用于调查代码仓库安全趋势的静态分析平台

Tim Sonnekalb、Christopher-Tobias Knaust、Bernd Gruner、Clemens-Alexander Brust德国航空航天中心（*DLR*）数据科学研究所德国耶拿{firstname.lastname}@dlr.de

Lynn von Kurnatowski, Andreas Schreiber德国航空航天中心（*DLR*）软件技术研究所德国科隆{名.姓}@dlr.de

Thomas S. Heinze格拉-艾森纳赫合作大学德国格拉 thomas.heinze@dhge.de

Patrick M¨ader数据密集型系统与可视化小组 伊尔梅瑙工业大学德国伊尔梅瑙 patrick.maeder@tu-ilmenau.de

arXiv:2304.01725v1 [cs.SE] 4 Apr 2023

摘要—**静态分析工具形式多样，配置各异，可用于处理（安全）开发流程中的各种任务：代码风格检查、错误/漏洞检测、验证等，并适应软件项目的具体要求，从而减少误报数量。**

丰富的配置选项给软件开发人员使用这些工具带来了障碍，因为工具无法开箱即用。然而，只有将静态分析工具集成到软件开发工作流中并定期使用，才能充分发挥其效益。例如，应通过版本历史集成漏洞管理，以识别热点。

我们提出了一种分析平台，它集成了多种静态分析工具，使基于 Git 的代码仓库能够持续监控其版本历史中的警告。该框架可以轻松扩展，支持其他工具和编程语言。我们提供了一个可视化组件，以仪表板的形式展示安全趋势和热点。我们的工具还可用于大规模创建安全警报数据库，非常适合用于机器学习应用，例如缺陷或漏洞检测。

索引词—**静态分析、软件安全、自动分析、静态应用安全测试工具数据库、安全仪表板**

I. 引言

静态分析安全测试（SAST）工具为查找代码中的缺陷和代码缺陷提供了强大手段。这些工具功能丰富，涵盖从代码风格检查到对底层代码进行形式化验证等多种能力。它们支持详细的配置设置和自定义规则，以适应软件项目的特征并最大限度地提高检测率。

相比之下，一项大规模研究发现，很少有开源项目将 SAST 工具纳入其开发工作流 [1]。研究还表明，即使使用了 SAST 工具，通常也不会调整默认配置。编写自定义规则时，这些规则的价值仅占全部规则总价值的一小部分。此外，该研究还发现，在整个开发周期内，大多数配置文件都不会再被调整。这样一来，静态分析就无法有效发挥作用，因为配置与软件的需求不匹配。

Johnson 等人 [2] 在访谈中发现，开发人员认为使用这些工具过于繁琐，而且由于时间限制，不会考虑使用缺陷跟踪工具。如果这些工具已集成到开发流程中，并且工具输出呈现得当，一些开发人员便会使用它们。

为了减少开发人员使用这些代码审查工具所需的大部分工作，我们开发了一个在线静态分析平台，它可以轻松集成到基于 Git 的流程中，按固定间隔运行，并通过仪表板展示结果。此外，我们的分析平台可用于大规模挖掘静态应用程序安全测试警告，并利用综合数据库开展进一步的数据科学应用，例如实证漏洞检测。

我们设计了一个满足以下要求的静态应用程序安全测试分析平台：

持续在线监控 我们的分析会按固定间隔检查给定代码库 URL 是否有新的提交，并对其进行分析，以支持 DevOps。**高覆盖率** 由于我们采用了多种运行模式各异的 SAST 工具，因此检测到缺陷的概率更高，从而最大限度地减少假阴性的数量[13]。**虚拟化** 我们将分析平台部署为 Docker 镜像，以便以操作系统无关的方式运行。**模块化设计** 我们的实现支持轻松扩展各种 SAST 工具和编程语言。**基于 Git** 我们使用广泛采用的 Git 版本控制系统，因此可以对所有基于 Git 的代码仓库进行分析。**挖掘／可视化** 分析结果会写入数据库，并在仪表板中可视化。

Schreiber 等人 [3] 以德国新冠接触者追踪应用的开源代码库为案例研究，对该平台进行了测试。他们还提供了一个数据库 [4] ，可供使用。

通过我们的仪表板。Schreiber 等人 [5] 讨论了软件组件可视化的要素，包括静态应用程序安全测试警告，我们以此为基础构建了仪表板。我们在这项工作中的贡献是实现并发布代码，以供后续工作使用。我们通过 GitLab [1]提供分析平台、仪表板和 Dockerfile 的源代码。

II. 静态应用程序安全测试平台设计

我们的分析平台组件如图 1 所示。从 Git URL 开始，脚本会下载整个代码仓库，并提取指定提交时的代码仓库状态。首次运行分析时，会按时间顺序下载并分析各次提交。此后每次运行分析，都会将自上次运行以来新增的提交加入分析队列。分析结果经过解析后写入数据库。此外，结果还可以显示在我们的仪表板中，也可以从数据库中提取，并使用 Grafana 等其他可视化工具进行展示 [2]。

A. SAST 工具

我们的平台目前支持五种 SAST 工具。表 I 列出了已实现的工具及其类别、所分析的编程语言和预定义配置。这些类别涵盖了从编码规则到形式化验证的不同领域。在编程语言方面，本文主要关注安卓和 Java 的分析。我们根据这些工具是否适合分析应用程序的源代码来进行选择。

我们调整了这些工具的配置（见表 I），使其发出与安全相关的警报，以检测漏洞。用户可以根据项目的具体情况调整该工具。开发者仍需手动生成最佳可用配置，但借助我们的平台，只需几个简单步骤，就能在整个开发过程中对其进行测试。

分析本身在 Docker 容器中运行，以实现平台独立性。要启动分析，脚本只需要 Git URL 和待分析的编程语言。我们提供了相关说明，可将脚本作为 systemd 服务运行，以定期更新数据库。我们的脚本使用 Python 编写，因此可以轻松调整，以用于更多数据科学应用。我们可以将多个代码仓库加入分析队列。工具的执行不会影响仪表板中的结果输出，因为平台的这两个部分独立运行。理想情况下，分析服务运行在服务器上，而结果则显示在浏览器中用户客户端。

我们使用的工具对待分析软件有不同的要求：它们可能使用纯源代码、编译后的目标代码，或者将分析过程直接集成到编译流程中。目前，我们的平台通过检查主文件夹或子文件夹中是否存在相应文件，并执行相关的构建命令，支持 Gradle 和 Maven 构建。每次提交后，我们都会执行进行一次干净构建，确保所有更改都已正确应用。只需在 Dockerfile 中添加工具的安装步骤，定义使用配置初始化分析的方式，并编写一个解析方法，即可轻松添加其他静态应用安全测试工具。我们在代码仓库中提供了相关说明。

我们特别重视容错性。如果静态应用安全测试工具运行失败或被中止，则会在下一次迭代中重复进行分析。如果分析反复以错误代码结束，则会跳过使用该工具的提交。

B. SAST 数据库

我们使用关系型数据库来存储静态应用安全测试警告、代码仓库和工具信息。来自多个代码仓库的警告，以及所有工具信息及其配置，均存储在六个关联表中。这使我们能够通过仪表板后端查询和分析数据。我们使用提交哈希或快照 ID 表示代码状态。正如作者针对漏洞和溯源数据的联合分析所展示的那样，关系模型也可用作其他数据挖掘和分析工具的接口 [3]。

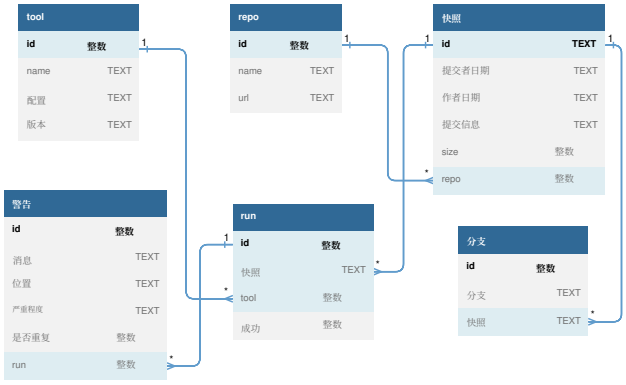


图 2. 我们的静态应用安全测试数据库的关系模式。

图 2 展示了数据库架构。它包括以下表：

代码仓库包含已提交的代码仓库名称及其 Git URL。**快照**或提交包含提交哈希、提交日期、作者名称、提交消息，以及按代码行数衡量的代码大小。**分支**将分支名称分配给特定的快照。**工具**对应一个静态应用安全测试工具，包含名称、配置和版本号。**运行**对应在给定快照上执行工具，并确认成功标志，该标志用于确认工具运行成功或指出错误 [3]。**警告**是所遇到的所有静态应用程序安全测试警告的集合，包含消息、作为文件路径的位置、严重程度（如有）以及重复项检查字段。

[1]https://gitlab.com/dlr-dw/sast-tool-platform [2]Grafana 是一款用于数据库的开源分析和监控工具。https://grafana.com/

[3]用户可以在日志文件中查看详细的错误消息。

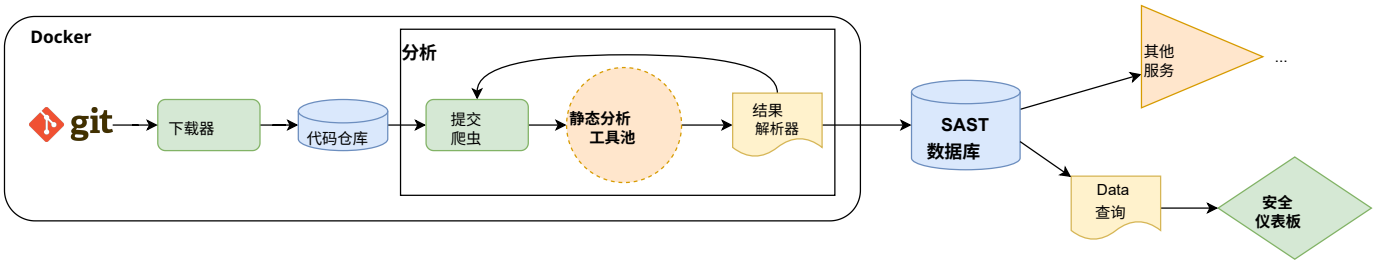


图 1. 平台概览。

表 I 纳入的静态分析工具。

静态应用安全测试工具	类别	语言	配置
FlowDroid	污点分析	安卓	*java -jar soot.jar -s ./flowdroid/SourcesAndSinks.txt -o report.xml*
Infer	形式化验证	Java、安卓、C、C++、iOS	*infer run –results-dir {resultDir} –no-fail-on-issue – {compileConfig}*
MobSF	各种	安卓、iOS、Windows	*mobsf –apikey {Key} upload {Apk} > mobsf report.json*
PMD	编码规则	Java、JavaScript	*pmd -R rulesets/java/sunsecure.xml -failOnViolation false -f json*
Xanitizer	污点分析	Java、Scala、JavaScript、TypeScript	*xanitizer* 在发现列表报告中生成详细信息=*True* 覆盖配置文件=*True*

表字段的数据类型为文本或整数。我们在表之间使用一对多关系，以便对这些需求进行建模。

工具运行完成后，结果会根据工具的不同以不同形式返回，例如以 JSON 文件形式返回，然后进行适当解析，以便通过 SQL 插入语句写入数据库。工具的输出（例如警告消息、位置和严重程度）因工具而异。例如，可以使用静态分析结果交换格式（SARIF） [6]来规范化静态应用程序安全测试警告。我们在将值写入数据库之前不会对其进行规范化，因为我们会为未来的使用保留这种灵活性。

我们选择的数据模型设计支持仪表板的可视化用例。在 警告 表中，我们可以轻松筛选警告，例如按位置或严重程度筛选。核心表是 运行，它与警告、代码仓库和快照相关联。通过联接警告 和运行.标识符，并选择快照.作者日期，可以深入了解警告的历史记录。此外，还可以使用静态应用程序安全测试数据，并通过版本历史将其与软件度量指标关联，以进行更深入的分析，例如，将代码变更量与漏洞数量相关联 [7]。

III. 仪表板

我们创建了一个仪表板，用于可视化 SAST 数据库中的安全发现。图 3 展示了一个示例。前端应帮助开发人员更快速地筛选警告，或识别存在大量错误的软件模块。此外，该仪表板还旨在展示安全趋势，表明过去是否使用过 SAST 工具，以及在版本历史过程中，整体代码质量是否有所提升。后者尤为重要，特别是对于遗留代码。目前，该仪表板主要提供三种基本可视化图表：

版本历史折线图 该图展示了版本历史中静态应用安全测试警告的数量，以呈现整体安全趋势。该图位于图 3 的左上方。

警报类型树状图 此图显示最常见的警报类型，可根据其警告消息或漏洞类型（如通用弱点枚举（如果存在））进行分类。该图位于图 3 的右上方。

热点模块图 此映射使查看者能够识别热点，即警报数量特别多的模块。

Schreiber 等人讨论了 [3]和 [5] 中的可视化组件。我们按工具展示结果，因为如果不对报告进行标准化，就无法汇总这些数值。虽然可以进行简单汇总，例如统计每个代码位置的警告数量，但如果不解析报告，我们无法排除其中包含不同缺陷的可能性。我们还列出了数据库的内容，以便在图 3 底部阅读完整消息。我们计划集成一个指向集成开发环境中代码位置的直接链接。

IV. 局限性与未来工作

我们的分析工具可直接用于大量基于 Git 的项目。不过，该平台目前专注于 Java 项目，因此仅支持其中一部分静态应用安全测试工具。未来，我们计划扩展此分析平台，纳入更多工具和其他编程语言。

目前，我们重点收集有关代码仓库版本历史的各类警告。我们不会对静态应用安全测试报告进行定性评估或规范化处理。根据所使用的工具，采用多种静态应用安全测试工具可能会导致误报和真阳性数量都增加。

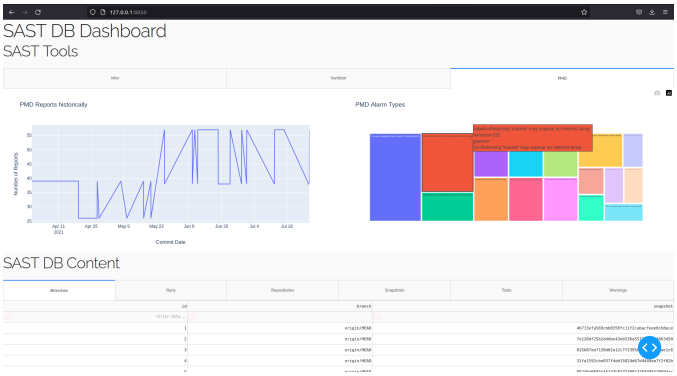


图 3. 仪表板展示。

Park 等人 [8] 开发了一种方法，通过纳入执行环境的快照来减少误报数量。我们也可以通过扩展平台并加入动态分析工具来引入这种方法。此外，Transformer 网络可以帮助确定静态应用程序安全测试警告的优先级，从而为开发者提供筛选功能 [9]。下一步，我们还计划开展用户研究，以评估仪表板中的哪些元素对开发人员有帮助，或哪些方面可以改进。我们希望通过焦点小组来开展这项研究，了解哪些呈现方式在开发团队中最有效。这种方法对于改进仪表板的用户体验设计非常有效 [10]。此外，这也有助于实现项目特定可视化。

静态分析可作为机器学习应用的真实标注 [11]。通过大规模运行我们的分析平台，可以创建全面的静态应用程序安全测试警告数据集。它还可用于对 SAST 工具进行基准测试。未来，我们将使用 SAST 数据库汇总消息，并通过计算代码和代码仓库指标，进一步提供安全与质量评估。

V. 相关工作

代码审查工具可分为 SAST、动态应用安全测试、交互式应用安全测试和运行时应用自我保护工具。SAST 工具是一种白盒测试，无需在生产环境中运行代码。它们非常适合在开发过程中进行代码检查，或用于质量保证的漏洞扫描。相比之下，动态应用安全测试（动态分析安全测试）工具对应黑盒测试，可通过执行被分析的软件来检测运行时错误。交互式应用安全测试（交互式应用安全测试）工具结合 SAST 和动态应用安全测试，并在运行时执行代码扫描。运行时应用自我保护（运行时应用自我保护）是一种采用运行时插桩技术、能够阻止网络攻击的技术。我们的平台完全基于 SAST 工具，但也可以扩展支持其他工具。

ReviewBot 是 Balachandran [12],开发的一款工具，它整合了多个静态分析工具的结果，并自动生成代码审查报告。他们通过开展访谈收集用户对审查结果的反馈，并发现大多数的自动生成的注释很有用。这些工具要么是代码检查工具，要么是模式检测工具。我们无法找到可复现此工具的实现。

Nunes 等人 [13] 将静态应用安全测试工具结合起来，用于 Web 安全应用，具体针对 WordPress 插件中的 SQL 注入和跨站脚本攻击漏洞。他们使用历史漏洞数据库来评估工具的性能。其流水线利用指标对工具进行排名并合并其输出。我们的平台针对 Git 仓库的当前状态，而非历史数据。此外，我们不局限于特定的漏洞类型，而是让工具检测所有类型的漏洞。不过，我们也未能找到他们的代码，因此无法复现其结果。

Batyuk 等人 [14] 构建了一项针对 Android 二进制文件的分析服务，可生成详细报告。这些二进制文件从 Android Market 下载后，会经过反编译并通过数据挖掘操作进行分析。我们的分析服务从源代码入手，而非应用程序的发布版本，因此使用了不同的静态应用安全测试工具。

Scrub [15],是 Source Code Review User Browser（源代码审查用户浏览器）的简称，是一种将经典的同行代码审查与机器生成的分析相结合的工具。它提供了一个类似于我们的仪表板的用户界面，开发人员可以在其中将静态应用安全测试（SAST）的结果与他们手动审查的代码进行比较。我们没有计划在概述中纳入手动审查的代码，因为这类审查总会受到开发人员经验的影响。

与此同时，GitLab [16] 和 GitHub [17] 通过内置于 CI/CD 中的静态应用安全测试工具提供安全扫描。此类分析（如我们的分析）的优势在于具有高度的可定制性，并且会输出数据库，可用于进行更详细、更个性化的评估。我们的平台还开创了先河：能够测试不同的工具配置，从而可视化随时间变化或不同软件子模块之间的安全趋势。

Arai 等人 [18] 采用游戏化方法，激励开发人员使用 SAST 工具并移除代码中的警告。FindBugs 的缺陷报告被分类为缺陷模式，并按开发者和团队分别计算得分。在下一次提交中完成修正后，得分会提高。这可能是对我们的仪表板一个有趣的扩展。

六、结论

我们提供一个开源静态分析平台，作为 Docker 容器，易于使用且只需少量工作即可扩展。我们还提供一个仪表板，用于展示版本历史中的趋势。我们公开了代码，以确保可复现性。

我们方法的新颖之处在于，它基于项目的完整开发历史，因此能够推导出安全趋势并识别热点。通过查看工具配置在代码仓库历史记录中的迄今表现，可以更轻松地对其进行测试。我们的分析平台可以帮助软件开发人员在日常工作中进行代码审查，并评估软件项目的版本历史，以开展安全和漏洞分析。

参考文献

[1] M. Beller、R. Bholanath、S. McIntosh 和 A. Zaidman，“分析静态分析的现状：对开源软件的大规模评估”，载于*2016*年*IEEE*第*23*届软件分析、演化与再工程国际会议（*SANER*），第1卷，第470–481页，IEEE，2016年。[2] B. Johnson、Y. Song、E. Murphy-Hill 和 R. Bowdidge，“软件开发人员为什么不使用静态分析工具查找缺陷？”，载于*2013*年第*35*届国际软件工程会议（*ICSE*），第672–681页，IEEE，2013年。[3] A. Schreiber、T. Sonnekalb、T. S. Heinze、L. von Kurnatowski、J. M. Gonzalez-Barahona 和 H. Packer，“基于溯源的安全审计及其在新冠肺炎接触者追踪应用中的应用”，载于数据与流程的溯源和注释（B. Glavic、V. Braganholo 和 D. Koop 编），（施普林格国际出版），第88–105页，施普林格国际出版，2021年。[4] T. Sonnekalb，“代码仓库 Luca App Android 的 SAST 数据库。” https://doi.org/10.5281/zenodo.5036046，2021年6月。[5] A. Schreiber、T. Sonnekalb 和 L. v. Kurnatowski，“面向溯源驱动的静态应用安全测试的可视化分析仪表板”，载于*2021*年*IEEE*网络安全可视化研讨会（*VizSec*），第42–46页，2021年。[6] G. Paul Anderson，“静态分析结果：一种格式与协议：SARIF 和 SASP”，2018年。[7] Y. Shin、A. Meneely、L. Williams 和 J. A. Osborne，“评估复杂度、代码变更量和开发人员活动指标作为软件漏洞指示器”，*IEEE*软件工程汇刊，第37卷，第6期，第772–787页，2010年。[8] J. Park、I. Lim 和 S. Ryu，“真实环境中 JavaScript Web应用静态分析误报问题的应对”，载于*2016*年第*38*届*IEEE/ACM*国际软件工程会议伴会（*ICSE-C*），第61–70页，IEEE，2016年。[9] T. Sonnekalb、T. S. Heinze 和 P. M¨ader，“程序代码的深度安全分析”，实证软件工程，第27卷，第1期，第1–39页，2022年。[10] L. D. Roberts、J. A. Howell 和 K. Seaman，“给我一个可定制仪表板：高等教育中的个性化学习分析仪表板”，技术、知识与学习，第22卷，333，2017年，第317页。– 。[11] R. Russell、L. Kim、L. Hamilton、T. Lazovich、J. Harer、O. Ozdemir、P. Ellingwood 和 M. McConley，“使用深度表示学习自动检测源代码中的漏洞”，载于*2018*年第*17*届*IEEE*机器学习与应用国际会议（*ICMLA*），第757–762页，IEEE，2018年。[12] V. Balachandran，“使用自动静态分析和审查者推荐，减少同行代码审查中的人工工作并提高质量”，载于*2013*年第*35*届国际软件工程会议（*ICSE*），第931–940页，IEEE，2013年。[13] P. Nunes、I. Medeiros、J. Fonseca、N. Neves、M. Correia 和 M. Vieira，“结合多种静态分析工具实现 Web 安全：一项实证研究”，载于*2017*年第*13*届欧洲可信计算会议（*EDCC*），第121–128页，2017年。[14] L. Batyuk、M. Herpich、S. A. Camtepe、K. Raddatz、A.-D. Schmidt 和 S. Albayrak，“使用静态分析自动评估并缓解安卓应用中的不受欢迎和恶意活动”，载于*2011*年第*6*届恶意与不受欢迎软件国际会议，第66–72页，2011年。[15] G. J. Holzmann，“Scrub：一种代码审查工具”，*Innov. Syst. Softw. Eng.*，第6卷，第311–318页，2010年12月。[16] “静态应用安全测试（SAST）- Gitlab。” https://docs.gitlab.com/ee/user/application security/sast/。访问日期：2022-12-13。[17] “功能 - 安全 - Github。” https://github.com/features/security。访问日期：2022-12-13。[18] S. Arai、K. Sakamoto、H. Washizaki 和 Y. Fukazawa，“激励开发人员消除缺陷模式工具警告的游戏化工具”，载于*2014*年第*6*届实证软件工程实践国际研讨会，第37–42页，IEEE，2014年。