台通常基于 RAG 构建统一知识中心，将研发文档、设计资料、代码规范及内部制度统一纳入检索范围。如果知识库缺乏细粒度权限控制，模型可能在一次检索过程中同时访问多个权限域的数据。攻击者通过精心构造 Prompt，可能诱导模型输出超出当前用户权限范围的信息，形成“模型级越权访问”。此外，随着 MCP 协议和 Agent 工具生态的发展，大模型能够直接调用浏览器、数据库、Git、CI/CD 平台及各类 SaaS 系统。当 Agent 拥有过大的工具权限时，攻击者可能利用 Prompt Injection 或工具调用链缺陷，引导 Agent 读取本地文件、访问内部数据库或上传敏感数据至外部服务，实现自动化数据窃取。

2.3 隐性依赖与组件来源不透明风险

AI 辅助开发显著提升了开发效率，但也带来了更加复杂的依赖管理问题。特别是“AI 自动引入依赖”模式使软件供应链透明度进一步降低，依赖关系更加复杂，软件物料清单难以及时反映真实依赖情况。同时，AI 幻觉、虚假开源项目及许可证冲突等问题不断出现，使供应链治理面临新的挑战。

2.3.1 AI 推荐第三方组件及开源依赖投毒风险

AI 编码工具在生成代码时频繁推荐第三方库和开源组件，这一功能在提升开发效率的同时，也成为供应链投毒的主要渠道。

AI 编码工具引入的第三方组件具有以下三个显著特点：

- **一是依赖来源更加隐蔽**，开发人员可能无法准确感知 AI 生成代码背后引用了哪些外部组件；
- **二是依赖选择更加自主化**，AI 可能基于训练数据中的历史代码、流行项目或公开仓库推荐组件，而非基于当前企业安全策略进行选择；
- **三是组件可信度更加难以验证**，部分推荐组件可能来自个人仓库、低维护项目甚至恶意仿冒包，从而扩大软件供应链攻击面。

这些特点使 AI 开发环境的供应链攻击具有更强的隐蔽性和扩散能力，也导致未知第三方组件与开源依赖投毒风险进一步扩大。

一方面，AI 自动推荐组件导致依赖透明度下降。AI 代码助手在生成代码时，经常自动引用第三方库、SDK 及工具框架，以快速完成开发任务。这些依赖通常并非开发人员主动选择，而是模型依据训练经验自动推荐。但由于生成过程高度自动化，开发人员容易直接采纳 AI 建议，而忽视组件来源、安全状态及维护情况。部分生成代码甚至会自动添加新的 Package 依赖，使恶意组件能够绕过传统人工审核流程进入项目。在 Agent 自动编程模式下，模型还可能自主执行包安装命令、修改依赖配置或自动更新版本，进一步增加供应链风险。

另一方面，恶意开源包利用 AI 开发生态扩大攻击影响，开源生态投毒事件迅速增加。由于 AI 大模型“幻觉”和自主执行的特性，在生成代码时可能引用并不存在的第三方库、函数或软件包。攻击者利用这一特点，在 npm、PyPI、RubyGems 等公共开源生态中提前注册与 AI 模型高概率生成结果相匹配的恶意软件包，植入恶意代码。当开发人员按照 AI 建议安装这些软件包时，即可触发供应链攻击。随着 AI 编码工具的大规模普及，这一攻击可能成为未来软件供应链的重要威胁。

2.3.2 开源许可证冲突与知识产权穿透风险

AI Coding 不仅带来了新的安全风险和技术适配不足等问题，也对传统软件知识产权管理体系带来了新的挑战。

首先，大模型训练过程中学习了大量开源代码，其生成结果可能与已有开源项目存在较高相似性，甚至直接复现部分受许可证约束的代码片段。如果开发人员未经审核直接采用相关代码，可能导致企业违反 GPL、AGPL 等强 Copyleft 许可证要求，引发开源合规风险。

其次，AI 自动引入第三方组件时，也可能混合使用不同许可证的软件包，形成复杂的许可证继承关系。例如 MIT、Apache-2.0、GPL 及商业许可组件同时存在于同一项目中，可能导致授权义务冲突，影响软件商业发布。

此外，随着 AI 生成代码比例不断提升，代码原创性、版权归属及责任认定问题也逐渐受到关注。企业需要建立 AI 生成代码的知识产权审查机制，对生成代码来源、相似性、许可证兼容性及商业授权风险进行系统评估。

2.4 2026 年典型安全事件复盘与启示

为了更直观地理解这些新型威胁的破坏力，报告对 2026 年发生的三起地标性安全事件进行了深度复盘。

2.4.1 Cline CLI 供应链投毒事件

事件背景：2026 年 2 月，Cline CLI 的 npm 发布链发生未经授权版本发布事件。Cline 是一款开源的

AI 编码助手，它还有一个发布在 npm 上的 CLI 版本。在 VS Code 和 JetBrains 上拥有超过 500 万用户，是人们日常用于 AI 辅助开发的工具之一。2 月 17 日，攻击者利用被盗令牌向 npm 发布了 cline@2.3.0，并在 package.json 中加入 postinstall 逻辑，所有安装或升级到 cline@2.3.0 的用户，在不知情的情况下都会额外安装 OpenClaw。在该软件包被撤下之前的 8 小时内，大约有 4,000 次下载。

事后分析表明，该版本 CLI 二进制文件与前一版本字节完全一致，唯一变化为 package.json 文件，未发现公开证据表明存在恶意代码或后门植入。



图 7 Cline CLI 供应链投毒

事件影响：该事件首次系统性展示了 AI Agent 作为 CI/CD 自动化执行主体时可能成为软件供应链攻击的新入口。智能体因其高执行权限和非确定性的提示词接收，已沦为新型供应链攻击的绝佳入口，传统的代码签名和静态 CI/CD 隔离策略对此完全失效。

2.4.2 Claude Code 凭据泄露漏洞

事件背景：2026 年 1 月，Anthropic 修复并披露了 Claude Code 项目配置加载漏洞 CVE-2026-21852，受影响版本为 2.0.65 及之前版本。

NVD 及厂商安全公告表明，Claude Code 项目加载流程中的一个漏洞允许恶意仓库在用户确认信任前窃取包括 Anthropic API 密钥在内的数据。如果用户在攻击者控制器仓库中启动 Claude Code，且仓库包含一个设置文件将 ANTHROPIC_BASE_URL 设置为攻击者控制的端点，Claude Code 就会在显示信任提示前发出 API 请求，并可能泄露 API Key 等敏感信息。

4 月至 6 月，多名安全研究人员进一步披露 Claude Code Action 存在 Prompt Injection 攻击风险。当 CI/CD 工作流处理来自外部 Pull Request、Issue 或 Comment 等不受信任内容时，攻击者可通过恶

意提示词诱导 Claude Code 调用本地工具访问运行环境中的敏感文件（如/proc/self/environ），读取 GitHub Token、API Key 等环境变量，造成 CI/CD 凭据泄露。Anthropic 随后在 Claude Code 2.1.128 版本中完成修复，并进一步完善了 Claude Code Action 的安全防护机制和官方安全指南。



图 8 Claude Code 凭据泄露事件

事件影响：Claude Code 漏洞说明 AI 开发工具的信任边界已经被重新定义，同时也暴露出 AI Agent 在工具调用授权、运行环境隔离及上下文可信校验等方面仍存在安全缺陷。这些漏洞与传统代码漏洞不同，攻击者无需利用系统漏洞即可借助 AI Agent 自身的工具调用能力访问运行环境中的敏感数据，进一步扩大攻击范围。该事件推动 GitHub、Anthropic 及开源社区重新评估 AI Agent 在自动化研发流程中的权限模型，并促使行业开始关注 Agent Tool Use、Context Isolation 及 Runtime Sandbox 等关键安全能力。

2.4.3 Claude Code 后门风波

事件背景：2026 年 7 月，工业和信息化部网络安全威胁和漏洞信息共享平台（NVDB）发布安全通报，指出 AI 编程工具 Claude Code 部分版本（2.1.91—2.1.196）存在恶意代码植入风险，可能导致开发终端敏感信息外传及供应链安全隐患。平台建议相关单位立即排查受影响版本，升级至官方修复版本，并加强开发终端外联访问控制、开发工具完整性校验及异常网络流量监测，以降低开发环境供应链攻击风险。



图 9 Claude Code 后门通知

事件影响：该事件反映出 AI 开发工具自身已成为软件供应链的重要组成部分，一旦开发工具受到恶意代码污染，其影响范围将覆盖开发终端、代码仓库、CI/CD 流水线以及企业内部研发网络。由于 AI 编程工具通常具备代码读取、文件访问、终端执行及网络通信等高权限能力，若工具完整性遭到破坏，可能导致源代码、身份凭据及研发资产持续泄露，并进一步影响软件交付安全。该事件也促使国内企业开始加强 AI 开发工具版本管理、软件来源可信验证及开发终端安全管控。

2.4.4 启示与总结

上述事件表明，AI 开发环境正在沦为软件供应链安全的新攻击目标，且安全问题已经从单一工具漏洞演变为影响软件供应链全生命周期的系统性风险。随着 AI 编程工具、代码智能体逐渐从辅助编码工具演变为能够访问代码仓库、调用系统工具、执行自动化任务的智能开发主体，传统以代码安全、组件安全和 CI/CD 隔离为核心的软件供应链防护体系已难以覆盖新的风险边界。

第三章 AI 开发与供应链安全技术

传统 SAST、DAST、SCA、SBOM 等技术仍然发挥基础作用，但面对 AI 生成代码、Agent 自主执行、Prompt 动态调用等新风险特征暴露出诸多不足。企业必须构建覆盖 AI 开发全生命周期的安全能力，确保 AI 开发工具可信、AI Agent 可控、研发流程可审计。

AI 开发与软件供应链安全技术，是面向 AI 原生软件开发模式建立的新型安全体系。核心能力包括 AI 开发安全、AI 代码可信共识验证、软件供应链安全三部分。

其中：

- AI 开发安全解决“AI 如何安全参与开发”的问题，核心技术包括：AI 代码安全、AI Agent 安全、AI 开发治理。
- AI 代码可信共识验证解决“AI 生成代码是否可信”的问题，核心技术包括：多模型代码验证、多元化代码安全检测、代码可信评分。
- 软件供应链安全解决“AI 软件如何全过程可信交付”的问题，核心技术包括：AIBOM 管理、可信模型管理与 AI 围栏技术、依赖组件安全管理、流水线可信门禁。

三者共同构成覆盖 AI 代码生成、验证、构建、交付全过程的 AI 原生软件安全保障体系，是传统 DevSecOps 向 AI DevSecOps 演进的重要技术基础。



图 10 关键技术框架

3.1 AI 开发过程的关键安全技术

AI 开发安全是保障 AI 参与软件开发全过程安全可信的基础能力。与传统软件开发安全不同，AI 开发安全不仅关注“开发人员写出的代码是否安全”，更关注“AI 生成、修改和维护的软件是否可信”。重点是围绕 AI 代码生成、AI Agent 开发行为以及 AI 开发流程治理展开，在 AI 开发环境中建立覆盖代码、Agent、模型、数据和流程的安全防护体系。

3.1.1 AI 代码安全检测和审计

AI 代码安全检测与审计，是面向 LLM/Coding Agent 生成代码、AI 开发全链路风险构建的新一代代码安全审计范式。它以大模型语义理解、多智能体协同为核心底座，从“检测代码”演进为“检测整个 AI 编码过程”，覆盖提示词输入、AI 生成代码、代码入库、Agent 执行及最终交付全过程。

核心能力包括：

(1) AI 代码漏洞检测：利用大模型安全分析能力结合传统代码扫描技术，识别 SQL 注入、命令执行漏洞、权限控制缺陷、敏感信息泄露、不安全加密实现、输入验证缺失等代码缺陷问题，实现从传统规则匹配向语义理解检测演进。

(2) AI 安全编码规范检查：针对 AI 生成代码容易出现的问题，如使用过期 API、使用弱安全算法、缺少异常处理、默认配置不安全，通过 AI 代码审查能力，实现安全编码规范自动检查。

(3) AI 代码生成安全：AI 代码生成安全是 AI 开发安全中的新增方向，重点解决

- 代码幻觉风险：例如，生成不存在的函数、调用错误 API、使用错误技术方案。
- 不安全代码生成：例如，生成硬编码密码、生成弱认证机制、引入危险函数。
- 恶意上下文影响：例如，Prompt Injection 诱导 AI 生成恶意代码、上下文污染导致安全策略失效。

技术路线方面，目前产业主要形成两类演进方向：一类是传统代码安全厂商主导的“AI 增强”代码检测和审计路线；另一类是 AI 模型厂商主导的“AI 原生”代码检测和审计路线。



图 11 两种技术路线

- AI 增强代码检测和审计，是在传统 SAST、SCA 等程序分析技术基础上，引入了大模型语义理解、智能推理和自动化验证修复能力，实现代码安全检测范式升级。是传统代码安全能力的智能化升级。
- AI 原生代码检测和审计，突破了传统以 AST 规模匹配为核心的检测模式，以大模型语义理解、多智能体协同为核心引擎，通过融合程序分析与大模型推理能力，实现静态检测智能化、动态验证自动化和代码审计智能化。作为新一代安全范式，是 Agentic SDLC 的长期发展方向。理论上，两者都采用了大语言模型的语义理解能力，能够发现传统 SAST 难以检测的复杂漏洞，特别是那些需要深度上下文理解才能识别的业务逻辑漏洞和权限控制缺陷。

但 AI 增强架构则更多依赖传统应用程序分析技术，检测准确性和稳定性更成熟，更易融入现有开发体系。而 AI 原生驱动对代码语义理解能力更强，适合复杂业务逻辑分析，但对大模型和 Agent 框架的依赖性较强，目前的准确性和稳定性仍有不足。

代码安全检测和审计智能化升级，并非是在传统审计架构中简单增加 LLM 大模型能力，更重要的是

面向 AI 编码环境和 Agent 化开发流程，对传统代码检测、漏洞分析和安全审计能力进行体系化重构。具体包括：

- 增强静态代码安全分析能力，提升复杂漏洞、业务逻辑漏洞和 AI 生成代码风险识别能力；
- 利用大模型驱动智能化模糊测试，实现测试用例自动生成、攻击路径探索和漏洞验证自动化；
- 通过 AI 代码审计能力模拟安全专家进行代码 Review，实现漏洞解释、风险分析和修复建议生成。

3.1.2 AI Agent 安全

在 AI 驱动的软件开发中，具备自主决策能力的 AI Agent（如自动 Coding Agent、AI Dev-Ops Agent）逐渐接管代码修改、环境部署及自动化测试等核心工作。其安全性直接关系到软件工程底座的稳固。AI Agent 安全需要遵循多层防护原则，重点措施包括：

- **最小权限与执行隔离环境：** 为每一个 AI Agent 分配细粒度的角色访问控制（RBAC）与属性访问控制（ABAC），限制其对敏感代码库、生产配置文件及密钥管理的访问。Agent 的自动化代码编译与测试过程必须完全运行在沙箱或轻量级容器隔离环境中。
- **提示词注入与越狱防护：** 针对针对性极强的 Prompt 注入攻击（试图改变 Agent 的执果或使其越权操作），在 Agent 输入端部署高灵敏度的输入过滤层，并采用系统提示词强约束（System Prompt Hardening）防护技术。
- **行为监测与异常打断机制：** 建立 Agent 行为基线，对高风险操作（如提交 PR、修改 CI/CD 流水线配置文件、删除生产数据等）引入强行的人类确认门禁，并具备一键安全打断能力。

3.1.3 AI 开发流程治理

传统的开发流程治理主要是做好职责分工与流程管理，但在 AI 开发环境下，还需要确保 AI 辅助工具在合规、可控前提下发挥作用，覆盖数据/隐私、工具与模型准入、知识产权、身份与权限、生成内容复核、开发过程可追溯和退出机制。核心包括：

- **AI 数据安全与隐私保护：** 建立敏感数据脱敏与机密计算机制，防止企业核心源码、商业机密或客户敏感信息（PII）在作为 Prompt 提交给外部公有云模型时发生泄漏；确保企业使用的 AI 模型符合版权与开源协议合规性（如 Avoid Copyleft Infringement）。
- **工具与模型准入管理：** 建立 AI 资产清单，明确各业务部门与团队 AI 代码生成工具的使用边界，自动标记 AI 生成代码的比率与位置，为后续的代码版权保护和责任追溯提供依据。高风险工具明确退出、替代和应急禁用路径。
- **生成代码的审计和追溯：** 对 AI 参与的代码变更，优先记录模型/Agent 标识、版本、任务、相关 Prompt/Context 摘要、工具调用、提交/PR、复核人和构建 Provenance。AI 代码来源管理依靠工程元数据和证据链，而不是事后使用低可靠性的“AI 代码检测器”猜测作者。

3.2 AI 代码可信共识验证机制

传统 DevSecOps 更多是“发现代码漏洞”，而 AI 编程时代还要验证“代码是否可信、依赖是否可信、行为是否符合预期、运行是否安全”，安全目标需要从“发现代码漏洞”延展为“验证 AI 生成代码的安全性、可信性和行为一致性。”

AI 代码可信共识验证是 AI DevSecOps 体系中的关键安全能力，通过融合多 AI 模型、多元化安全检测工具、自动化测试、供应链分析，对 AI 生成代码进行多维度验证，并形成可信判断。其核心思想是从“单一工具发现漏洞”转向“多主体、多证据形成可信共识”。

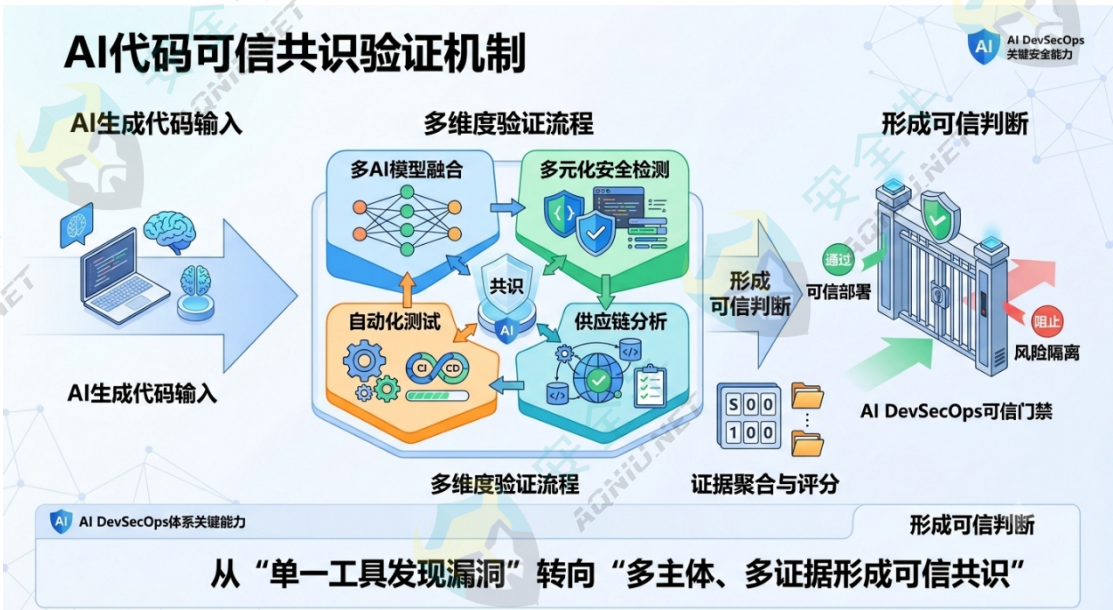


图 12AI 代码可信共识验证机制

共识验证机制通常包括：多模型代码验证、多元化代码安全检测、代码可信评分。

3.2.1 多模型代码验证

鉴于单一 AI 模型均存在固有的“幻觉”边界和知识盲区，采用多模型共识机制可大幅降低误报率与漏报率，提高判断准确性。多模型交叉审查可作为语义证据之一，适合发现需要跨文件、跨函数和业务语义理解的问题。

- **交叉验证与异构模型对比：**采用架构不同、训练数据来源不同的多元大模型（如开源安全强化模型与商业顶尖模型），对同一段 AI 生成的业务代码或重构逻辑进行平行安全审计与意图交叉校验。
- **多数表决与共识裁决算法：**基于语义相似度计算与漏洞特征匹配，自动整合多模型给出的审查意见。当多个独立模型对某段代码的安全性和功能正确性达成一致意见时，方可赋予更高的信任度。

3.2.2 多元化代码安全检测

在多模型验证的基础上，结合传统的硬核安全工具链，建立多层次的深度检测机制。常用的检测方法：AST 与 LLM 深度融合的语义分析、自动化安全测试。

- **AST 与 LLM 深度融合的语义分析：**将基于规则的静态分析、动态分析、交互式分析（IAST）与 LLM 的深度上下文理解能力相结合，大幅提升对复杂业务逻辑漏洞（如越权、逻辑异常、并发冲突）的识别能力。
- **自动化生成安全单元测试，在代码生成后即时验证。**利用专注于测试用例生成的 AI 工具，针对生成的代码自动构造边缘条件与攻击载荷，在隔离环境中自动运行测试，验证代码的鲁棒性与边界安全性。发现严重风险禁止继续执行。

3.2.3 代码可信评分

代码可信评分用于排序和管理残余风险，是量化评价 AI 生成代码安全状态的决策依据。通过综合分析漏洞严重度/可利用性、代码质量、依赖健康度与许可证，形成 AI 代码可信分值，以判断代码是否满足发布要求。

- **多维加权评分模型：**构建涵盖代码安全性（漏洞密度、敏感函数使用）、鲁棒性（单测覆盖率、异常处理逻辑）、合规性（开源协议冲突、代码相似度）、来源可信度（使用的模型版本、Prompt 安全评级）的综合评分模型。
- **动态阈值与安全基线：**根据业务系统的安全级别（如核心金融结算系统 vs. 边缘演示项目）设定差异化的可信评分阈值。低于安全门槛的代码将自动被标记并拦截，阻止进入下一阶段。

3.2.4 可信网关/门禁

AI Code Trust Gate 与 CI/CD 可信门禁是将可信评分与共识验证结果深度无缝集成至企业的 DevSecOps 流水线中，实现全自动化的卡点控制。其目标是在代码进入构建、测试、发布之前进行可信判断。

- **Trust Gate（可信网关）：**是在开发者提交代码（Git Push / Pull Request）的入口处设置智能化网关，面向 AI 生成、修改和执行代码建立的前置可信验证机制，通过对代码来源、内容、依赖、行为及生成意图进行综合评估，在代码进入代码仓库、构建流水线或执行环境之前实施风险识别与安全准入控制。
- **CI/CD 可信门禁：**是面向软件持续集成、持续交付全过程建立的安全控制机制，通过对代码、依赖、测试、构建/制品及发布环境多维度数据进行综合安全验证，确保只有满足安全、质量、供应链和合规策略的软件制品才能进入下一交付阶段，并对 Hard Fail 和可例外风险采用不同处置路径。

可简单理解为，AI Code Trust Gate 是“AI 原生开发安全”的前置门禁，CI/CD 可信门禁是“软件供应链安全”的交付门禁。二者共同构成从 AI 代码生成到软件发布的纵深可信控制体系。



图 13 安全控制流程图

3.3 AI 时代的软件供应链安全技术

AI 开发最大的风险之一是企业越来越难以掌握实际使用了哪些 AI 工具、模型、插件和 Agent。AI 开发供应链的扩展不仅导致大量未经审批的 AI 能力进入研发体系，滋生影子 AI 风险，还会造成 AI 资产底数不清、依赖关系无法溯源，数据泄露、提示注入、模型越权调用等各类安全隐患持续隐匿。

本章节将从 AI 开发环境资产管理、可信模型管理、依赖组件管理 3 个方面对 AI 时代软件供应链安全技术进行阐述。



图 14 软件供应链安全的核心技术

3.3.1 面向 AI 环境的新一代 SBOM 管理

SBOM 是软件可信、安全和合规的重要基础能力。但在 AI 开发环境下，软件供应链的构成、依赖关系和风险边界发生了根本变化。传统 SBOM 已经无法完整描述 AI 软件供应链，需要进一步演进，实现 AI 软件供应链全生命周期管理。

面向 AI 环境的下一代 SBOM 管理需要解决传统 SBOM 在 AI 软件开发场景中的多个局限性，包括：AI 模型的表示和追踪、AI 生成代码的标识、动态依赖的管理以及跨域供应链的关联等问题。



图 15AIBOM 管理的关键技术

(1) AIBOM 是 AI 时代软件供应链透明化管理的重要基础

AIBOM (AI Bill of Materials, 人工智能物料清单) 是指用于描述、记录和追踪人工智能系统中所有组成要素、依赖关系、来源信息和生命周期状态的结构化清单。其核心目标是：实现 AI 组件全链路透明可视，支撑 AI 供应链风险评估与可信管理。

AIBOM 不仅是单一的文件，而是对 AI 应用程序所涉及的全部广义资产进行统一声明的技术，核心由以下四个深度维度组成：

- **模型元数据**：声明模型的原生提供商、具体版本哈希、底层架构类型、微调权重适配器特征、模型评估指标等底层参数。
- **数据集资产**：披露模型微调和训练数据集的血缘 (Lineage)、版权归属分类、数据清洗与脱敏方案、匿名化技术以及在 tot-link 中的完整哈希凭证。
- **Prompt 上下文**：定义静态 System Prompt 模板版本、安全分类防护阈值以及关联的 API 调用策

略。

- **Agent 基础设施**: 梳理 Agent 绑定的 MCP 连接服务器、本地与外部工具链（命令行、网页抓取）、容器沙箱和网络出网权限基线。

AIBOM 分析可以帮助企业进一步做好可信模型管理

目前, CycloneDX 与 SPDX 等国际 SBOM 标准正在加速向 AI 供应链场景扩展:

- **CycloneDX 1.6**: OWASP CycloneDX 最早在 1.5 版本 (2023 年 6 月) 就开启了对 ML-BOM (机器学习物料清单) 的初步框架支持, 是较早将 AI 资产纳入 SBOM 体系的标准之一。CycloneDX 1.6 进一步增加了对机器学习模型特征、超参数、数据集、推理配置元数据的针对性 Schema 定义, 便于安全工具执行快速漏洞关联, AI SBOM 提供了标准化的表示格式。
- **SPDX 3.0**: Linux 基金会推出了 SPDX 3.0, 扩展了 AI 与 Dataset 相关的规范配置文件 (AI Profile & Dataset Profile), 通过标准化语义模型描述 AI 模型、训练数据以及相关依赖关系, 为模型来源追踪、数据治理和 AI 供应链安全分析提供支持。

(2) AI 生成代码的溯源和标识是下一代 SBOM 的另一个关键能力

当代码库中同时包含人类编写代码和 AI 生成代码时, 需要能够区分两者的来源, 以便在出现安全问题时进行准确归因。GitHub 和 Anthropic 等厂商已经开始在 AI 生成的代码中嵌入元数据标识, 例如特定的提交消息签名或代码注释标记。Georgia Tech 的 Vibe Security Radar 项目正是利用这些标识来追踪 AI 生成代码相关的 CVE。

未来, AI 生成代码的 SBOM 可能需要包含生成模型信息、生成时间、原始提示或需求描述以及人类审查者信息等元数据。

(3) 动态依赖管理是 AI 开发环境对 SBOM 提出的新挑战

传统 SBOM 主要追踪静态依赖关系 (如 package.json 或 requirements.txt 中声明的依赖), 但 AI 开发环境中的许多依赖是动态确定的。例如, AI 编码工具可能在运行时根据代码上下文动态推荐和安装依赖包, RAG 系统可能在查询时动态检索和加载文档片段, MCP 服务器可能在执行时动态发现和调用工具。

这些动态依赖关系难以通过静态分析完全捕获, 企业需要构建“动态 AIBOM 生成与实时追溯体系”, 在运行时监控动态更新 SBOM。

理想状态下, AIBOM 应该实现自动化、实时更新: 即每次构建或部署时由系统自动生成, 而非依赖人为填写。企业通过在 CI/CD 中嵌入轻量级审计探针, 可以实时捕捉新增或更新的模型和依赖, 并将其与最新漏洞库对比, 自动标记受影响项。这种动态 SBOM 的方案使得即使在 AI 应用持续集成/部署的快节奏下, 也能保持对整个供应链安全态势的可视化和可追踪。一旦发现上游组件出现漏洞, 可迅速定位受影响的模型或服务, 并触发补丁或降级等措施, 实现快速响应。

(4) 跨域供应链关联是下一代 SBOM 管理的高级能力

AI 软件的供应链横跨多个领域：AI 模型供应链（训练数据、模型权重、推理服务）、软件供应链（开源组件、商业库、内部代码）、基础设施供应链（云服务、容器镜像、Kubernetes）。下一代 SBOM 管理需要能够在这三个域之间建立关联，实现端到端的供应链可视性。例如，当某个训练数据集被发现包含隐私敏感信息时，需要能够通过跨域 SBOM 追踪所有基于该数据集训练的模型，以及这些模型被哪些应用所使用。这种跨域关联能力对于全面理解和管理 AI 供应链风险至关重要。

3.3.2 可信模型管理与 AI 围栏技术

相比 AIBOM，可信模型管理关注的是：**一个模型进入企业环境后，是否持续保持安全、可靠和可控。**

AI 安全围栏，也被称为大模型安全网关、AI 防火墙、AI 安全管理平台，安全可覆盖模型引入、使用、运行、更新、退役的全生命周期，是当前企业 AI 大模型应用安全管理的重要安全基础设施，也是企业防御“影子 AI”模型供应链投毒与后门、内容生成等 AI 风险最核心的技术抓手之一。

其本质是部署在大模型 API 与用户、数据、工具、业务系统之间的一层智能安全控制层，目标是对输入/输出双向流量进行合规检测与安全阻断。

- **入向安全过滤**：检测研发向大模型发送的 Prompt 中是否包含敏感业务源码、硬编码 Key，并利用启发式规则阻断外部返回的针对 Agent 的提示词注入 Payload。一旦发现几何距离低于安全红线，直接判定为恶意注入攻击并强制阻断。
- **出向敏感拦截**：对大模型返回的代码进行实时敏感词、后门特征指纹过滤与重写阻断。一旦检出，可利用安全占位符替换（如统一替换为“*”），拦截凭证的外泄路径。

3.3.3 依赖组件安全管理技术

AI 原生应用高度依赖复杂的第三方框架与插件生态（如 LangChain、LlamaIndex、向量数据库等），依赖关系的延伸带来了更为庞大的攻击面。典型的是近年来，针对 AI 开发场景的开源供应链攻击不断增加，例如恶意 npm/PyPI 包通过名称混淆、依赖混淆、投毒 README 或 Prompt 等方式诱导 AI 工具推荐或自动安装。

- **智能化依赖项安全分析 (SCA)**：升级传统 SCA 工具，深入识别针对 AI 框架及其拓展插件的专有漏洞、恶意组件及后门。
- **投毒防御与依赖锁定**：严防“依赖包抢注/typo-squatting”攻击，推行严格的私有镜像源隔离机制与依赖项哈希强锁定，阻断不安全依赖包进入开发流水线。

第四章 产业格局及国内外代表性厂商

本章节将围绕 AI 开发与供应链安全的产业发展现状，结合国内外厂商调研，对代表性厂商进行梳理，分析不同类型厂商的技术路线、产品布局和竞争优势，并进一步研判未来市场演进方向。

4.1 市场规模与产业格局变化

随着生成式 AI、AI 编码助手和智能 Agent 逐步融入软件研发流程，企业对 AI 生成代码可信性、AI 开发过程安全管控以及软件供应链全生命周期管理的需求快速增长，推动 AI 开发安全与软件供应链安全逐渐成为网络安全产业新的增长方向。

从市场规模来看，AI 开发安全与软件供应链安全正处于快速发展阶段，并呈现多元融合增长特征。

一方面，随着企业加速采用 AI 编码工具，AI 驱动的软件开发市场规模持续扩大。际商业研究公司 The Business Research Company (TBRC) 预测，全球 AI 代码工具市场规模预计从 2025 年的约 78.8 亿美元增长至 2034 年的 705.5 亿美元，年复合增长率超过 27%，AI 代码生成、智能补全、自动测试和代码优化等能力正在成为软件研发基础设施的重要组成部分。

另一方面，随着软件供应链攻击频率增加以及监管要求提升，软件供应链安全市场正在从传统开源组件管理向覆盖代码仓库、CI/CD 流水线、第三方依赖、软件制品和 AI 生成代码的综合治理体系演进，逐渐成为企业软件安全建设的重要投入方向。

4.1.1 三大阵营竞争格局

从产业发展来看，产业竞争格局发生了明显变化。传统代码安全厂商 Checkmarx、Snyk、Veracode、Synopsys 等开始全面引入 AI 能力，向 AI 代码安全、AIBOM 管理、AI 组件治理以及 AI 开发流程安全扩展；同时，大模型安全、AI 治理和开发工具生态厂商 GitHub、Anthropic、OpenAI、Google、腾讯、阿里、字节等厂商也纷纷推出 AI Coding 产品，向代码安全、Agent 安全和 AI 应用运行时防护领域延伸。整体来看，形成传统安全能力与 AI 原生安全能力融合发展的竞争格局。

从产业供给角度，AI 开发与供应链安全市场仍处于能力融合和生态形成阶段，尚未形成类似传统网络安全领域成熟稳定的生态体系。当前市场主要由三类能力厂商共同驱动：

AI开发与供应链安全市场供给格局（产业视角）



图 16 产业格局

(一) 传统 AST/SCA 厂商积极推进代码安全能力 AI 升级

以悬镜安全、孝道科技、海云安、思客云为典型代表的专业 AST/SCA 厂商正在积极推进软件供应链安全产业 AI 化升级。该类厂商长期深耕开发/供应链安全领域，拥有成熟的 SAST、SCA、IAST 和 DevSecOps 能力，随着 AI 生成代码规模增长，开始利用大模型增强漏洞检测、风险分析和自动修复能力，向 AI 生成代码检测、AIBOM 管理、AI 开发流程安全和软件可信发布方向演进。

其中，具备平台化产品体系和较强研发投入的头部企业依托长期积累的漏洞知识库、DevSecOps 平台及企业客户基础，能够较快完成 AI 能力融合。而中小型及检测能力相对单一的厂商，由于研发资源有限，加上模型及头部平台化厂商的市场挤压，面临的转型和生存挑战相对较大，AI 能力建设和产品升级进展相对较慢。

(二) 主张 AI 治理的综合网络安全厂商开始横向拓展

以绿盟科技、安恒信息、赛讯科技为代表的综合安全厂商秉承“AI 对抗 AI”的核心理念，依托安全大模型的训练实力及模型安全评估、提示词防护、内容安全、AI 应用防护等安全能力，向 AI 治理方向拓展，并进一步延伸到 Agent 安全、AI 运行时防护以及 AI 开发治理领域。

与传统 AST 厂商相比，这类厂商重点围绕 AI 全生命周期重新构建安全产品体系，更加关注模型及 Agent 所带来的新型风险，强调模型安全、AI 运行时安全、AI 供应链安全、Agent 安全，如 Prompt Injection、上下文污染、模型越权调用、Tool Abuse、Memory 污染、MCP 安全、AI 运行时监测及模型供应链风险等问题，并逐步形成覆盖模型开发、Agent 运行及 AI 应用部署全过程的安全能力。

(三) 大模型及运营商依托模型/Agent 优势开始生态布局

随着 AI Coding 成为大模型的重要应用场景，以腾讯、阿里、火山引擎为代表的大模型厂商和云服务提供商也在快速成为代码安全赛道的重要参与者。相比传统安全厂商，其最大的优势在于大模型训练和微调能力与 Agent 平台化能力，可将代码安全审计能力内嵌到大模型，并融入 AI Coding 和 IDE 工具调用过程中，实现“生成即安全”的研发模式，从源头降低漏洞生成概率。

从各厂商模型新版本发布参数来看，大模型在不断增强代码生成、代码审计、Agent 权限控制、敏感数据保护等代码安全能力。与此同时，云平台服务提供商也在积极推动 AI 基础设施、模型服务、开发平台、安全能力和治理能力，构建面向企业级 AI 开发的一体化安全体系，使代码安全能力成为云平台的基础服务之一。

总体来看，AI 开发与供应链安全市场正在经历从“传统安全能力适配 AI 场景”向“AI 原生安全能力构建”的转变。

未来市场竞争焦点将不再局限于单一漏洞检测或组件治理，而是围绕 AI 开发全生命周期形成覆盖代码生成、代码审计、依赖治理、Agent 行为控制、身份权限管理、AIBOM 管理以及持续风险治理的一体化安全能力。领先厂商需要同时具备软件供应链安全基础能力和 AI 原生安全创新能力，才能满足企业在智能化开发环境下对于软件可信、安全和可控的综合需求。

4.1.2 四大核心产品赛道

从产品布局来看，全球 AI 开发和供应链安全厂商正在重点围绕 AI 辅助编码、AI 代码审计、AI 供应链治理、Agent 安全管控四大方向加速布局，形成四条相互融合的产品赛道。



图 17 四大核心产品赛道

(1) AI 编码工具

AI 编码工具主要关注 AI 生成代码过程中的安全控制，包括安全 Prompt、实时漏洞预防、安全编码建议、安全代码生成、安全策略约束等能力，目标是在代码生成阶段减少漏洞产生，实现“Security by Generation”。

这类工具提供商由模型厂商、云厂商、开发工具厂商、开发安全厂商共同推动。

代表性工具，如：海云安 D10、阿里云 Qoder、腾讯云 Code Buddy、智谱-5.2，以及国外的 GitHub Copilot、Claude Code、Cursor、Gemini Code Assist 等。

(2) AI 代码审计产品

AI 代码审计利用 LLM 增强传统 SAST、IAST 及 Code Review 能力，实现漏洞智能分析、自动根因定位、风险解释、漏洞修复建议及自动 Patch 生成，提高代码审计效率和开发者修复效率。

当前国内厂商悬镜安全、安全玻璃盒、安恒等厂商均已发布了 AI 代码审计产品。**代表性产品，如：悬镜安全灵脉 Code AI、孝道科技 AI-SAST、安恒 AI 代码审计、绿盟 AI 代码安全、腾讯云 Code Buddy Security。**

(3) AI 供应链治理产品

围绕 AI 开发过程中涉及的软件包、模型、数据集、插件、MCP 服务、Agent 工具链等对象开展统一治理，包括 SBOM/AIBOM 生成、依赖分析、许可证管理、模型来源验证、签名校验、软件完整性验证以及供应链风险评估等，逐步形成覆盖“代码+模型+数据+工具”的新一代供应链治理体系。

代表性产品，如：悬镜安全问境 AIST、塞讯科技防投毒智能网关。

(4) Agent 应用安全类产品

Agent 作为一种新型应用形式，由于具备自主规划、工具调用、代码执行和外部交互能力，已成为当前 AI 系统最大的风险入口。国家标委在 2026 年 7 月发布的新标准规划中已将《智能体应用安全基本要求》标准计划列为强制标准。

Agent 运行时安全管控是 2026 年 AI 安全领域出现的新方向，指针对能够自主调用工具、执行动作的 AI Agent（智能体）在运行时的安全防护。核心功能包括：Prompt Injection 防护、Tool 权限控制、Agent 身份认证、工具可信调用、Memory 安全、运行时监测以及 Agent 行为审计等。

安全牛最新产业调研显示，Agent 安全赛道的参与者诸多，涵盖软件供应链安全厂商、传统网络安全厂商与云厂商。

代表性产品，如：悬镜安全问境 AIDR、阿里 ASC、腾讯云 AI Agent 安全中心、火山引擎 Jeddak AgentArmor 可信智能体安全框架等。

综上，AI 辅助编码关注开发效率与代码生成可信性，AI 代码审计聚焦生成代码质量与安全缺陷发现，AI 供应链治理扩展至模型、组件、数据和工具链风险管理，Agent 安全则进一步覆盖智能化开发流程中的权限控制与行为治理。但上述四个方向并非相互独立，而是伴随 AI 融入软件开发生命周期逐步融合，

共同推动软件安全体系由传统 DevSecOps 向 AI 驱动的可信开发体系演进。

4.2 国外代表性厂商及产品方案

报告梳理了海外代表性厂商 GitHub、Anthropic、Snyk、Checkmarx、JFrog、GitLab 在 AI 转落地方面的举措与实践路径。其中：

- GitHub、Anthropic 是 Coding Agent 方向的典型代表；
- Snyk、Checkmarx 是 AI 增强应用安全方向的典型代表；
- JFrog、GitLab 是软件供应链安全治理方向的典型代表。

国外技术路线及代表性厂商



图 18 国外技术路线及代表性厂商

4.2.1 GitHub：构建 AI 编码与代码安全融合体系

作为全球最大的代码托管平台，GitHub 依托微软生态，将 AI 编码能力与软件安全能力深度融合，形成以 GitHub Copilot、GitHub Advanced Security、CodeQL 和 Copilot Autofix 为核心的 AI 开发安全体系。

主要能力包括：AI 生成代码安全检测、AI 辅助漏洞修复、开发流程内生安全。

GitHub 代表了“DevSecOps 向 AI DevSecOps 演进”的技术路线，其优势在于开发生态覆盖广、安全工具链完整，但复杂业务逻辑漏洞发现能力仍依赖 AI 模型能力和传统安全分析技术结合。

4.2.2 Anthropic: Claude Code 推动 AI 原生代码安全发展

Anthropic 推出的 Claude Code 及 Claude Code Security 代表了 AI 原生代码安全 Agent 方向。与传统 SAST 工具不同，Claude Code Security 并非依赖规则匹配，而是利用大模型代码理解和推理能力，对代码仓库进行上下文分析，实现漏洞发现、风险解释和修复建议生成。

其主要特点是：大模型驱动的代码理解能力、漏洞分析与修复闭环、融入 AI 开发流程。底层大模型早期依托 Claude Opus。目前，Claude 大模型已经升级到了 Fable 5/Mythos 5。但由于美国管制政策，Mythos 5 受限开放，目前，Claude Code 默认后端使用的是公开商用版 Fable 5，常规开发、普通代码评审足够，但一旦任务偏向深度漏洞挖掘，能力自动回落至 Opus 4.8。

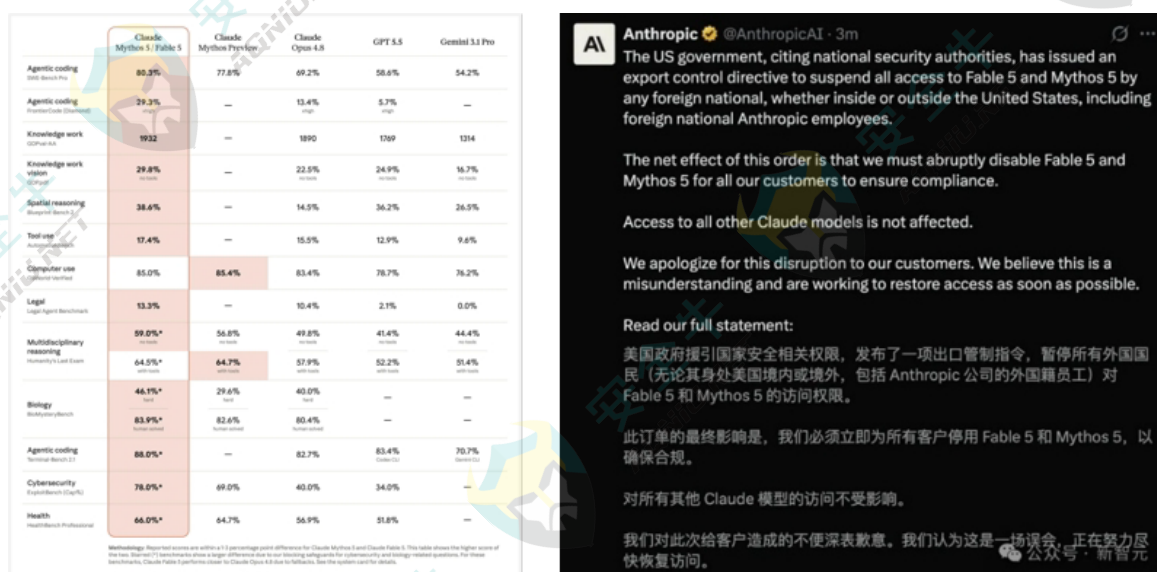


图 19 Claude Code

Claude Code 代表未来 Agentic SDLC 中的安全 Agent 发展方向，其核心价值在于利用 AI 推理能力提升代码安全分析效率，但由于模型主要依赖云端服务，在高度敏感环境下仍面临数据隔离和部署限制。

4.2.3 Snyk、Checkmarx: AI 时代的软件开发安全平台

Snyk、Checkmarx 是国外开发者安全、应用安全领域的重要厂商，其产品体系覆盖 SAST、SCA、容器、AI Security 等能力。

目前，都在围绕 AI 生成代码安全进行能力扩展，包括：

- AI 代码风险检测：支持对 AI Coding 工具生成代码进行安全分析，识别。
- AI 辅助漏洞修复：基于大模型能力生成修复建议，帮助开发人员快速解决漏洞。
- 开发者安全体验优化：通过 IDE 插件、代码仓库集成和 CI/CD 集成，实现安全能力嵌入研发流程。

Snyk 代表了传统 AppSec 厂商向 AI 增强安全平台升级的发展方向。相比 Snyk, Checkmarx 的定位更加偏向大型企业 DevSecOps 安全平台, 重点满足复杂软件研发环境下的统一安全治理需求。

4.2.4 JFrog: 面向软件制品全生命周期安全治理

JFrog 通过 JFrog Xray 构建软件供应链安全体系。主要能力包括: 软件组成分析 (SCA)、SBOM 生成与管理、漏洞关联分析、制品仓库安全治理。

随着 AI 应用发展, JFrog 进一步关注 AI 模型制品安全、AI 组件依赖管理、AI 软件供应链风险治理。推动软件供应链安全从“组件治理”向“AI 软件全生命周期治理”方向发展。

4.2.5 GitLab: 融合 DevSecOps 与软件供应链安全

GitLab 通过 GitLab Duo 和 GitLab Security 平台, 将 AI 辅助开发、安全测试、CI/CD 流水线、软件供应链治理进行融合。其目标是构建覆盖代码开发、测试、发布全过程的 AI 驱动 DevSecOps 平台。

综合来看, 国外厂商正在沿着传统安全工具向 AI 安全 Agent, DevSecOps 向 AI DevSecOps, 软件供应链安全向 AI 供应链安全的方向演进, 市场正在形成“AI Coding Agent+AI 增强 AppSec+软件供应链治理+ AI DevSecOps 平台”的融合发展格局。

4.3 国内代表性厂商及产品方案

国内厂商能力覆盖了代码安全检测/审计、软件供应链安全、Agent 运行时安全多个维度。基于调研, 分别对国内厂商悬镜安全、塞讯科技、奇安信、安恒信息、腾讯云安全的方案进行介绍。



图 20 国内代表性厂商

4.3.1 悬镜安全：灵脉 Code AI 代码安全智能体

悬镜安全起源于北京大学网络安全技术研究团队“XMIRROR”，是数字供应链安全开拓者，国家级专精特新“小巨人”企业。首创基于“AI 原生安全+软件供应链安全+AI 供应链安全情报”的新一代数字供应链安全治理体系，以 AI 治理 AI，从源头治理 AI 数据准备、模型训练调试、智能体开发部署到运行迭代等关键环节面临的 AI 原生安全风险，帮助企业用户构筑一套从“代码与组件”到“数据与智能”的智能体 AI 全生命周期内生安全治理体系，持续守护新一代数字供应链安全。同时主导推出全球首个国产数字供应链 SBOM 标准 DSDX，搭建起涵盖软件供应链、AI 原生安全、供应链情报的全栈技术能力体系。



图 21 悬镜安全新一代数字供应链安全体系

面向 Agentic Coding 全新开发范式，悬镜安全推出新一代 AI 原生代码安全智能体——悬镜灵脉 Code AI。产品坚持“以 AI 治理 AI”，搭建完整闭环代码安全治理体系，破解 AI 生成代码漏洞泛滥、传统安全工具误漏报、AI 智能体供应链风险失控等现实难题，是国内数字供应链安全厂商在 AI 代码安全领域的标杆实践。



图 22 灵脉 Code AI 技术原理

灵脉 Code AI 采用自研 Agent 自主循环工作架构，融合大模型深度推理能力，形成 Plan- Search- Analysis- Verify- Fix 五段式全自动漏洞处置闭环，补齐传统规则型 SAST 短板。依托仓库级跨文件、跨函数的代码语义理解能力，能够同时覆盖 AI 生成代码特有缺陷、常规业务逻辑漏洞以及 AI 供应链链式风险检测，精准还原漏洞数据流与业务逻辑，大幅减少误漏报。

研发赋能方面，与 IDE 深度集成，实现安全左移。深度兼容全栈研发工具链，原生打通 VS Code、JetBrains、Git、Jenkins 与 Cursor、通义灵码、Claude Code 等主流 AI 编码工具，构建安全左移与 AI 代码安全护栏双重能力。在 IDE 实时编码场景下拦截 AI 代码风险，精准识别 AI 投毒、密钥硬编码、权限绕过、Prompt 注入等新型安全问题；借助 Agent 自动化处置闭环，一站式完成漏洞定位、根因研判、补丁生成、修复提交和回归校验，减少超 70%人工安全排查工作，实现研发效率与安全管控双向保障。

生态协同方面，支持全栈软件供应链协同防护。灵脉 Code AI 深度嵌入悬镜完整数字供应链安全体系，可与源鉴 SCA、问境 AIST、灵脉 PTE、云脉 AI 情报平台全线联动，打通软件资产与 AI 智能体全生命周期防护：（一）联动多模态 SCA 能力，覆盖源码、二进制文件、容器镜像、AI 模型组件的全方位安全审计，全面排查各类资产潜藏风险；（二）对接问境 AIST 平台，支持 MCP 工具、智能体 Skills 插件安全检测，精准拦截工具投毒、数据越权窃取等新型安全威胁；（三）联动云脉 AI 情报平台，每 2 小时同步全球 AI 框架、模型、插件的 0day 漏洞及投毒预警，依托 AI-BOM 清单精准定位风险资产、锁定风险影响边界，实现快速溯源止损。

场景适配方面，覆盖软件全生命周期合规管控。灵脉 Code AI 全面覆盖软件研发全生命周期安全管

控场景，贯穿需求编码、构建测试、发布上线、运营监测全流程，形成闭环式安全防护体系。编码阶段可实时防控 AI 编码安全风险，测试阶段联动 AI 渗透能力完成漏洞精准验证，交付阶段输出标准化 SBOM/AI-BOM 物料，支撑合规审计落地，运营阶段持续同步前沿情报预警、迭代防护规则，实现动态安全防护。产品深度适配金融、政企、车联网、工业互联网、头部互联网等多行业业务场景，既能满足等保、信创、开源合规、AI 安全监管等各类硬性合规要求，又可有效解决企业 AI 规模化应用落地后，代码安全管控失控的核心难题。

4.3.2 塞讯科技：CyriGate 防投毒智能网关

塞讯科技（Cyrityx），是一家定位于攻击者情报研究和攻防验证的新锐安全厂商，成立于 2021 年。在 APT 组织情报之外还聚焦软件供应链安全相关情报收集，针对近年来软件供应链投毒、恶意依赖包传播、开源组件风险以及 AI 辅助开发带来的新型供应链威胁，推出软件供应链防投毒智能网关解决方案。

不同于多数 AST/SCA 厂商从代码安全/SCA 分析向威胁情报演进的路径，其创始团队深耕 APT 组织、勒索团伙、黑产组织的攻击技术分析，擅长站在攻击者视角还原完整入侵链条。攻击者视角威胁情报库是其在软件供应链安全领域的核心竞争资产。

- 2024 年，塞讯科技依托原有威胁情报库能力，开始从网络黑产向开源软件供应链扩展，切入开源供应链投毒研究。包括：常态化监控全球主流包仓库（NPM、PyPI、Maven、Go、NuGet）和 AI Skills 市场，重点识别恶意开源包、依赖混淆、名称劫持、账号劫持篡改恶意版本、AI SKILLS 等风险。供应链情报能力覆盖了传统包、恶意 AI Skills、AI 插件投毒、恶意开源组件。
- 2026 年，公司正式推出 CyriGate 防投毒智能网关，聚焦制品下载链路，专门防御开源组件、AI-Skill、AI 编程工具的供应链投毒攻击。该产品的推出，标志着公司实现了从情报服务商向软件供应链安全的战略升级。



图 23 塞讯科技软件供应链防投毒智能网关

相比传统基于 SDLC 治理的软件供应链安全体系，塞讯软件供应链防投毒智能网关更偏供应链入口防投毒，可部署在企业开发环境入口，作为私有制品仓库代理，也可与 AI 代码安全、AI Agent 安全、SCA 等软件供应链安全能力组合共同构成完整防护体系。

4.3.3 奇安信：Qcode Agents 推动代码卫士全面升级

奇安信是作为一家综合网络安全厂商，也是国内传统代码安全向 AI 增强 DevSecOps 和软件供应链安全平台化方向演进的典型代表厂商

奇安信代码安全智能体（Qcode Agents 简称 QCAs）依托奇安信在 AST、SCA 领域多年技术积累，以及代码卫士、开源卫士在 2000+大型企业 与机构的规模化落地经验打造的全场景代码安全解决方案。



图 24 奇安信 Qcode Agents

产品以多智能体协同为核心，底层搭载全维度代码安全知识库，融合传统程序分析技术与前沿大模型推理能力，通过代码安全智能体集群在代码提交、分支合并、CI/CD、安全测试等开发、集成、测试环节自动执行安全检查，实现从代码变更审核到业务逻辑漏洞挖掘的全场景、专家级安全管控，支持研发无感接入，为企业代码开发提供全方位安全保障。

4.3.4 安恒信息：恒脑 AI 代码审计智能体

安恒信息是传统安全厂商向 AI 增强型应用安全平台方向发展的典型代表厂商。

2026 年 4 月，安恒信息在数字中国建设峰会上，正式推出了企业级 AI 原生代码安全审计产品——恒

脑 AI 代码审计智能体 2.0。



图 25 恒脑 AI 代码审计智能体 2.0

其功能覆盖需求安全建模→编码检测→漏洞研判→沙箱验证→智能修复全研发安全链路, 支持 Java、Python、Go、C/C++、PHP 等主流编程语言, 支持私有化部署、公有云服务、IDE 插件、DevOps 流水线集成多种交付形态。

产品核心依托安恒自研的恒脑安全垂域大模型（深度融合了安恒在安全领域十余年的攻防经验与数据积累研发出的自进化漏洞挖掘框架），采用全局调度、追踪、漏洞挖掘、漏洞验证等多智能体协同框架，实现了从代码理解、漏洞定位到利用验证的全流程自动化。

凭借这一技术底座，恒脑 AI 代码审计智能体在全维度代码覆盖率与超大规模代码库审计能力两大核心维度上取得重大突破，能够高效处理海量代码的深度检测，精准识别业务逻辑漏洞及多漏洞组合利用链，为企业构建主动防御体系提供了领先的智能化解决方案。

4.3.5 腾讯云：CodeBuddy IDE 与安全平台

腾讯云是国内 AI 研发平台与智能编码安全领域的重要布局厂商，围绕 AI 辅助编程、智能研发和代码安全审计持续构建产品体系，先后推出 CodeBuddy IDE 平台和 CodeBuddy Security AI 代码安全审计产品。

腾讯云 CodeBuddy 定位于企业级 AI 研发助手，提供智能编码、代码解释、自动测试、代码审查和研发知识库能力。可以嵌入主流的 IDE（如 VS Code、JetBrains）工具，底层依托“混元+DeepSeek 双模型”驱动，覆盖需求→设计→编码→测试→部署，支持 200+编程语言，提供独立 IDE、第三方插件、CLI

命令行三种交付形态。

为应对 AI 代码漏洞激增、传统代码审计遭遇天花板的痛点，2026 年 6 月，腾讯云正式发布商业独立的企业级代码漏洞审计产品 CodeBuddy Security。

CodeBuddy Security 是以 CodeBuddy 为执行底座构建的安全专用的漏洞挖掘与验证工作流，结合自研 AI 深度审计引擎与静态分析工具 Xcheck，形成“AI 深度审计+规则扫描”双引擎协同的代码安全方案。通过情报驱动的聚焦扫描、独立二次校验与沙箱自动化验证，实现从漏洞发现到 PoC 确证的闭环交付，为企业代码提供深层安全保障。



图 26 腾讯云 CodeBuddy Security

第五章 企业 AI 驱动 SDLC 安全建设指南

企业 AI 代码安全建设的核心目标，不仅是发现 AI 生成代码中的安全缺陷，更是建立可信软件保障体系，实现 AI 驱动软件开发过程的**可控、可视、可审计和可信**的过程。

以下将从建设原则、评估模型、AI 开发平台建设、AI DevSecOps 建设方面展开分析，提出面向企业实际落地的 AI 代码安全建设思路。

5.1 AI SDLC 安全建设的六项基本原则

面对 AI 编码工具、大模型代码生成以及智能开发 Agent 带来的软件研发模式变化，企业 AI 代码安全建设不能简单沿用传统软件安全治理模式，而需要结合 AI 参与软件生命周期后的新风险、新场景和新技术特点，构建覆盖开发全过程、融合 AI 能力的新型安全体系。

AI 驱动 SDLC 安全建设应遵循“**AI 原生适配、开发周期全覆盖、安全左移与自动化、代码可识别可追溯、依赖与供应链安全、治理与合规**”六项核心原则，推动安全能力从传统的代码检测向覆盖 AI 研发全过程的智能化、自动化和持续化安全治理演进。



图 27 AI SDLC 安全建设六项核心原则

5.1.1AI 原生适配原则

AI 参与软件开发后，传统安全工具面临检测对象变化、攻击面扩展和风险类型变化等挑战。企业 AI 代码安全建设需要面向 AI 生成代码、开发 Agent 以及大模型交互过程进行能力升级，而不是简单叠加大模型能力。

具体包括：适配 AI 生成代码特点、支持自然语言驱动开发场景、支持 Agent 开发模式、增强上下文理解能力。

企业需要建设面向 AI 时代的软件安全能力，实现从“代码安全检测”向“AI 赋能开发安全治理”转变。

5.1.2 开发周期全覆盖原则

AI 时代的软件开发流程中，安全风险不再局限于传统代码编写和上线环节，而是逐步前移并贯穿需

求分析、提示词交互、AI 代码生成、模型调用、依赖引入、代码构建以及 Agent 自动执行等多个环节。

因此，企业 AI 代码安全建设需要实现从传统的“上线前代码检测”向“AI 代码全生命周期安全治理”转变。围绕“需求设计—AI 编码—代码提交—持续集成—构建发布—运行维护”完整软件生命周期，将安全能力嵌入研发流程，实现开发阶段安全防护、提交阶段安全验证、构建发布阶段安全管控、运行阶段持续监测。

通过全过程治理，实现安全能力与研发流程的深度融合，将安全由传统的“上线前检查”转变为贯穿软件生命周期的“内生安全能力”，降低安全问题在开发后期集中暴露和返工修复的成本。

5.1.3 安全左移与自动化原则

AI 代码生成显著提升了软件开发效率，代码生成、修改和迭代速度远高于传统人工开发模式。如果仍采用“开发完成后统一检测”的传统安全模式，容易出现安全检测滞后、问题修复成本高以及安全能力与研发流程脱节等问题。

因此，企业应推动安全能力进一步左移，将安全控制前置至 AI 编码和代码生成阶段。将安全检测能力嵌入 IDE、AI Coding 工具、代码仓库和 CI/CD 流水线，实现安全检查自动触发、风险自动识别和高风险代码自动阻断。

核心理念是：代码生成速度越快，安全验证越应该自动化。

5.1.4 AI 生成代码可识别、可追溯原则

AI 生成代码具有生成主体非人工化、生成过程动态化、代码来源复杂化和变更频繁化等特点，传统软件开发模式下以开发人员、代码仓库和版本记录为核心的追踪机制，企业无法知道哪些代码是 AI 生成的、由什么模型生成、经过哪些修改以及最终进入了哪个软件版本。当 AI 生成代码需要功能维护，出现漏洞、安全事件或供应链风险时，无法快速定位受影响代码及其生成来源。

因此，企业需要建立 AI 生成代码可识别机制，AI 生成代码全链路追溯机制，同时强化审计与责任追踪能力。形成建立“代码身份+生成来源+变更过程+安全检测+发布流向”的完整证据链。明确开发人员、AI 编程工具、Agent 及自动化流水线的责任边界，对 AI 生成代码进行身份标识、来源记录、人工审核和责任追踪，确保“代码可追溯、行为可审计、责任可认定”。

5.1.5 依赖与供应链安全原则

AI Coding 显著提高了代码生成和第三方依赖使用的速度，扩大了软件供应链范围。代码来源已经不仅包括开发人员和开源社区，还包括 AI 模型生成代码、第三方模型服务、Agent 插件、工具链组件等，导致供应链风险日益严峻。

关注点包括：AI 幻觉导致的不存在/错误依赖、恶意或被投毒的开源组件、Typosquatting、依赖版本风险、Transitive Dependencies、包仓库供应链攻击、构建环境投毒。这也是传统 SBOM 向 AIBOM

演进的重要基础。

因此，企业需要从传统软件供应链安全扩展至面向 AI 开发环境的**可信 AI 软件供应链治理**。通过供应链透明化和可追溯管理，降低代码投毒、恶意依赖和隐藏风险传播可能。

5.1.6 治理与合规原则

AI 开发作为 AI 应用服务的重要组成部分，同时具有软件开发、人工智能应用和软件供应链等多重属性，其安全治理不能仅依赖传统代码安全措施，而应建立覆盖网络安全、人工智能安全、软件安全和供应链安全的综合治理与合规体系。

首先，企业 AI 开发活动应满足国家网络安全和数据安全相关法律法规要求，并落实关键信息基础设施保护、网络安全等级保护等基础安全要求。

其次，对于涉及生成式人工智能、AI Agent 等应用场景的，还应进一步遵循人工智能专项法律法规、政策、管理办法及安全标准要求，确保 AI 开发活动在主体责任、数据安全、内容安全、模型应用和服务安全等方面满足监管要求。**尤其是强制标准要求 GB 45438 及正在起草的《智能体应用安全基本要求》。**

再次，2026 年以来，我国在加速完善人工智能、智能体、软件安全及软件供应链相关标准规范，AI 开发安全治理开始由原则性要求逐步向**可执行、可评估、可验证、可审计**的标准体系演进。新标准包括 GB/T 47020-2026《网络安全技术 软件物料清单数据格式》（2026-08-01 实施），GB/T 47470-2026《网络安全技术 软件安全开发能力评估准则》（2026-11-1 实施）以及是 5 月立项的《人工智能代码生成服务安全要求》。

可见，未来企业软件开发与供应链治理将会以法规标准为基础约束，建立覆盖 AI 开发全生命周期和软件供应链全链条的安全治理体系，推动 AI 开发安全从技术防护向体系化、规范化、持续化治理演进。未能满足相关法规标准要求的企业，将可能面临合规风险、安全事件责任以及监管处罚风险。

5.2 企业 AI 开发安全成熟度评估模型（AI-DSMM）

为帮助企业客观评估其在 AI 开发和供应链安全领域的防护现状，报告参考 CSA 的 AI 安全成熟度模型 (AISMM) 与 SANS 的 AI 安全成熟度框架，设计了“企业 AI 开发安全成熟度评估模型 (AI Development Security Maturity Assessment Model, AI-DSMM)”。旨在为企业提供一个系统性的框架，用于评估和改进其 AI 编码安全管理的成熟度水平。

该模型从五个等级、五个维度对企业 AI 开发安全管理能力进行评估。

5.2.1 AI-DSMM 模型定义

AI-DSMM 模型包含五个顺序等级，各等级具体定义如下：

AI-DSMM 模型：AI编码安全成熟度五级模型

从不可见到可进化，构建企业AI编码安全的阶梯式能力体系



Level1:初始级——未部署任何 AI 专用的检测与管理手段，企业对研发人员使用 AI 的行为完全缺乏感知，处于严重的“影子 AI”暴露状态。

Level2:可观测级——部署了基础的安全工具，引入静态代码扫描作为初步兜底，建立基本的 AI 编码安全策略和流程，能够识别主要 AI 工具/调用，感知研发团队正在使用哪些第三方 AI 开发工具。但覆盖范围和执行深度有限。

Level3:受控合规级——形成标准化的 AI 编码安全管理体系，安全工具链完整且集成，流程覆盖 AI 编码全生命周期。如，上线了统一准入、NHI/权限、基础 Trust Gate、人工复核和整改闭环，核心系统开发禁止使用非合规工具，初步建立人工 Code Review 对 AI 代码的复核制度。

Level4:动态可信级——安全实践的效果通过数据进行度量和持续改进机制有效运行，安全能力与 AI 编码速度相匹配。如，落地了 AIBOM、Provenance、Coding Agent 行为监控、运行时证据形成动态度量，能够防御典型的间接提示词注入攻击。

Level5:自适应优化级——AI 编码安全实践达到行业领先水平，AI 安全智能体深度参与安全治理，安全成为竞争优势。如，实现全链路“AI Native DevSecOps (AIOps-Sec)”，通过高可信的 AI 审计智能体矩阵实现自主的安全编码决策、漏洞一键智能化自愈，安全策略随模型演进实现全面自适应。

5.2.2 AI-DSMM 评估维度

企业 AI 开发安全成熟度评估模型应从**组织管理、技术能力、供应链治理、人员能力和监督改进**五个维度全面衡量企业 AI 开发安全体系的成熟程度。

AI-DSMM五个评估维度



图 29 AI-DSMM 五个评估维度

(1) 安全治理与流程管理

安全治理与流程管理是 AI-DSMM 的基础能力领域，主要评估企业是否建立了覆盖 AI 研发全生命周期的安全管理体系，确保 AI 开发活动有制度可依、有流程可循、有责任可落实。

重点关注的事项包括：

- 是否建立 AI 开发安全治理组织，是否制定 AI 开发规范、Prompt 管理规范、Agent 使用规范、模型接入规范以及 AI 工具准入制度；
- 是否将 AI 安全要求纳入需求分析、设计开发、测试验证、上线发布及运维管理全过程；
- 是否建立 AI 资产管理、风险评估、变更管理及事件响应机制，实现 AI 开发过程规范化、制度化管理。

随着 AI Agent 逐渐具备自主调用工具、自主生成代码、自主执行任务等能力，传统 DevSecOps 流程已无法完全覆盖 AI 研发场景，因此企业需要构建 AI 原生 DevSecOps (AI DevSecOps) 流程，实现安全治理向 AI 研发流程全面延伸。

(2) 技术与工程控制能力

传统代码安全工具主要关注源码漏洞检测，而 AI 开发环境新增了 Prompt、模型、Agent、MCP 工具、知识库、AI 插件等大量新型攻击对象，因此需要构建覆盖 AI 研发全流程的安全工具链。

技术与工程控制能力主要评估企业是否具备支撑 AI 安全开发的技术能力，以及是否通过自动化技术实现 AI 风险持续检测与防护。

主要重点关注的事项包括：

- 企业是否部署 AI 代码审计工具、AI 生成代码检测能力、AI Prompt 安全检测、模型安全扫描、Agent 行为监控、AI 插件检测、AIBOM 管理、Secrets 检测、AI 供应链风险检测等能力；
- 是否具备静态分析、动态分析、软件成分分析、模型成分分析、AI 安全策略控制等技术能力；
- 是否能够与 CI/CD、IDE、代码仓库及 AI 开发平台进行自动化集成，实现安全左移和持续检测。

成熟企业通常能够通过自动化安全平台实现 AI 开发全过程实时监测，并基于风险策略自动阻断高风险 AI 行为。

(3) 供应链可信管理

AI 时代的软件供应链已由传统的软件组件扩展至基础大模型、AI 代码助手、Agent 框架、Prompt 模板、训练数据集、MCP 工具、插件市场、知识库、Embedding 模型等多类资产，其来源复杂、更新频繁、可信性差异较大，导致供应链攻击面显著扩大。

供应链可信管理主要评估企业是否能够识别、管理和持续监控 AI 开发环境中的供应链风险，建立可信 AI 供应链管理体系。

重点关注事项包括：

- 企业是否建立 AI 资产清单，是否对第三方模型、AI 工具、Agent 平台及开源组件实施准入管理；
- 是否建立 SBOM/AIBOM 管理机制，实现模型、组件、Prompt 及 Agent 依赖关系可追溯；
- 是否开展供应商安全评估、许可证合规管理、模型来源验证、数字签名验证及完整性校验；
- 是否能够持续识别投毒模型、恶意插件、依赖污染及供应链漏洞风险。

AI 开发环境下，供应链治理不仅关注软件，还应覆盖模型、数据、Prompt 及 Agent 全链路，实现 AI 供应链可信管理。

(4) 人员意识

AI 开发环境中，大量安全事件并非源于技术漏洞，而是由于研发人员过度信任 AI 生成结果、Prompt

泄露敏感信息、违规上传企业代码、错误引用 AI 生成代码或未经审核直接上线所导致。因此，提升人员安全意识已成为 AI 开发治理的重要组成部分。

人员意识主要评估研发人员、AI 使用人员及安全管理人员是否具备 AI 安全意识、安全技能及规范使用 AI 工具的能力。

重点关注：

- 是否建立 AI 安全培训体系，是否开展 AI 开发规范培训、Prompt 安全培训、数据保护培训、AI 供应链风险培训及安全编码培训；
- 是否建立研发人员 AI 工具使用规范、敏感数据输入规范及 AI 输出审核机制；
- 是否定期组织安全演练、安全宣传及能力认证，不断提升研发团队识别 AI 风险和正确使用 AI 工具的能力。

建议企业形成“制度+培训+考核+文化”相结合的 AI 安全意识建设体系，使 AI 安全成为研发团队的日常行为规范。

(5) 合规审计

随着国内外 AI 治理框架不断完善，企业不仅需要做好安全建设，还需要证明自身符合相关法规及行业要求。

合规审计主要评估企业是否建立 AI 开发安全的监督检查、持续评估和合规验证机制，确保各项安全制度能够真正落地执行，并持续优化。该维度企业应重点关注：

- 是否建立 AI 开发安全审计制度，是否定期开展成熟度评估、安全检查及风险评估；
- 是否建立 AI 开发行为日志留存、模型调用留痕、Prompt 审计、Agent 操作审计及 AI 资产审计机制；
- 是否建立安全指标（KPI/KRI）、整改闭环及持续改进机制；
- 是否能够满足监管机构及客户关于 AI 安全、数据安全及软件供应链安全的合规要求。

建议企业形成“持续监测—持续审计—持续优化”的闭环管理体系，实现 AI 安全治理能力不断提升。

这五个维度覆盖了“管理—技术—供应链—人员—运营”五大核心能力，共同构成 AI-DSMM 成熟度模型的完整能力框架。实践中，每个指标应设定可量化的评分标准（如自动化漏洞修复率达到多少，模型资产登记率达标等），并结合权重形成整体评分。通过量化评估，企业可以定期评估自身成熟度和改进效果。

5.3 企业自评指南与建设优先级建议

企业开展成熟度评估时，应从自查开始，根据 AI-DSMM 测评各项指标得分，识别薄弱环节。通常建议先从高风险环境切入（如生产环境的模型服务、防泄漏措施），以及人机协同中最易发生偏差的环节（如开发者 IDE 中的提示注入监测）。

安全建设优先级可依据业务敏感度和技术风险决定。比如，金融行业应先关注数据脱敏和审计链路，

互联网企业则着重在线环境的实时监控。通过逐步提升，从低级别自动化到高级别的全生命周期防护，企业可以稳健地提升 AI 安全能力。

建议分短期、中期、长期分层规划，循序渐进地落地企业 AI 开发安全体系建设：

- **短期（P0，1-3 个月）：**全面普查并下线全网研发侧的“影子 AI”，强制收拢 AI 调用端点至企业合规 AI 安全网关；在开发者终端部署针对硬编码密钥的实时检测。
- **中期（P1，3-6 个月）：**推动引入 AIBOM 标准，将其嵌入现有的 CI/CD 流水线；对研发人员开展“对抗自动化偏见”的专项安全心智培训，重塑 PR 人工复核底线。
- **长期（P2，6 个月以上）：**搭建全流程 Agentic SDLC 沙箱防御体系，演进至动态可信及自主优化阶段，全面释放 AI 生产力同时确保底层供应链的绝对安全。

5.4 AI 开发平台安全建设

AI 开发安全平台（AI Development Security Platform）是面向 AI 辅助编码、AI Agent 开发以及智能化软件交付全过程的一体化安全基础设施，通过融合代码安全检测、软件供应链治理、AI 行为管控、安全智能分析和风险运营能力，实现对“人、AI、代码、工具、组件和运行环境”的统一安全治理。

AI 开发安全平台建设主要包括平台架构设计、核心能力模块建设以及分阶段建设路径三个方面。

5.4.1 AI 开发平台安全架构

AI 开发平台应采用分层解耦、能力开放、智能驱动的架构设计，通过数据采集、智能分析、安全服务和生态集成，实现对 AI 开发全过程的安全治理。

整体架构通常包括数据采集层、数据治理层、智能分析层、安全能力服务层和生态集成层五个层次。

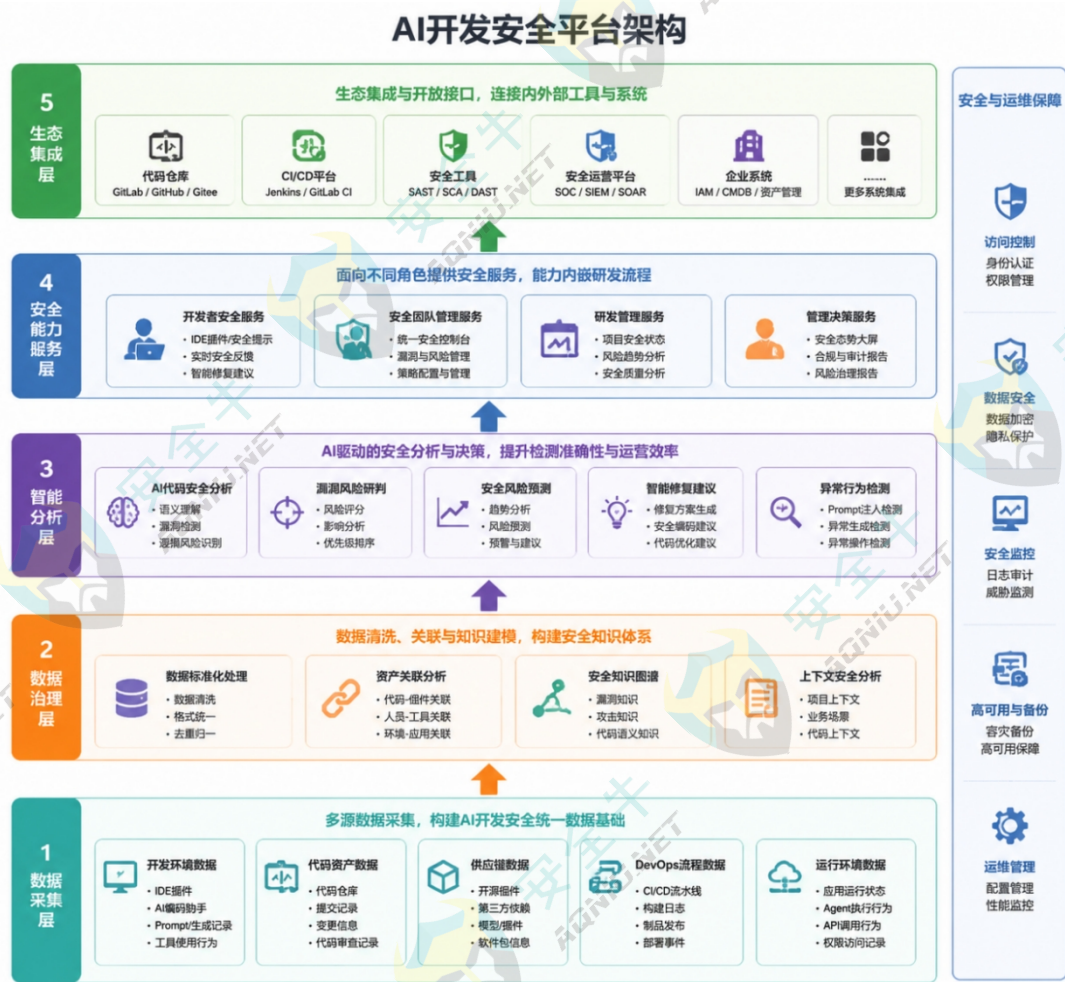


图 30AI 开发安全平台架构

- (1) 数据采集层，是构建 AI 开发安全统一数据基础。负责收集 AI 开发环境如 IDE、代码仓库、CI/CD 流水线、容器运行时等，的多源安全数据，包括代码变更、依赖更新、构建日志、部署事件等，为后续风险分析和安全决策提供基础支撑。
- (2) 数据治理层，是构建 AI 开发安全的知识体系。负责对采集的数据进行清洗、归一化、关联分析和知识建模，形成面向 AI 开发安全场景的数据基础。
- (3) 智能分析层，实现 AI 驱动的安全检测与决策。是 AI 开发安全平台区别于传统 DevSecOps 平台的核心能力，通过融合大语言模型（LLM）、机器学习、安全规则引擎和安全知识库，实现智能化安全分析。
- (4) 安全能力服务层，面向不同角色提供安全服务。负责将平台能力以不同形式提供给研发、安全和管理人员，实现安全能力融入研发流程。
- (5) 生态集成层，实现安全工具和研发体系融合。AI 开发安全平台需要具备开放集成能力，通过标准化接口与企业现有研发和安全体系融合。

5.4.2 核心安全组件

AI 开发安全平台应围绕“代码安全、供应链安全、AI 行为安全、安全智能化、治理运营、合规管理”六大方向建设核心能力。



(1) **AI 代码安全检测模块**：AI 代码安全检测模块是平台的基础能力，主要面向 AI 生成代码、人工编写代码以及 AI 辅助修改代码进行安全检测。该模块应融合传统代码安全分析能力和 AI 增强分析能力，包括：AI 生成代码安全检测、静态应用安全测试增强、软件成分分析能力、敏感信息检测、智能代码审查等。推动代码安全从“提交后扫描”向“编码过程中实时防护”演进。

(2) **AI 软件供应链安全管理模块**：AI 开发模式扩大了软件供应链范围，除传统开源组件外，还引入 AI 模型、Prompt 模板、Agent 插件、工具链和数据集等新的供应链对象。AI 软件供应链安全管理模块主要包括 SBOM 自动生成与管理、AIBOM 管理能力、依赖风险分析、开源组件治理、许可证合规检查等能力。实现从传统软件供应链治理向 AI 时代软件供应链治理扩展。

(3) **AI/Agent 开发行为监控模块**：AI 编码工具和开发 Agent 具有更强的代码生成、修改和执行能力，也带来了新的行为风险。AI 开发行为监控模块主要用于监控 AI 编码工具调用行为、记录 Prompt 输

入和代码生成过程、分析异常代码生成模式、检测 Prompt Injection 攻击、识别 Agent 权限滥用和异常操作、建立开发行为风险画像。通过行为监控，实现对“AI 如何参与研发过程”的安全治理。

(4) 安全智能体模块：安全智能体模块利用大模型 Agent 能力，实现安全任务自动化执行，提高安全运营效率。**主要场景包括：智能代码审查 Agent、漏洞分析 Agent、修复建议 Agent、安全运营 Agent、合规检查 Agent。**安全智能体将成为未来 AI 开发安全平台的重要智能化入口。

(5) 风险态势感知模块：风险态势感知模块负责整合平台各类安全数据，形成企业 AI 开发安全全局视图。主要能力包括：AI 开发资产统一管理、风险集中展示、漏洞趋势分析、攻击路径分析、风险优先级排序、安全运营指标统计。通过风险态势感知，实现企业从单点安全检测向整体风险治理转变。

(6) 合规管理模块：随着 AI 开发安全相关标准体系逐步完善，企业需要建立持续合规能力。支持企业满足 ISO/IEC 27001、ISO/IEC 42001、NIST AI RMF 以及行业监管要求。主要能力包括：AI 开发安全规范检查、软件供应链安全要求检查、开源软件合规审计、AI 模型和 Agent 安全评估、自动化合规报告生成等。

5.4.3 平台建设路径

AI 开发安全平台建设应遵循“先治理基础风险、再完善平台能力、最终实现智能自治”的原则，采用分阶段建设模式。

第一阶段：安全能力接入与风险治理阶段，重点实现 AI 开发安全基础管控。包括：AI 编码工具资产摸底，统一 AI 工具接入管理，建立代码安全检测能力，接入 SAST、SCA、安全扫描工具，建立基础风险管理流程。

第二阶段：平台融合与智能分析阶段，实现从工具管理向平台化治理演进。重点建设：AI 开发安全统一平台、AI 代码安全分析能力、SBOM/AIBOM 管理能力、AI 行为监控能力、安全知识库和智能分析能力。

第三阶段：智能化安全运营阶段，最终形成覆盖“AI 编码—代码提交—构建发布—运行维护”的全生命周期 AI 开发安全体系。重点建设：安全智能体体系、自动漏洞分析和修复、AI 驱动安全决策、自动化风险处置、Agentic DevSecOps 安全体系。

通过建设 AI 开发安全平台，企业能够有效应对 AI 驱动软件研发模式带来的新型安全挑战，实现开发效率提升与安全风险控制之间的平衡，推动软件研发体系由传统 DevSecOps 向 AI 原生安全研发体系演进。

5.5 AI DevSecOps 建设

相比 AI 开发安全平台，AI DevSecOps 是基于 SDLC 开发周期，实现持续开发持续交付的一套“工具+AI+安全+流程”的方法论，更强调安全能力在流程中内嵌。

企业建设 AI DevSecOps 体系，需要围绕研发流程重构、安全能力智能化升级、自动化安全治理和

持续风险运营等方向展开，实现 AI 时代软件研发过程的安全可信。

AI DevSecOps建设



图 31AI DevSecOps 建设

5.5.1 开发工具链与安全控制点的选择

DevOps 工具链覆盖 Plan → Code → Build → Test → Release → Deploy → Operate → Monitor 八大环节，是 AI DevSecOps 建设的基础。按组件集成方式，可将 AI DevSecOps 架构划分成 IDE、代码仓库、CI/CD 流水线、安全平台四层架构。



图 32 AI DevSecOps 的组件结构

工具链建设方面，企业应选择支持 AI 编码安全的工具组合，并实现工具之间的协同工作。如：IDE 层，选择支持安全插件的 IDE（如 VS Code 配合安全插件），配置实时安全检测和 AI 安全提示；代码仓库层，集成安全扫描作为 Pull Request 的自动化检查，配置安全策略阻止高风险代码合并；CI/CD 层，在流水线中嵌入 SAST、SCA、密钥检测和 SBOM 生成，实现安全测试的自动化执行；安全平台层，部署统一的安全态势管理平台，整合各工具的安全发现，提供全局可视性和智能优先级排序。

选择工具时，应特别关注以下 AI 安全能力：

- 对 AI 生成代码的检测准确性；
- 误报率在实际 AI 编码场景下的表现；
- 与主流 AI 编码工具（Copilot、Cursor、Claude Code 等）的集成深度；
- AI 辅助的安全修复能力。

在 AI 参与开发后，软件生产过程发生变化，代码生成、需求理解、测试验证等环节均可能由 AI 辅助甚至自主完成。企业需要将安全控制点覆盖 AI 参与的软件生产全过程。主要包括：

- **需求设计阶段安全治理**：在需求阶段引入 AI 辅助分析能力，对需求描述进行安全分析，包括识别潜在安全需求缺失、分析业务风险和合规要求、自动生成安全设计建议、提前识别隐私和数据保护要求。通过安全前置，降低后续开发阶段风险。
- **AI 编码阶段安全控制**：针对 AI 生成代码场景，需要在编码阶段实现实时安全防护，包括 AI 生成代码安全检测、Prompt 安全分析、安全代码模式识别、开源依赖风险提示、安全编码建议生成。

使开发人员能够在代码产生阶段及时发现风险。

- **测试验证阶段智能化升级：**传统测试主要依靠人工测试和自动化测试工具，AI DevSecOps 需要进一步引入 AI 辅助安全测试、智能漏洞分析、AI 增强 Fuzzing、自动化攻击模拟、智能测试用例生成。提高安全测试覆盖率和效率。
- **部署运行阶段持续安全监控：**在软件上线后，需要持续监控代码变更风险、组件漏洞变化、运行环境异常行为、AI Agent 执行行为、外部依赖风险。实现软件安全风险持续治理。

5.5.2 推动安全工具智能化升级，实现 AI 增强安全检测

传统 DevSecOps 体系通常由 SAST、DAST、SCA、IAST、RASP 等安全工具组成，但这些工具主要依赖规则库、特征匹配和人工分析，在面对 AI 生成代码和复杂业务逻辑时存在一定局限。

AI DevSecOps 需要推动传统安全工具向智能化方向升级。主要包括：

- **SAST 向 AI 代码审计演进：**传统静态分析主要基于规则匹配，而 AI 增强代码审计进一步引入大模型代码理解能力、语义级漏洞分析、上下文关联分析、漏洞影响评估、自动修复建议。实现从“发现代码缺陷”向“理解代码风险”转变。
- **SCA 向智能供应链分析演进：**随着 AI 开发大量依赖开源组件和第三方代码，传统软件成分分析需要进一步扩展 AI 推荐组件安全分析、开源依赖可信评估、恶意包识别、组件风险传播分析、SBOM/AIBOM 关联分析。实现软件供应链风险智能治理。
- **安全测试向 AI 辅助测试演进：**AI 能够提升安全测试自动化水平，包括自动生成测试场景、智能构造攻击输入、自动分析测试结果、自动定位漏洞根因。推动安全测试由人工驱动向智能化驱动发展。

5.5.3 建设安全自动化流水线，实现安全策略内嵌

AI DevSecOps 的核心价值之一，是将安全能力从独立工具转变为研发流水线中的自动化能力，使安全控制成为软件交付流程的一部分。

企业应建设覆盖代码提交、构建、测试、发布全过程的安全自动化流水线。主要包括：

- **代码提交阶段安全检查：**在代码进入仓库前，通过自动化机制检测 AI 生成代码风险、敏感信息泄露、编码规范问题、高危漏洞。减少问题进入后续流程。
- **CI/CD 阶段安全门禁：**在持续集成和持续交付流程中设置安全控制点，包括代码质量门禁、漏洞等级控制、依赖组件风险检查、制品安全验证。根据风险等级决定是否允许继续发布。
- **自动化修复闭环：**结合 AI 能力，实现自动生成修复代码、自动创建修复任务、自动验证修复效果、自动更新安全规则。降低安全整改成本。

5.5.4 建立 AI 开发安全策略管理体系，实现统一治理

由于企业内部可能同时使用多个 AI 编码工具、大模型服务和 Agent 平台，如果缺乏统一安全策略管理，容易导致安全控制不一致。

因此，需要建设统一 AI 开发安全策略体系。主要包括：

- **AI 使用策略管理：**明确哪些 AI 工具允许使用、哪些代码可以提交给外部模型、哪些数据禁止输入 AI 服务、哪些场景必须经过审批。降低违规使用风险。
- **安全规则统一管理：**建立统一安全规则库，包括安全编码规范、漏洞检测规则、Prompt 安全规则、Agent 行为策略、合规要求。并根据威胁变化持续更新。
- **安全策略自动执行：**通过平台化能力，将安全策略自动应用于 IDE、代码仓库、CI/CD 流水线、Agent 运行环境。减少人工干预，提高治理效率。

5.5.5 融合 AI 安全运营能力，实现持续风险发现与响应

AI DevSecOps 不仅关注开发阶段安全，更需要形成持续安全运营能力。

企业应结合安全运营平台，实现 AI 开发环境风险的持续监测。主要包括：

- **AI 开发资产管理：**持续发现 AI 编码工具、模型服务、Agent 实例、开发插件、API 调用关系。形成 AI 研发资产清单。
- **安全风险分析：**结合日志、代码、安全检测结果和威胁情报，分析 AI 生成代码风险、异常模型调用、Agent 异常行为、供应链攻击迹象。提升风险发现能力。
- **安全事件响应：**针对 AI 开发安全事件，实现自动告警、风险阻断、权限调整、代码回滚、安全修复。形成闭环响应机制。

5.5.6 流程建设建议

除了工具链和安全能力，AI DevSecOps 建设还必须做好贯穿软件交付全链路、可自动化执行的持续交付 workflow。流程建设中，建议企业基于自身的开发模式和 AI 采用程度，设计适配的 AI DevSecOps 流程。

- 对于刚开始采用 AI 编码的团队，建议采用“增强审查”模式。所有 AI 生成的代码必须经过人工安全审查，引入的新依赖需要安全团队确认。
- 对于 AI 编码已广泛采用的团队，建议采用“智能闸门”模式。利用 AI 安全工具自动进行安全评估，只有高风险代码才需要人工审查，低风险代码可以自动通过。
- 对于 AI 编码已深度集成的团队，建议采用“持续防护”模式。安全检测完全自动化，AI 安全智能体实时监控和响应安全事件，人工介入仅处理需要复杂判断的例外情况。

第六章 行业实践及厂商案例分享

本章节将围绕金融、互联网、智能制造等典型行业及安全厂商的实践探索展开分析，为企业推进 AI DevSecOps 建设、提升软件供应链安全能力提供实践参考。

6.1 三大典型行业 AI 代码安全分析与实践

本章节内容基于 AQ 研究中心调研整理。

6.1.1 金融：严合规场景 AI 代码审计、溯源与数据防泄露实践

金融行业对安全合规要求极高，AI 开发安全的典型特点，包括：

- 要求全程审计链：每行 AI 生成代码都要留有变更日志，并对 AI 建议的每个依赖进行事前审批。
- 重视数据脱敏和访问控制，只允许 AI 模型使用经过严格脱敏的历史数据。
- 倾向使用私有化部署的 AI 平台，使得模型训练和推理全程在受控环境中进行，以防止模型调用公开 API 时造成数据泄露。

某大型国有商业银行在推行 AI 辅助研发时，实施了严苛的“全面本地化算力隔离+零信任网关”架构。所有的 AI 编程插件只允许连接行内私有化部署的定制模型；研发终端通过虚拟桌面（VDI）彻底隔离，禁止任何形式的外部网络出现。同时，流水线强制启用代码指纹追溯技术，所有 AI 生成的函数段均被打上不可篡改的元数据标签，确保在满足金融监管的非对称审计要求的前提下，实现零敏感数据外流。

6.1.2 互联网：高并发快迭代场景 Coding Agent 提效与安全平衡实践

互联网公司强调开发效率，同时也面对高并发流量下的安全压力。

其普遍做法是在高频场景嵌入“轻量级安全卡点”。例如，在 CI 流程中对 AI 生成的每个微服务代码自动进行 SAST 扫描，并在构建成功后用 AI 模型自动回归测试接口安全性。由于快速迭代，线上漏洞修复时间窗口小，一些公司通过上线“热修复 Agent”来即时拦截危险流量。在开发者文化上则推行“安全即代码”的理念，例如团队代码审查中引入 AI 辅助审查机器人，让全员习惯在 Pull Request 阶段就发现安全问题，从而在保证迭代速度的同时维持基本安全保障。

某头部跨境电商企业在全面引入 Coding Agent 以加速敏捷业务重构时，采用“双轨共识验证”机制——他们放弃了周级别的全量重度扫描，转而将轻量级 SAST 和 AI Auditor 智能体审查矩阵直接嵌入 Git Hook。小幅的代码变更在 5 分钟内通过多 Agent 黑盒测试共识即可自动发布；高危核心模块变更则自动触发运行时沙箱的密集模糊测试（AI Fuzzing），实现了生产效率跃升与全线敏捷防控的弹性平衡。

6.1.3 智能制造：高可靠性场景 AI 原生代码供应链管理

制造业往往涉及嵌入式系统和工业控制软件，对可靠性要求极高。

AI 辅助开发在此领域常用于控制算法、图像识别等模块。其最佳实践包括对 AI 生成代码进行严格的版本锁定和环境隔离。例如，将 AI 工具链全部部署在离线环境或私有容器中，以防引入不确定依赖。供应链上，制造企业会建立详细的 AIBOM，不仅记录软件依赖，还包含硬件固件版本。

针对工业控制场景，企业强调对 AI 模块的可追溯性。每次生成的控制程序都附带生产日期、模型 ID 等元数据，以便在后续出现故障时快速定位问题源头。此类高可靠性管理需与传统工业自动化流程紧密结合，确保 AI 与物理系统的安全协同。

某全球化汽车制造巨头在利用 AI 辅助编写车载工控与车联网（OTA）系统底层 C/C++ 代码时，将安全能力聚焦于防范 AI 原生内存安全漏洞与开源组件穿透。由于工控软件对高可靠性的极端要求，企业在 CI/CD 阶段部署了由 AIBOM 深度关联驱动的严苛软件成分关联分析，并利用自动化硬件在环（HIL）沙箱模拟 AI 原生代码在极限工况下的行为，严防大模型由于幻觉臆造导致物理工控逻辑失效。

6.2 安普诺（悬镜安全）：依托灵脉 Code AI 落地数字供应链 AI 原生代码全生命周期闭环治理实践

案例来源：北京安普诺信息技术有限公司（悬镜安全）

6.2.1 项目背景

数字经济与人工智能深度融合，软件成为企业核心生产要素。研发模式从瀑布、敏捷、DevOps 快速演进至 AI 原生研发，AI Coding、Vibe Coding、AI Agent 等新型开发范式广泛落地。

研发效能大幅提升的背后，安全治理体系建设相对滞后，企业代码及供应链安全遭遇多重挑战，传统防护手段难以适配新场景。

- （一）传统 SAST 检测能力受限，误漏报高发、深层漏洞识别不足；
- （二）AI 生成代码风险泛滥，新型安全漏洞缺乏专项治理手段；
- （三）AI 工具链与供应链链路复杂，隐蔽风险管控失控；
- （四）研安协同机制割裂，人工排查成本高、治理效率低下；
- （五）合规体系适配不足，全生命周期合规治理难以落地。

与此同时，国家与行业针对代码安全、AI 安全、软件供应链安全发布多项监管规范，成为企业 AI 研发安全建设的硬性要求。

在此背景下，悬镜安全基于灵脉 Code AI 代码安全智能体，构建 AI 驱动、全流程闭环、合规适配的 AI 研发代码及供应链安全管理体系。推动安全左移与能力升级，搭建闭环治理能力，夯实供应链安全底座，兼顾合规落地，实现企业降本增效提质。

6.2.2 四层协同防御总体架构

悬镜安全采用“静态分析+AI 智能增强”混合技术架构,以灵脉 Code AI 新一代代码安全智能体为核心,汇聚代码智能检测、AI 安全防护、供应链安全治理、全流程集成管控四大模块,构建面向 AI 原生研发的一体化安全管控平台。



图 33 灵脉 Code AI·AI 代码安全护栏

灵脉 Code AI 搭载全方位核心能力组件,涵盖智能 AI 代码漏洞挖掘、多模智能检测引擎、AI 开发安全助手、AI 代码安全护栏、AI 业务逻辑漏洞检测、MCP 工具安全检测、SCA 供应链融合检测、全维度项目管理、自定义规则策略、全生态集成及多格式报告生成十大核心模块,全面覆盖 AI 原生研发模式下的代码安全与软件供应链安全全场景管控需求。

整体架构分为数据采集体、智能检测层、智能治理层、服务输出层四层体系,层层联动、闭环协同,全面适配 AI 原生研发安全管控场景:

- 数据采集体: 全面对接企业各类研发场景,兼容主流代码仓库、IDE 开发工具、AI 编程工具、CI/CD

流水线及项目管理系统，全域采集人工编写代码、AI 生成代码、第三方开源组件、MCP 工具调用行为、供应链安全情报等多维研发数据，实现研发全场景数据全覆盖、无死角采集。

- **智能检测层：**搭载自研多模智能检测引擎，深度融合词法语法分析、语义分析、数据流分析、传统规则匹配与大模型 AI 增强技术，整合多维检测能力，可高效完成代码缺陷检测、AI 越权风险识别、MCP 工具安全检测、SCA 供应链成分分析、后门程序查杀等核心检测工作，兼顾传统代码安全与 AI 新型安全风险识别。
- **智能治理层：**依托大语言模型与专属代码知识图谱，构建智能化风险治理能力，支持 AI 漏洞自动验证、批量智能安全审计、误报智能标注、AI 代码自动修复、业务逻辑风险识别等核心能力。同时搭配灵活可配的安全策略、自定义规则与全生命周期项目管理能力，实现漏洞风险从发现、研判到整改的全流程智能治理。
- **服务输出层：**支持多格式定制化报告生成、全生态工具链集成、多渠道风险告警推送，精准适配企业合规审计、日常安全管控、风险复盘追溯、流程闭环治理等各类业务场景，实现 AI 代码安全与供应链安全能力轻量化、常态化、体系化落地。

整体架构兼容各类操作系统、CPU 架构与主流研发工具链，无需改造现有研发流程，可柔性嵌入企业 DevSecOps 体系，真正实现安全能力与研发业务的深度融合、高效协同。

6.2.3 八个核心能力构成

灵脉 Code AI 凭借八大核心能力，为企业构建覆盖 AI 研发全链路风险的立体化安全风控体系，推动 AI 安全与研发业务深度融合，做到风险可感知、过程可管控、业务可协同。

（一）智能 AI 代码漏洞挖掘。灵脉 Code AI 以大模型为核心推理引擎、多智能体协同为执行架构，实现源码漏洞自动化深度挖掘，有效弥补传统规则引擎的检测盲区。平台通过项目攻击面精准建模，结合分层解析、知识图谱分片技术突破超大工程与长代码上下文限制，适配大型项目检测场景。依托多 Agent 动态拆分审计任务，专项针对注入、反序列化、权限缺陷等高危风险，完成跨文件、跨层级全链路数据流追踪，精准定位链式漏洞。同时通过攻防双 Agent 对抗验证漏洞真实可达性，依托完整调用链路固化风险证据链，输出分级置信度风险结果与可落地修复方案，全程审计日志可视化、可追溯、可解释。支持公有云、本地私有化大模型灵活部署，适配企业多样化数据安全合规要求，显著提升未知漏洞、复合型漏洞检出能力，大幅降低检测误报率。

（二）AI 业务逻辑漏洞检测能力。依托大模型语义理解、代码解析与多 Agent 协同技术，聚焦传统检测难以覆盖的身份认证、权限控制、业务流程类隐性安全缺陷。平台可自动完成账户体系、支付交易、审批流程等核心业务资产建模，重构跨层级权限调用链路，兼容 Spring Security、JWT、RBAC 等主流鉴权框架。通过智能解析业务状态机，精准识别参数篡改、重复提交、流程跳过、状态异常等业务隐患，全面覆盖订单、资金、资源访问等高风险业务场景。采用攻防 Agent 交叉对抗验证机制，精准甄别水平/垂直越权、认证绕过、权限提升等各类逻辑漏洞，自动溯源风险成因、核验漏洞有效性，剔除无效告警，

全方位筑牢业务系统权限与流程安全防线。

(三) AI 原生研发安全护栏能力。针对 AI Coding、Vibe Coding、AI Agent 等新型智能研发场景，搭建专属 AI 原生代码安全防护体系，填补 AI 生成代码的安全管控空白。平台无缝适配 OpenAI Codex、Claude Code 等主流 AI 编程工具，实现代码生成、接口调用、线上执行全流程闭环安全管控。构建“风险扫描-后门查杀-漏洞审计-智能修复-结果核验”自动化 workflow，可精准拦截 AI 生成代码中的安全缺陷、恶意后门、逻辑漏洞，同步过滤无效告警。从研发源头规范 AI 代码产出，遏制 AI 原生安全漏洞的产生与扩散，实现智能研发场景安全可控、全程可管。

(四) MCP 工具与 AI 链路风险检测能力。基于 MCP 工具及协议落地专属源码安全检测能力，聚焦 AI Agent、MCP Tool 调用全链路风险场景，精准识别各类新型智能工具安全隐患。可专项检测工具影子调用、工具投毒、地毯式恶意变更、间接提示词注入、硬编码密钥、凭证窃取、认证绕过、敏感数据泄露等高风险，全面覆盖 MCP 协议层、工具调用层、智能体执行层的安全短板，全方位补强 AI 工具链与智能交互流程的安全防护能力，保障 AI 工具调用合规、可控、安全。

(五) 智能精准代码检测能力。构建新一代 AI 增强静态代码检测体系，突破传统 SAST 检测精度低、误漏报高、复杂漏洞识别弱的技术瓶颈。采用“静态规则基线+AI 智能增强”双引擎架构，内置 8000+ 专业缺陷检测器，全面覆盖安全漏洞、代码质量、编码规范三大维度问题，支持 40+ 主流编程语言混合扫描。结合抽象语法树解析、跨文件数据流分析、动态污点追踪技术，叠加大模型深度语义理解能力，精准识别传统工具无法检出的复杂逻辑漏洞、链式传导漏洞、隐性权限越权风险。具备高效扫描性能，百万行代码仅需 3-12 分钟，兼顾极致检测精度与研发交付效率，有效降低误报、漏报率。

(六) 高效 AI 开发安全助手。依托大模型与代码安全知识图谱，打造一站式 AI 开发安全助手，全方位赋能研发提效降本。具备五大核心辅助能力：通过代码向量化与 LLM 智能编排，自动生成合规漏洞修复代码，大幅降低大规模代码库整改成本；结合历史缺陷数据与代码调用链路构建知识图谱，输出专家级审计依据，支持批量审计、一键误报标注，大幅减少人工核查工作量。配套 IDE 原生插件，实现编码阶段实时缺陷检测、一键极速修复，压缩研发迭代周期。同时可智能识别代码所属业务场景，匹配差异化审计标准与修复方案，帮助研发人员快速提升编码安全能力与开发效率。

(七) 全维度软件供应链安全治理能力。打造“组件风险检测+供应链情报预警”双维度闭环治理体系，全方位管控软件供应链安全风险。联动双 AI 引擎 SCA 软件成分分析能力，精准检测第三方开源组件、依赖包存在的安全漏洞，结合代码链路完成漏洞可达性分析，科学评估风险影响范围与危害等级。融合云脉供应链安全情报能力，联动企业 SBOM 资产清单，实时监控并主动预警供应链投毒、组件高危漏洞、组件断供、版本废弃等各类风险，实现供应链风险从被动处置向主动发现、提前预判、前置规避转变，构建全流程供应链安全治理机制。

(八) 合规与策略定制体系建设。内置金融、汽车、运营商等多行业专属安全合规模板，全面适配国内外主流安全合规标准，满足各行业常态化合规检查需求。支持用户自定义检测规则、污点跟踪策略、敏感数据识别规则，可通过白名单配置、路径过滤、规则阈值调节精准优化检测结果，适配企业个性化管

控场景。同时支持代码质量门禁、安全评分体系自定义配置，可统一企业代码安全基线与研发质量标准，适配规模化团队、多业务线的差异化安全管控需求，助力企业实现安全合规标准化、常态化、精细化治理。

6.2.4 分阶段落地建设

灵脉 Code AI 可实现全链路研发生态无缝集成，具备高适配、低侵入、易运维的企业级部署能力，全方位适配不同规模企业的业务体量、IT 架构与运维诉求。平台支持单机部署、分布式部署等主流企业级部署模式，兼容 Linux、macOS、Windows 全系列操作系统，适配 AMD64、ARM、ARM64 等各类主流 CPU 架构。同时支持 CLI 命令行快速接入现有研发体系，无需改造企业原有 IT 架构、研发流程与业务系统，部署方式灵活兼容、接入成本极低、后期运维简便，可快速融入企业现有研发生态。

结合企业数字化建设节奏与研发安全治理落地规律，平台整体建设采用循序渐进、迭代落地的实施模式，分为五大阶段稳步落地，实现能力从搭建、试用、优化到全面常态化闭环：

- 第一阶段：部署落地与环境适配。完成灵脉 Code AI 平台整体部署、环境调试与基础参数配置，精准适配企业现有操作系统、研发架构、网络环境及硬件资源。同步完成平台基础规则库、各行业合规模板的初始化配置，搭建适配企业基础安全管控的底层能力底座，保障平台稳定、合规运行。
- 第二阶段：系统集成与能力开通。打通全链路研发生态，完成代码仓库、CI/CD 流水线、IDE 开发工具、AI 编程工具、项目管理系统的无缝对接与集成。全面开通代码漏洞挖掘、AI 业务逻辑检测、AI 原生研发安全护栏、MCP 工具链路风险检测、供应链安全治理、合规检测等全部核心能力，实现安全检测能力全覆盖接入研发流程。
- 第三阶段：策略优化与试点运行。深度结合企业业务场景、行业规范及内部管控要求，自定义优化检测规则、污点跟踪策略、代码质量门禁与系统权限体系。选取核心业务项目开展小范围试点运行，针对性优化检测误报、漏报问题，打磨适配企业自身业务特性与安全标准的专属安全基线，验证平台落地有效性与适配性。
- 第四阶段：全面推广与闭环落地。在企业全业务线、全研发项目推广平台常态化应用，全面落地 AI 智能审计、漏洞智能修复、缺陷闭环管理机制。将安全检测、风险研判、整改修复、合规校验深度嵌入研发全流程，实现 AI 研发全场景、全流程、全维度安全管控常态化、标准化。
- 第五阶段：复盘迭代与持续优化。定期开展安全数据、检测效果、风险态势复盘分析，持续优化平台检测规则、风控策略与 AI 检测模型。紧跟 AI 研发新技术场景、软件供应链新型风险、行业合规新标准，动态迭代平台安全能力，持续提升企业研发安全治理水平，构建长效安全防护体系。

6.2.5 价值分析

灵脉 Code AI 研发安全与软件供应链安全体系全面落地后，可系统性解决企业新型 AI 研发场景下的安全治理痛点，打破传统安全检测“精度低、效率慢、整改难、合规弱、成本高”的瓶颈，实现安全防护、研发效率、合规管控、运维成本四维能力全面升级，核心量化成效如下：

（一）检测能力大幅升级。彻底根治传统 SAST 工具高误报、高漏报的行业痛点，针对复杂业务逻辑漏洞、AI 场景越权漏洞等高危隐性漏洞，检出率提升 80% 以上；百万级代码扫描效率提升 60%，实现检测覆盖无盲区、风险识别更精准，检测精度与综合能力达到行业领先水平。

（二）AI 代码风险源头拦截。依托 AI 原生研发安全护栏，可有效拦截 85% 以上的 AI 生成代码安全漏洞，从研发源头阻断 AI 原生漏洞的产生与扩散，全面提升 AI 编码场景的代码安全性与合规性，杜绝新型智能研发场景的安全风险隐患。

（三）漏洞整改效率显著跃升。通过 AI 智能修复能力，可减少 90% 以上的人工代码修复工作量；结合 AI 批量审计功能，大幅缩减 90% 的人工审计耗时，整体漏洞修复闭环周期缩短 50%，有效打通研发与安全协同壁垒，研安一体化协作效率大幅优化。

（四）供应链风险全域可控。实现开源组件、MCP 工具、AI 智能体调用链路、供应链安全情报的全维度、全流程管控，供应链高危风险预警覆盖率达 100%，可提前预判、有效规避组件漏洞、供应链投毒、组件断供、版本废弃等各类高危安全事件，筑牢软件供应链安全防线。

（五）代码质量持续迭代优化。通过常态化精准检测与智能整改治理，企业整体代码安全缺陷率降低 70%，核心业务系统安全缺陷率降低 60%，从根源减少代码层面安全隐患，彻底杜绝因代码漏洞、供应链风险引发的系统宕机、业务异常、数据泄露等重大安全问题。

（六）研发交付提质增效。将安全检测前置、全流程自动化融入 CI/CD 研发链路，无需人工介入值守，有效解决传统安全测试后置导致的漏洞堆积、项目返工、交付延期等问题，企业整体产品研发交付周期缩短 30%，实现安全与效率双向提升。

（七）合规能力全面达标落地。全面适配金融、汽车、运营商等多行业监管规范与国内外主流合规标准，支持标准化、个性化定制合规报告快速输出，可高效支撑企业各类安全审计、资质核查与监管检查，实现合规治理常态化、标准化、轻量化落地。

6.2.6 安全牛评

悬镜安全依托全国五地研运服务网络与十余年数字供应链安全技术积淀，沉淀出国际领先的数字供应链防护体系。灵脉 Code AI 依托整体防护底座与情报能力，遵循悬镜“智能情报驱动，“以 AI 治理 AI”的安全理念，针对性抵御各类 AI 新型安全威胁。产品可为企业构建覆盖代码生成、风险检测、自动修复、供应链情报预警的一体化内生安全屏障，助力组织在借力 AI 实现研发提效的同时，牢牢守住代码安全与 AI 智能体供应链安全底线。

从技术路线看，该案例体现了“传统程序分析+LLM/Agent 语义能力+软件供应链治理+研发流程集成”的融合方向，适合作为 AI SDLC 安全实践参考。

第七章 趋势预测

随着人工智能技术快速融入软件研发全生命周期，AI 开发模式、软件供应链体系以及安全防护理念正在发生深刻变革。同时，AI 生成代码、AI Agent、模型调用链、数据资产以及第三方 AI 组件带来的新安全风险和治理挑战，将推动代码安全、供应链安全和 AI 安全能力持续融合。

本章将结合 AI 开发模式演进趋势、关键安全技术发展方向以及市场生态变化，对未来 AI 开发与供应链安全领域的发展路径进行分析，为企业开展面向未来的软件研发安全体系建设提供参考。

7.1 监管政策趋势分析

全球范围内，围绕 AI 开发和供应链的安全监管体系正在持续完善。整体趋势表现为从单纯的“已知漏洞风险防御”，演变为“涵盖代码、数据、模型、工具链的全生命周期透明度（AI-SBOM）与合规审查”。



图 34 国内外 AI 监管趋势对比

7.1.1 国际监管持续强化，向全生命周期与供应链透明化演进

首先，软件供应链安全要求透明化，SBOM 成为基础合规要求。美国 NIST 发布并持续完善安全软件开发框架（SSDF）及配套实践 SP800-161，强调软件来源可信、开发过程可追溯以及供应链透明化。美

国政府采购及国防领域也正在加强对 AI 开发平台和大模型供应商的软件供应链安全评估。SBOM 正逐步向覆盖模型、数据集、推理组件和 Agent 依赖关系的 AIBOM 方向扩展，虽然 AIBOM 目前尚未成为强制监管要求，但已成为国际产业界重点推进的最佳实践。

其次，AI 生成代码被纳入开发周期安全管理。随着 GitHub Copilot、Cursor、Claude Code 等 AI 编码工具快速应用，国际标准开始关注 AI 生成代码带来的安全风险。NIST 在 DevSecOps 实践指南中明确指出，AI 可以用于代码生成、安全分析和漏洞修复，但 AI 生成内容必须经过人工验证，需要建立可信的软件开发流程，避免不安全代码直接进入生产环境。

第三，欧盟推动“设计即安全（Secure-by-Design）”和软件生命周期监管。欧盟近年来也在持续强化数字产品安全监管。《网络韧性法案》要求具有数字元素的软件和硬件产品，在设计、开发、维护全过程中满足网络安全要求。同时，欧盟《AI 法案》要求高风险 AI 系统建立完善的风险管理、日志记录、技术文档及可追溯机制，对 AI 开发安全同样适用。

随着高风险 AI 系统相关条款逐步实施，供应链透明化、模型来源说明和开发过程可审计性将成为企业满足合规要求的重要能力，AIBOM 等新型供应链清单也被广泛认为是支撑 AI 系统透明治理的重要技术手段。

7.1.2 国内政策标准持续完善，加速 AI 开发安全治理要求

首先，现有网络安全标准要求关键行业企业落地 SBOM 全生命周期管控。《关键信息基础设施安全保护条例》、《网络安全审查办法》、等保 2.0、SBOM 国标/行标（GB/T 47020、SJ/T 12097）、工信部漏洞管理规定，均要求关键行业企业落地 SBOM 全生命周期管控。

其次，生成式 AI 专项监管政策，正在推动 AI 代码安全治理。我国正在围绕生成式人工智能快速建立监管体系。已陆续发布了《生成式人工智能服务管理暂行办法》及安全基本要求（TC260-003）、强制标准 GB 45438—2025《网络安全技术 人工智能生成合成内容标识方法》等政策和标准。此外，随着 Agent 的大规模部署和应用、AI Agent、AI Coding 工具，国家层面已正式立项了强制标准《智能体应用安全基本要求》，将 AI 安全治理从模型安全扩展到智能体安全。

AI 生成代码作为生成式 AI 服务的一部分，必须确保符合 AI 专项合规要求。尽管现阶段尚未对 AIBOM 或 AI 辅助开发审计提出统一强制要求，但总体来看，监管趋势已更加关注 AI 系统全生命周期的可追溯性、模型来源可信性、开发过程审计留痕以及软件供应链安全治理。

第三，中国软件供应链安全治理持续加强。近年来，我国人工智能、软件安全及软件供应链安全治理体系持续完善，特别是 2026 年以来，相关标准规范呈现加速细化和体系化发展的趋势，安全治理要求正从宏观原则和基础要求，进一步向可执行、可评估、可验证的技术要求延伸。

软件供应链安全方面，国家发布了 GB/T 43698—2024《网络安全技术 软件供应链安全要求》，明确了软件全生命周期的供应链安全指标；2026 年 3 月底，国务院发布了《产业链供应链安全规定》（国务院令 第 834 号），将供应链安全升级为行政法规，从宏观层面要求企业与科研机构完善风险防控体系，建

立“关键领域产业链供应链安全风险监测预警制度”。

软件安全开发标准也正在进一步向能力评估和过程规范延伸。2026 年 4 月发布的 GB/T 47470—2026《网络安全技术 软件安全开发能力评估准则》将于 2026 年 11 月 1 日起实施，为软件安全开发能力提供标准化的评估依据；GB/T 47020—2026《网络安全技术 软件物料清单数据格式》已于 2026 年 8 月 1 日起实施，进一步统一 SBOM 数据格式，为软件组成透明化、供应链风险识别和追溯提供标准基础。与此同时，安全治理对象正在从传统软件进一步延伸至 AI 驱动的软件开发场景。2026 年 5 月下达的《网络安全技术 人工智能代码生成服务安全要求》国家标准计划，目前已进入起草阶段。

总体来看，我国 AI 开发安全治理正在形成“政策法规—基础标准—能力评估—技术规范—测试评价”的多层次治理体系，治理对象也由传统软件及软件供应链逐步延伸至 AI 生成代码、AI 编程工具和 Coding Agent 等新型开发场景。

7.1.3 开源组织，推动 AI 安全标准和可信软件供应链体系建设

国际开源安全组织正在加速完善 AI 安全风险框架、供应链透明标准以及开发安全实践指南，其中以 OWASP、CycloneDX、OpenSSF 等组织最具代表性。

国际开源安全组织正在加速完善AI安全风险框架、供应链透明标准以及开发安全实践指南

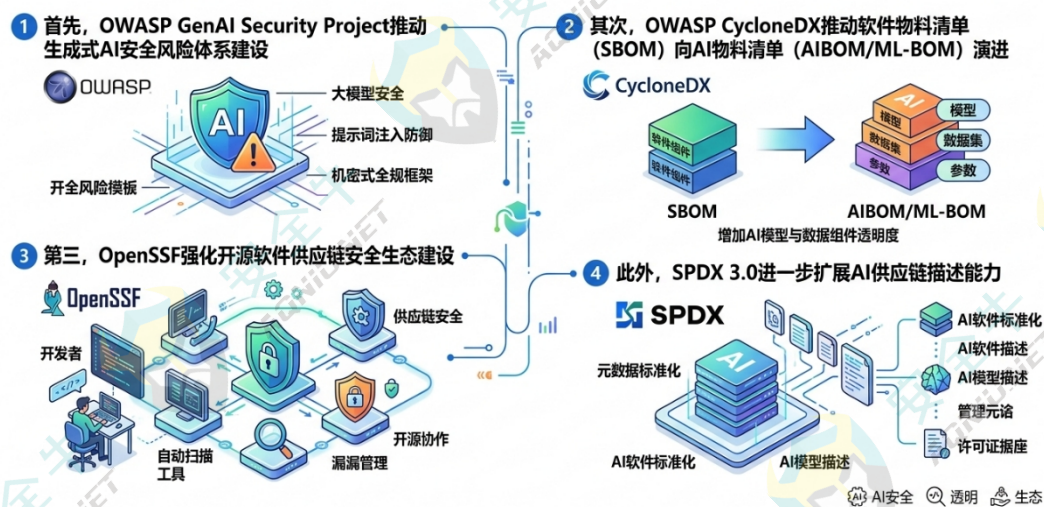


图 35 开源组织

首先，OWASP GenAI Security Project 推动生成式 AI 安全风险体系建设。OWASP 围绕生成式 AI 应用安全持续完善风险框架，先后发布了《OWASP Top 10 for LLM Applications》《OWASP Top 10 for Agentic Applications》以及《OWASP Agentic Skills Top 10》等项目，分别将大语言模型、AI Agent 以及 Agent 扩展能力引入传统应用安全治理体系。其关注范围已由早期的模型输出安全，逐步扩展至 AI 应用生命周期安全，包括提示注入、数据泄露、权限滥用、Agent 自治行为风险、AI 安全测试以及红队