安天在代码安全领域的新进展及引入大模型的思考

安天代码安全中心

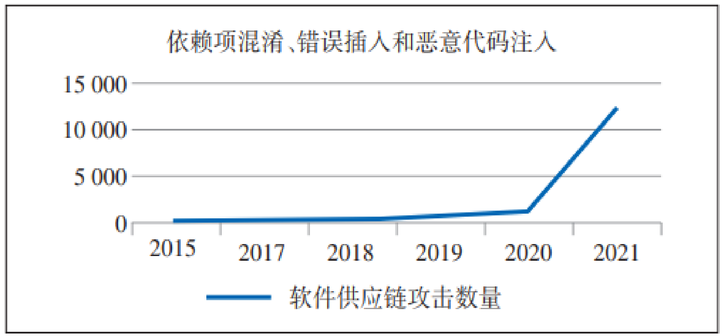
执行体治理赋能与大模型辅助

北向守望

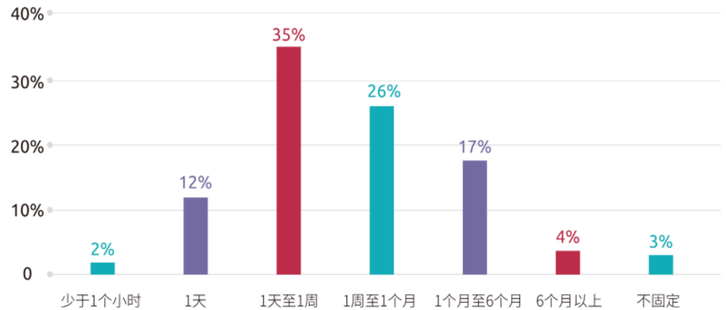
01 企业面临的挑战和需求

软件供应链的安全挑战

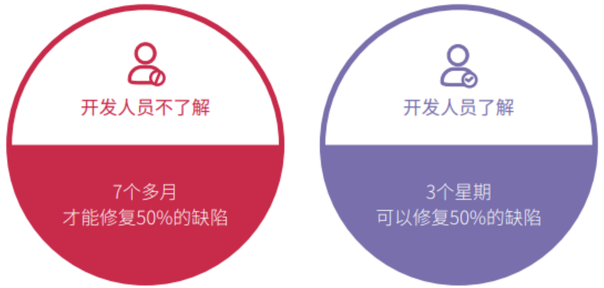
软件供应链攻击趋势（2015—2021）



从漏洞发现到修复所用的时间



了解漏洞信息对修复时间的影响



国际竞争环境加剧

在全球软件供应链安全问题日益突出，国际 IT 技术竞争激烈的背景下，对我国自主研发生产相关软硬件、研发创新技术及提高 IT 技术实力提出了新要求，同时需要做好软件供应链战略计划，确保我国软件供应链自主可控、安全高效。

软件开源化趋势增强，安全风险加剧

- 国内部分开源软件和开源组件还涉及到开源许可证的冲突风险和知识产权的风险，推进开源标准，完善开源知识产权和法律体系是我国实现开源发展体系化规模化的重要手段之一。
- 开源软件提高了企业软件供应链的暴露程度，国内大多数企业缺少对技术风险、法律风险和供应链风险的认知，极度缺乏专业知识和应对经验，在大量使用开源软件的情况下，可能使其成为攻击者的切入点。

难以处理敏捷开发与安全成本之间的平衡

在敏捷开发模式中，因为安全影响开发效率导致安全与开发严重割裂，传统安全团队上线前介入的模式已经严重滞后，不仅不能有效地进行系统的安全防护，同时也会影响应用的交付速度。如何在较短的开发周期内尽量完善软件功能，同时完成相对完备的安全性测试，实现质量、安全和速度的平衡是现阶段各企业及开发部门所面临的重要挑战。

数据来源：软件供应链安全治理与运营白皮书（2022）

客户侧痛点

软件开发外包风险

第三方风险挑战

依赖第三方供应商的产品或服务可能成为供应链攻击的目标。攻击者可以通过植入恶意软件、篡改产品或利用供应商的弱点来渗透到组织的网络中，从而导致数据泄露、服务中断或其他安全问题

企业由于第三方服务供应商遭网络攻击而受损

（近几年）44%→61%（2022年）

合规风险

- 供应商是否符合如SOC2等审计标准
- 否遵从当地的法律法规

地缘政治风险

供应链厂商可能随时因受到涉及国家/地区的相关法规限制被动禁售。

数据来源：世界经济论坛发布的《2022年全球网络安全展望》

客户侧痛点

行业合规性相关标准起步较晚

国内主要相关标准文件

- GB/T 36637 - 2018 信息安全技术 ICT供应链安全风险管理指南
- GB/T 38674 - 2020 信息安全技术 应用软件安全编程指南
- GB/T 33770.1 - 2017 信息技术服务外包要求
- GB/T 31506-2022 信息安全技术 **政务网站系统安全指南** 2022-11-01 施行
- GB/T 39204-2022 《信息安全技术关键信息基础设施安全保护要求》

国外主要相关标准文件

- ISO 28001 Security management systems for the supply chain
- ISO/IEC 27036 - 2 Information technology – Security techniques – Information security for supplier relationships
- ISO/IEC 27036 - 3 Information technology – Security techniques – Information security for supplier relationships
- NIST 800 - 161 Supply Chain Risk Management Practices for Federal Information Systems and Organizations

其他行业标准:

- 企业集团财务公司监管评级办法：信息科技管理权重提升
- GB/T 42927-2023《金融行业开源软件测评规范》
- 2022年11月16日，国家能源局发布了《电力行业网络安全管理办法》
- 税总信息办便函〔2023〕47号（非公开）：要求税务机构加强供应链安全的建设

客户侧痛点

各行业供应链投毒频发

半导体/智能制造

（2021 年 8 月，Realtek WIFIi 模块 SDK 漏洞事件。台湾芯片设计厂商 Realtek 称，其 WiFi 模块的三款开发包 (SDK)中存在 4 个严重漏洞，攻击者可利用这些漏洞攻陷目标设备并以最高权限执行任意代码。SDK 用于至少 65 家厂商制造的近 200 款物联网设备中。

系统管理软件供应商

2021 年 7 月，REvail 勒索软件攻击事件。攻击者获得 Kaseya 公司后端设施访问权限，在运行于客户现场的安全事件响应工具 VSA 服务器上部署 REvil 勒索软件。通过 VSA 服务器将勒索软件安装到联网工作站，从而感染其它第三方企业网络。攻击发生前，互联网上处于联网状态的 VSA 服务器超过 2200 台。

能源行业

2020年12月，SUNBURS后门首次被披露，该攻击利用流行的SolarWinds IT监控和管理套件传播木马。事件已确认受害的重要机构至少200家，波及北美、欧洲等全球重要的敏感机构，包括美国国务院、国防部、财政部、国土安全部等。

金融

2023年11月10日，某行在美全资子公司——在官网发布声明称，美东时间11月8日，遭受勒索软件攻击，导致部分系统中断。

政企

感染SolarWinds的美国政府机构包括美国国务院、国防部、财政部、国土安全部、能源部国家核安全局、商务部国家电信与信息管理局、美国国立卫生研究院等。

02

产品侧的能力迭代

我司产品能力的演进

2.0.1

SAST检测能力增强	升级Java引擎，支持二进制文件检测，完善GB和GJB规则库，支持接入编译流程扫描；升级C/C++引擎，支持规则和输出污点路径，支持接入编译流程扫描
SCA检测能力增强	新增支持IDE插件扫描；支持新版本依赖关系图；增加许可证及作者信息的扫描和报告展示
在线报告增强	新增支持支持生成word格式的扫描报告
新增资产中心模块	支持通过容器和CI/CD方式接入资产，支持发起定时任务和漏洞趋势统计
新增漏洞中心模块	提供查询常见的漏洞资源
用户体系升级	接入更普适的Ldap用户体系
新增全局搜索功能	支持搜索任务、组件、组件漏洞、源码漏洞、许可信息
系统维护升级增强	支持一键部署；支持漏洞库数据一键升级，提升数据更新效率

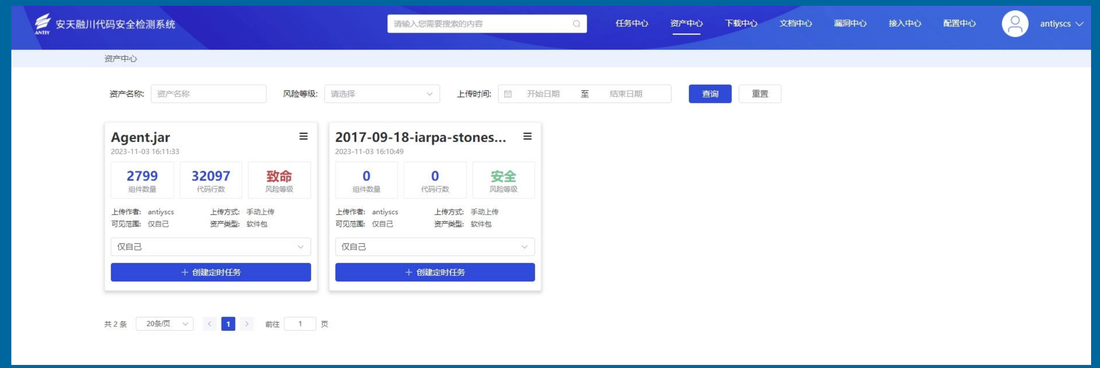
2.0.2

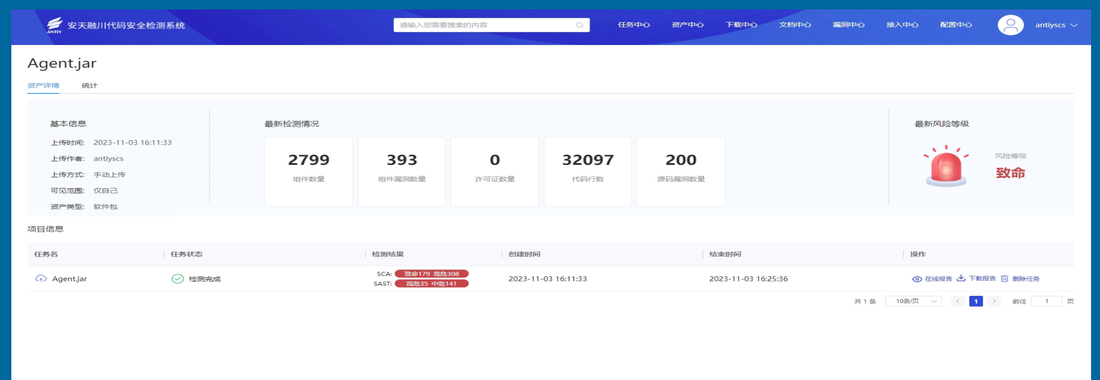
SAST检测能力增强	新增接入GJB、GB合规标准（Java、C/C++），提升合规检测能力；强化SAST模块的C/C++、Java语言扫描能力，提供自定义规则
SCA检测能力增强	支持分析生成软件物料清单
在线报告增强	新增增量扫描能力和展示，提升使用效率
系统维护升级增强	新增系统主程序升级功能和IP配置功能，提升配置和维护升级效率
第三方工具集成	新增SCA与SAST模块与Gitlab的集成功能，实现在Gitlab展示扫描结果
用户体验优化	优化了容器镜像扫描接入的交互等

2.0.3

SAST检测能力增强	新增C#、Object-C、Python语言的高精度扫描；优化SAST模块的JS/Go/Ruby/PHP语言的粗精度扫描为独立拆分扫描；新增Object-C语言通过CLI方式应用扫描；新增Visual Studio Code IDE扫描插件；新增部分CERT-C规则、完善SAST GJB规则；新增web页面配置引擎参数功能；修复SAST模块的Python引擎的并发问题
SCA检测能力增强	新增C/C++语言扫描能力
在线报告增强	优化SAST 扫描在线报告的路径跟踪图和路径跟踪表；优化扫描日志的Error显示
新增漏洞审计功能	新增源码漏洞和组件漏洞修改危险等级的功能，以排查误报；并优化漏洞统计在线报告的漏洞数量跟随审计动态变化
用户体验优化	优化资产中心交互，以降低操作学习成本

产品的能力新增-资产中心模块





功能价值

- 资产管理
- 资产跟踪
- 从用户资产的视角出发、更贴合用户的使用场景及使用习惯
- 资产定时排查、定时上报整改

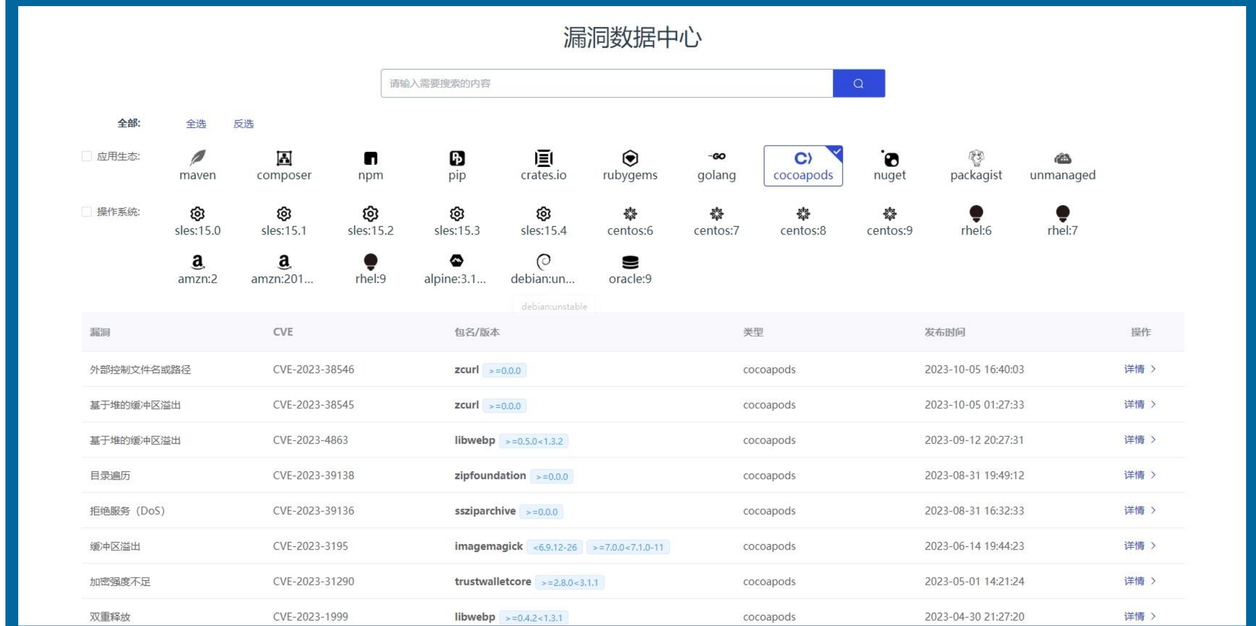
核心痛点

客户软件供应链资产不透明

资产追踪和定位难

资产定时检测维护困难

产品的能力新强-漏洞中心模块



功能价值

- 漏洞发现：跟踪和管理漏洞信息
- 安全风险管理：漏洞中心提供了一个集中的平台，用于跟踪和管理漏洞信息
- 有助于及早识别潜在的安全风险
- 合规性和审计支持：漏洞中心记录和跟踪漏洞修复的过程和结果，为合规性和审计提供支持

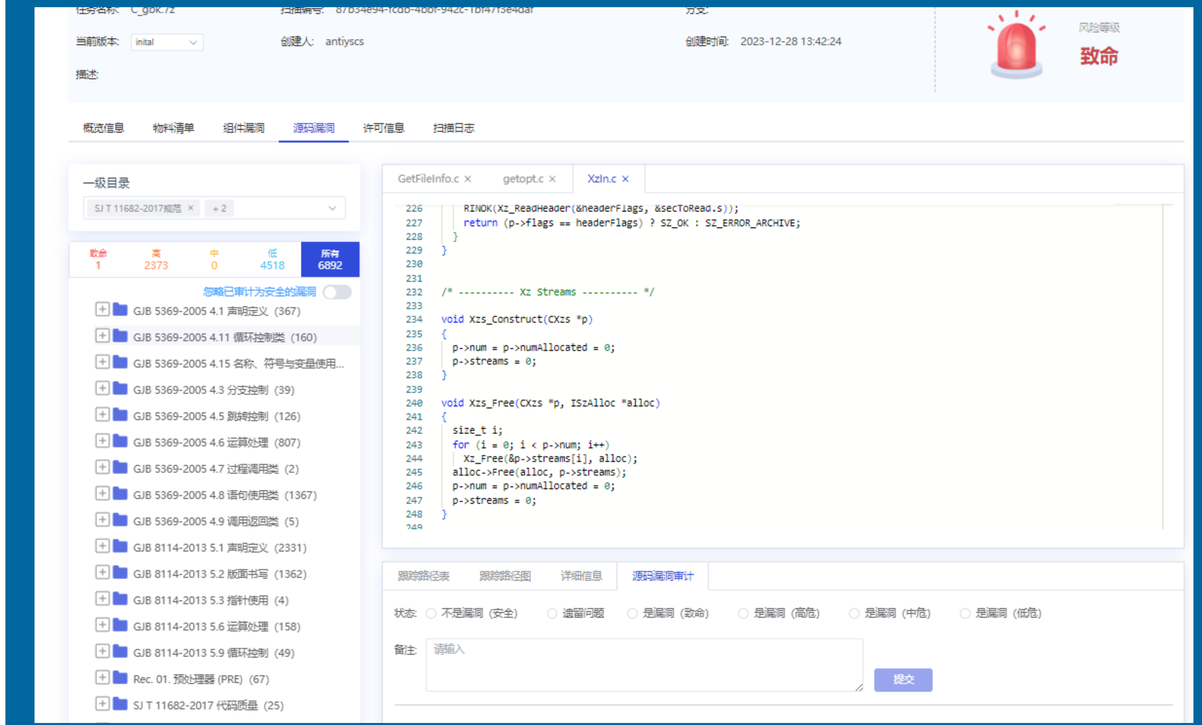
核心痛点

漏洞历史溯源困难

漏洞修复和补丁管理滞后

软件安全缺陷修复成本高

产品的能力新强-漏洞审计



功能价值

- 漏洞识别和评估：帮助组织发现存在的漏洞和安全弱点
- 安全合规性
- 安全策略评估：漏洞审计功能可以对组织的安全策略和措施进行评估
- 审计和报告

核心痛点

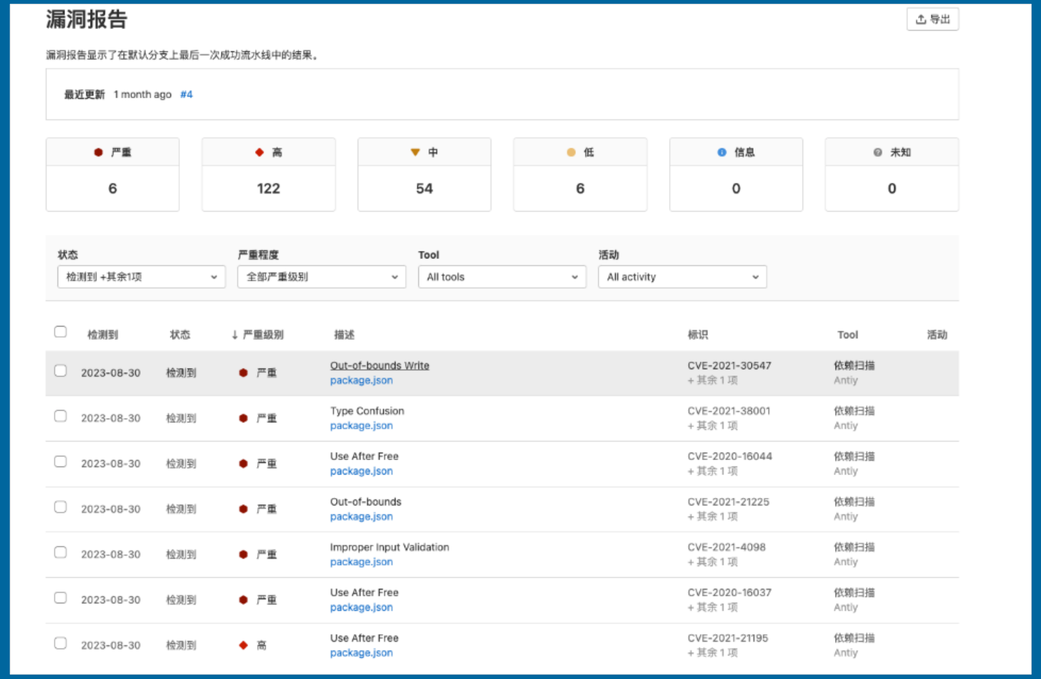
漏洞识别和评估不透明

漏洞人工勘误无法留痕

漏洞研判不能闭环

第三方工具集成能力新增-GitLab

与GitLab联动



漏洞报告

导出

漏洞报告显示了在默认分支上最后一次成功流水线中的结果。

最近更新 1 month ago #4

严重	高	中	低	信息	未知
6	122	54	6	0	0

状态：检测到 +其余1项　严重程度：全部严重级别　Tool：All tools　活动：All activity

检测到	状态	严重级别	描述	标识	Tool	活动
2023-08-30	检测到	严重	Out-of-bounds Write package.json	CVE-2021-30547 + 其余 1 项	依赖扫描 Antiy	
2023-08-30	检测到	严重	Type Confusion package.json	CVE-2021-38001 + 其余 1 项	依赖扫描 Antiy	
2023-08-30	检测到	严重	Use After Free package.json	CVE-2020-16044 + 其余 1 项	依赖扫描 Antiy	
2023-08-30	检测到	严重	Out-of-bounds package.json	CVE-2021-21225 + 其余 1 项	依赖扫描 Antiy	
2023-08-30	检测到	严重	Improper Input Validation package.json	CVE-2021-4098 + 其余 1 项	依赖扫描 Antiy	
2023-08-30	检测到	严重	Use After Free package.json	CVE-2020-16037 + 其余 1 项	依赖扫描 Antiy	
2023-08-30	检测到	高	Use After Free package.json	CVE-2021-21195 + 其余 1 项	依赖扫描 Antiy	

功能价值

- 协作和代码审查（SCA + SAST）
- 版本控制和代码管理
- 自动化构建和持续集成：GitLab集成了自动化构建和持续集成（CI/CD）功能，可以自动构建、测试和部署代码
- 自动化流水线检测

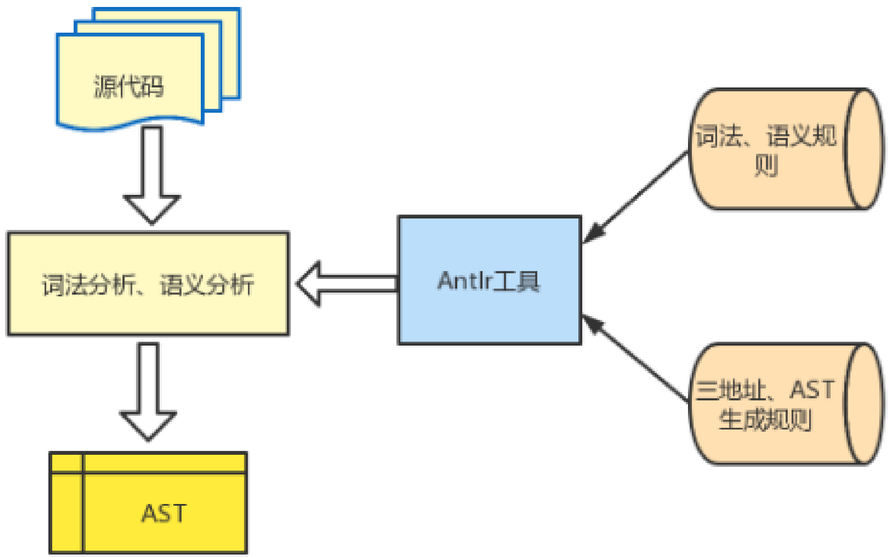
核心痛点

缺乏自动化和持续集成

缺乏集成的问题跟踪和修复流程

不完整的代码库分析

语义分析能力加强



词法分析及语义分析原理图

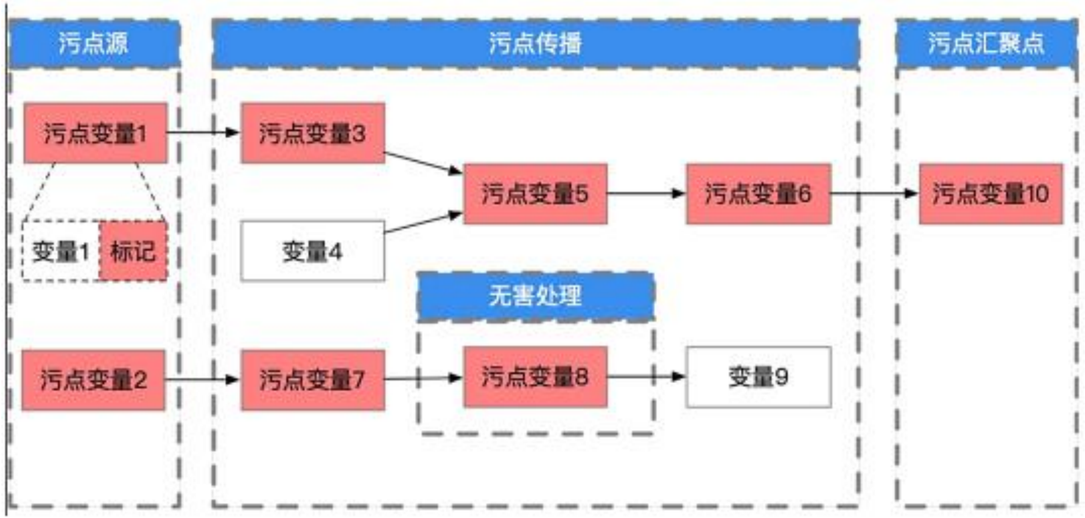
语义分析能力优化前存在的问题：
1. 对于一些大型项目或者特殊项目，**由于权限的划分或者编译环境的特殊性，会导致代码无法接入精确的数据流分析。**
2.或者是C/C++这一类的语言，由于语言特性，在**拿不到全部代码定义**的时候，甚至会导致语法解析失败，最终代码无法分析。

我们通过三种方式引入了语义分析：
1. 实现编译前端，尽可能还原出代码含义并进行数据流分析
2. call-site（调用点）分析，分析参数内容和可疑的定义
3. 基于基本机器学习的方式

语义分析能力提升后：

1. 提升代码不全时候的扫描能力
2. 无需代码编译，在丢失一定精度的情况下，允许我们对代码进行扫描。

精确化的数据流分析



污点传播

精确化的数据流分析：污点源标记（降低误报）

传统数据流分析中存在的问题：
传统数据流分析中，由于数据流传播的特性，会导致无法区分一条传播路径是否是真实的，由此会导致误报。

改进方式：
在进行污点传播的过程中，我们采取了一系列措施以确保有效区分**污点源**。我们对污点源函数、污点传播函数以及污点过滤函数进行了定义，并在其中引入了标签条件运算。这些运算符包括NOT、AND和OR，确保污点标签能够顺利传递，并在传播过程中进行适当的过滤。**仅允许符合污点标记的污点源被认定为有可能流向汇聚函数。**

相同样本，在引入精确化的数据流分析后分析表现结果集统计：

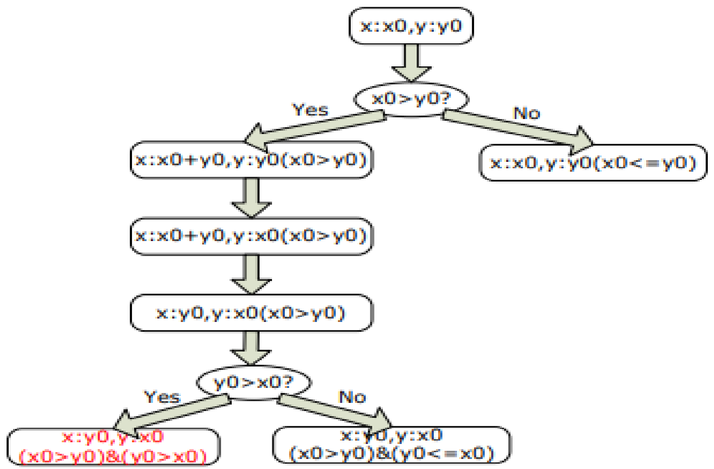
误报变化总计：减少了102次
漏报变化总计：减少了507次
平均误报变化：每个样例平均减少5.1次
平均漏报变化：每个样例平均减少25.35次

符号执行技术

符号执行求解步骤如下：

1. 为变量 x 和 y 分配符号 x0，y0。
2. 在每个分支点，PC 根据输入的假设确定不同的值。
3. 更新 PC，判断各个路径可达性。

最后可以得出如下的符号执行树。



我们可以看到，符号执行技术发现图中的红色路径是不可达的，也就是 foo1 永远不会被执行，因为 x0>y0 与 y0>x0 是矛盾

符号执行：降低误报，“动”+“静”结合

【路径探索】：
符号执行能够探索代码中的不同执行路径，包括各种条件分支和循环。**通过对每个路径进行符号执行，可以生成具有不同输入约束的程序状态**，从而更准确地模拟程序的行为。这样可以避免传统静态分析方法中的过度近似和简化，**减少误报的发生**。

【符号约束求解】：
符号执行使用符号表示来代替具体的变量值，将程序状态建模为符号约束的集合。通过符号约束求解器，可以求解这些约束以获得具体的输入值，从而探索具体的程序路径。这种符号化的方式可以更准确地对程序进行分析，避免了传统静态分析中对具体输入的近似和假设。

【路径剪枝】：
符号执行在执行路径时，可以根据路径条件进行剪枝，排除不可行的路径。例如，**当遇到无法满足的约束时，可以将该路径剪枝，避免进一步分析**。这种路径剪枝可以减少不必要的分析开销，同时减少了误报的可能性。

引用来源：https://github.com/enzet/symbolic-execution

产品在不同行业上的应用

半导体/智能制造

- IOT固件安全检测及验证
- 物联网设备和智能硬件
- 嵌入式系统开发安全

软件集成/开发供应商

- 代码审计和漏洞扫描
- 安全编码指导
- 恶意代码检测和防御

能源行业

- 工控系统安全
- 智能电网安全（关键组件的安全）
- 源供应链安全（确保关键节点和系统的安全性）
- 源设备和传感器安全（确保固件的安全性和完整性）

金融

- 金融应用程序安全
- 交易系统和支付安全
- 风险管理和合规性
- 移动银行和电子钱包安全
- 数据保护和隐私

政企

- 政府信息系统安全（用于政府信息系统的开发、部署和维护，确保系统的安全性和完整性）
- 公共安全和紧急响应（维护公共安全系统，包括安全监控、警报系统和应急响应平台等。）
- 法律合规和数据保护

北向守望

03

大模型对于漏洞修复的案例解读

文献来源说明

Examining Zero-Shot Vulnerability Repair with Large Language Models

Hammond Pearce*, Benjamin Tan†, Baleegh Ahmad*, Ramesh Karri*, Brendan Dolan-Gavitt*
*New York University, †University of Calgary

***Abstract*—Human developers can produce code with cybersecurity bugs. Can emerging 'smart' code completion tools help repair those bugs? In this work, we examine the use of large language models (LLMs) for code (such as OpenAI's Codex and AI21's Jurassic J-1) for zero-shot vulnerability repair. We investigate challenges in the design of prompts that coax LLMs into generating repaired versions of insecure code. This is difficult due to the numerous ways to phrase key information—both semantically and syntactically—with natural languages. We perform a large scale study of five commercially available, black-box, "off-the-shelf" LLMs, as well as an open-source model and our own locally-trained model, on a mix of synthetic, hand-crafted, and real-world security bug scenarios. Our experiments demonstrate that while the approach has promise (the LLMs could collectively repair 100% of our synthetically generated and hand-crafted scenarios), a qualitative evaluation of the model's performance over a corpus of historical real-world examples highlights challenges in generating functionally correct code.**

***Index Terms*—Cybersecurity, AI, code generation, CWE**

I. Introduction

Commercial large language models (LLMs), trained on vast amounts of source code, have been enthusiastically promoted as tools to help developers in general coding tasks like translating between programming languages and explaining code [1]–[4] by predicting likely text completions given some "prompt" comprising comments, function names, and other code elements. This is similar to the multi-tasking capabilities that LLMs for natural language exhibit [5], [6]. Of the many tasks coders do, we are interested in *fixing* security bugs; developers might ordinarily run security tools such as fuzzers or static analyzers, try to understand the feedback, locate the issue, and modify the code to repair the bug. This is hard.

In this paper, we ask: **Can LLMs for code *completion* help us fix security bugs** (Fig. 1)? "Out-of-the-box" LLMs for coding, such as OpenAI's Codex [7] and AI21's Jurassic-1 [8] are trained on open-source code in myriad languages that contain a large variety of comments [9]–[11] and functionality (both buggy and non-buggy). This powers the ability of LLMs to complete code in different ways, given some context such as the designer's intent in code comments.

While recent work [12] suggests that code completions with the LLM GitHub Copilot can introduce security weaknesses, Pearce *et al.* conclude that models can still "increase the productivity of software developers", especially when paired with "appropriate security-aware tooling during…generation to minimize the risk" [12]. As one can guide LLMs by adding *cues* to prompts (as suggested by user guides like [4]), we seek to characterize the feasibility of using black-box, "off-the-shelf" LLMs for "zero-shot" *generation of replacement code* for an identified security bug, perhaps as *part* of an overarching program repair framework. This contrasts prior work that trained specialized neural machine translation (NMT)-based models for fixing software bugs (e.g., [13]–[16]). Unlike prior approaches which are trained to predict the *exact same tokens* as a human developer's fix, we want to investigate off-the-shelf LLMs' apparent ability to "understand" the broader context from a source code file.

Fig. 1. Prior work suggests that large language models (LLMs) can help programmers write **functional** code. **Can they help fix security bugs too?**

We focus on creating *security* patches as they are important, can be tricky to write, and often require relatively small changes of code in a single file [17]. The smaller footprint of security patches suggests that current off-the-shelf LLMs might be capable of designing bug fixes. We want to know if LLMs—despite not being trained specifically on security fixes (and, indeed, being trained on a great deal of *insecure* code)—are nonetheless capable of generating valid fixes for vulnerable code. We seek answers to these research questions:

- **RQ1**: Can off-the-shelf[1] LLMs generate safe and functional code to fix security vulnerabilities?
- **RQ2**: Does varying the amount of context in the comments of a prompt affect the LLM's ability to suggest fixes?
- **RQ3**: What are the challenges when using LLMs to fix vulnerabilities in the real world?
- **RQ4**: How reliable are LLMs at generating repairs?

To answer these questions, we evaluate recent LLMs on a range of synthetic, hand-crafted, and real-world buggy scenarios. Our contributions are as follows.

(1) We compare prompts, contextual cues, and model settings (temperature, sampling strategy, etc.) for encouraging LLMs to generate functional and secure code. (Section III)
(2) We provide the first evaluation of LLMs for *zero-shot generation of security fixes*, showing that off-the-shelf models are capable of producing security fixes without any additional

[1]From this point on, our discussion of LLMs is specifically about off-the-shelf, general coding LLMs available at the time of this study.

中文题目：基于大语言模型的零样本漏洞修复研究
摘要：用于代码完成（Code Completion）的“开箱即用”的LLM能否帮助开发人员修复安全漏洞。

原文标题：Examining Zero-Shot Vulnerability Repair with Large Language Models

原文作者：Hammond Pearce, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, Brendan Dolan-Gavitt
发表会议：In 44th IEEE Symposium on Security and Privacy (SP 2023)
原文链接：https://arxiv.org/pdf/2112.02125.pdf
主题类型：漏洞自动化修复

公司由于安全的主要的大模型算力主要还是服务执行体的特征工程，因此，目前来看呢，我们在这个方向尝试还是依托外部的一些资源做一定的检验。

引用来源：Examining Zero-Shot Vulnerability Repair with Large Language Models

利用大语言模型，在"零样本"场景下对于漏洞修复的案例

本实验从CWE Top 25中选取2个漏洞作为人工合成程序的漏洞，分别是：1）CWE-787，越界写入（Out of bounds write）；2）CWE-89，SQL注入。
这里选择上述2个CWE的原因为：CWE-787往往发生在“低级”语言中，如C/C++可以直接操纵内存指针，而CWE-89常发生在“高级”语言如Java或Python中，这意味着本方案可以对软件开发的两个不同层面获得LLM修复漏洞能力的评估。

【1】漏洞程序生成流程

为了生成大量功能独特且漏洞相似的程序，采用LLM生成大量存在漏洞的程序作为评估数据集，采取如下步骤：

步骤1: 指定与CWE-787和CWE-89相关的存在漏洞的程序作为Prompt的开头；

步骤2: 将Prompt输入Codex大模型（分别使用code-cushman-001和code-davinci-001两个版本各生成）500个代码样本；

步骤3: 对这些代码样本进行单元测试**（漏洞验证）**；**将功能可用且存在漏洞的程序作为数据集**，表为生成的代码样本的统计详情，生成了功能正常且存在漏洞的代码样本分别为 388（CWE-787）和23（CWE-89）；在过滤了功能相似的代码样本后，共生成CWE-787和CWE-89的漏洞程序样本分别为95个和22个。

TABLE VIII
SYNTHETIC VULNERABLE PROGRAM GENERATION RESULTS

Scenario	# Gen.	# Vld.	# Fn.	# Vuln.	# Fn. & Vuln.	# Fn. & Safe.
CWE-787	500	440	410	452	388	22
CWE-89	500	500	491	23	23	468

Gen. (Generated), Vld. (compilable), Vuln. (Vulnerable), Fn. (Functional), Safe (Not Vulnerable)

引用来源：Examining Zero-Shot Vulnerability Repair with Large Language Models

利用大语言模型，在"零样本"场景下对于漏洞修复的案例

【2】漏洞程序修复流程

利用【1】漏洞程序生成流程将95个CWE-787和22个CWE-89漏洞程序代码作为Prmopt

步骤1: 分别向code-cushman-001和code-davinci-001两个LLM查询**漏洞修复代码**

步骤2: 每个模型每次生成500个代码样本，两个漏洞分别生成了47500和11000个，表为生成代码样本的详情

步 骤 3: CWE-787场景中的22034个有效程序中有2.2%被修复，而CWE-89场景的10796个有效程序中有29.6%被修复。

LLM即使在没有额外的上下文假设的前提下，也能够零样本（Zero-shot）生成无Bug代码，即在漏洞修复的下游任务中用于自动化漏洞修复。

TABLE II
SYNTHETIC PROGRAM REPAIR RESULTS. HIGHER VALID REPAIR PERCENTAGES (I.E. '# FN. & SAFE' / '# VLD.') ARE BETTER.

Scenario	# Gen.	# Vld.	# Fn.	# Vuln.	# Fn. & Vuln.	# Fn. & Safe.	% Vld. Repair
CWE-787	47500	22034	20029	21020	19538	491	2.2
CWE-89	11000	10796	7594	5719	4397	3197	29.6

Gen. (Generated), Vld. (compilable), Vuln. (Vulnerable), Fn. (Functional), Safe (Not Vulnerable)

引用来源：Examining Zero-Shot Vulnerability Repair with Large Language Models

真实CVE场景验证

【1】漏洞注入

为了进一步了解LLM用于真实场景下的代码修复能力，将开源项目中的CVE漏洞代码作为输入，评估大模型在真实软件项目中的漏洞修复的表现。这里从3个项目中收集了12个真实CVE漏洞，这些项目来自于ExtractFix数据集。表为本实验中使用的开源项目和CVE编号列表。

TABLE VI
REAL WORLD SCENARIOS (SUBSET OF EXTRACTFIX [34])

Program	Description	File	LoC	EF#	CVE
Libtiff	C library for processing TIFF files.	tiffcrop.c	9.1k	EF01	2016-5321
		thumbnail.c	683	EF02-1	2014-8128
		tif_next.c	162	EF02-2	2014-8128
		tiff2pdf.c	5.5k	EF07	2016-10094
		tif_jpeg.c	2.3k	EF08	2017-7601
		rgb2ycbcr.c	393	EF09	2016-3623
		tif_jpeg.c	2.4k	EF10	2017-7595
Libxml2	XML C parser and toolkit.	parser.c	15.7k	EF15	2016-1838
		parser.c	15.4k	EF17	2012-5134
		valid.c	7k	EF18	2017-5969
Libjpeg-turbo	C library for manipulating JPEG files.	wrbmp.c	557	EF20	2018-19664
		jdmarker.c	1.3k	EF22	2012-2806

引用来源：Examining Zero-Shot Vulnerability Repair with Large Language Models

真实CVE场景验证

【2】关键挑战

关键挑战是：由于每个LLM的令牌长度限制，以及开源项目的代码量普遍较大，为大模型提供源代码文件中完整的上下文信息，因此需要引入代码压缩步骤，以使得Prompt符合令牌限制的要求，具体如下：

步骤1： 根据LLM的用户指南，基于“开发人员本应知晓的”上下文，在Prompt开头包含文件的#define等信息；

步骤2： 跳转到源代码中与漏洞相关的函数开头，将代码添加到Prompt中，直到漏洞点所在的代码行，并采用下表所列的2种Prompt模版构建Prompt；

步骤3： 如果Prompt仍超出令牌最大长度，则截断Prompt直到符合令牌限制。

Template ID	Description
`c.a.`	Commented Code (alternative) - *Used for real-world examples, see Section V).* As with `c.`, but commented in the alternative style for C programs (i.e., in C commenting, `/*` and `*/` rather than `//`).
`c.n.`	Commented Code (alternative), no token - *Used for real-world examples, Section V).* As with `c.a.`, but with no ‘first token’ from vulnerable code.

【3】实验结果

12个CVE漏洞中的8个被至少一个LLM修复，即代码通过了与修复前相同的功能测试与安全测试过程。这表明，LLM在零样本状态下修复漏洞的能力与此前最先进的漏洞修复工具ExtractFix相媲美（后者修复率为10/12），尽管人工检查后发现一些LLM修复的代码中引入了新的Bug，但是通过Prmopt中给出的有限的上下文，这样的表现仍展现了LLM在漏洞修复任务中的前景，LLM甚至还修复一个ExtractFix也无法修复的漏洞（CVE-2018-19664）

引用来源：Examining Zero-Shot Vulnerability Repair with Large Language Models

04 总结与展望

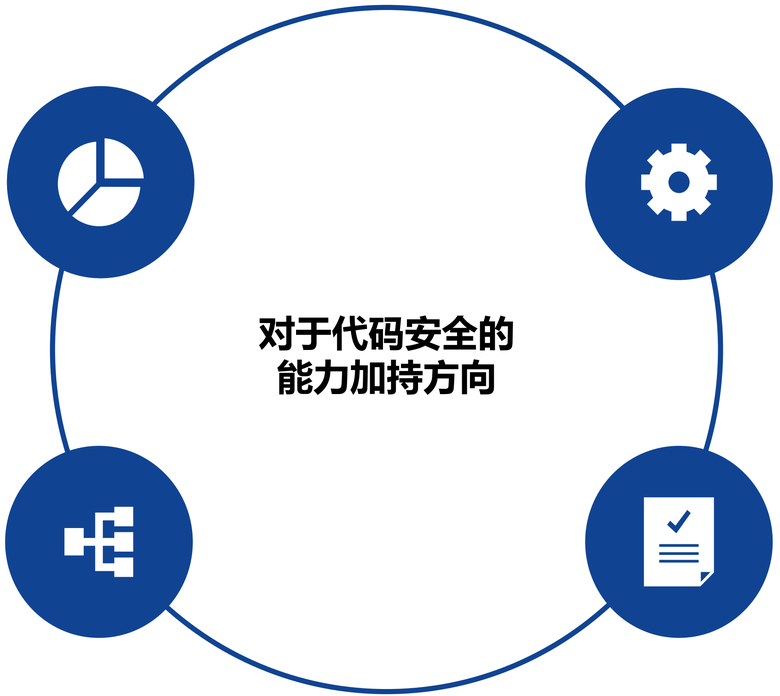
引入大模型在代码安全领域的能力提升总结

代码生成和审计

可以用于生成安全建议、编写安全文档或自动化安全审计，可以提供更全面和准确的代码审计和安全评估

安全规则和模式识别

SAST工具依赖于预定义的安全规则和模式进行代码分析。大语言模型可以用于**自动学习和生成这些规则和模式**，以帮助改进SAST工具的准确性和覆盖范围。可以提供更智能和自适应的安全分析能力



漏洞程序修复

（1）LLM，对源代码预训练，用于代码补全等任务
（2）基于深度学习的自动程序修复（DL-based APR）方法

提升产品的安全能力

例如攻击者通常会通过样本的快速变种绕过安全产品的防护，通过大语言模型的辅助安全产品可以快速生成规则，提升样本的检出率

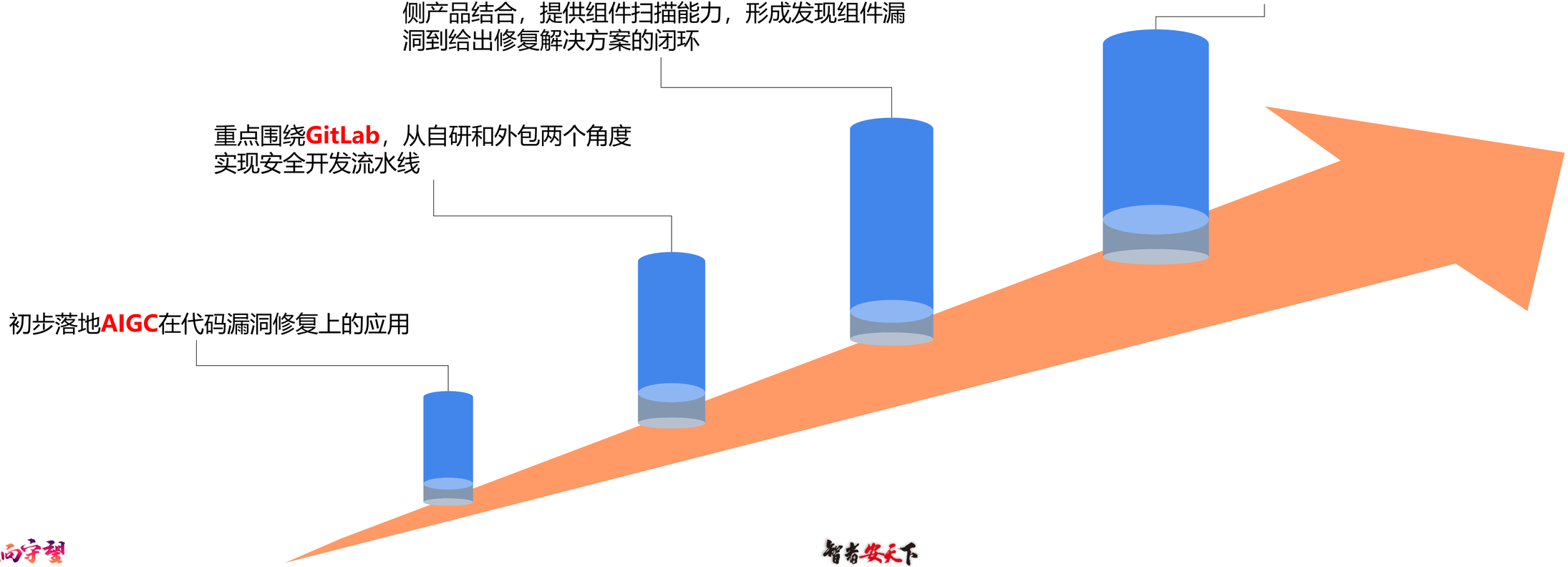
我司代码安全产品的发展方向及展望

初步落地**AIGC**在代码漏洞修复上的应用

重点围绕**GitLab**，从自研和外包两个角度实现安全开发流水线

以**执行体**为中心，SCA组件和与智甲和WAF等流量侧产品结合，提供组件扫描能力，形成发现组件漏洞到给出修复解决方案的闭环

以AIGC为辅，逐步具备漏洞管理和修复的能力，形成一套**安全开发平台**



网络空间威胁对抗与防御技术研讨会
暨 第十一届安天网络安全冬训营

持续努力，不断创新

安天冬训营 wtc.antiy.cn

执行体治理赋能与大模型辅助

北向守望