自动化软　　件测试：从静态分析入手——当前技术水平

Yan Wu[1], Jingyi Su[2, *], David D. Moran[3], 和 Chris D. Near[4]1,2美国俄亥俄州鲍灵格林的鲍灵格林州立大学3,4Spectare-Systems, Inc.，美国 yanwu@bgsu.edu, jsu@bgsu.edu，dmoran@spectare-sys.com，cdnear@spectare-sys.com

摘要——复杂软件的大规模生产使得人工测试其中的软件漏洞变得不可能。自动化安全测试工具种类繁多，在软件开发的不同阶段发挥作用，并针对不同类别的软件漏洞。我们拥有大量自动化工具可供选择，这固然很好，但这些工具尚未得到广泛采用和认可，却是一个问题。本研究旨在探索现有技术的可能性，同时通过探讨安全测试工具及相关技术的未来，拓宽视野。

关键词—软件漏洞；软件分析工具；静态分析；动态分析；机器学习；安全分诊

I. 引言

软件在现代生活的诸多方面变得日益重要，这也使得利用软件漏洞获取机密数据、经济利益等权限变得颇具诱惑力。由于当今软件数量庞大，即使针对特定功能，完全依靠人工测试也不切实际，因此对自动化测试解决方案（静态、动态、混合等）的需求十分旺盛。对软件使用自动化软件测试工具听起来高效且前景广阔，但现实可能并非如此乐观：由于工具报告的假阳性率过高、运行工具造成中断、描述令人困惑等原因，开发人员和质量保证团队成员往往容易弃用这些工具。事实上，安全测试工具通常只针对特定编程语言和特定类别的软件漏洞，而且假阳性率相对较高，也存在难以发现的假阴性问题；这些因素都使软件开发利益相关者普遍不信任现有的安全测试工具。

语言不同会使人们彼此隔绝，并破坏巴别塔计划；在漏洞描述中使用不同的语言也会造成同样的后果。为确保人们在交流已发现的软件漏洞时能够相互理解，通用弱点枚举（CWE）已从对已报告软件漏洞零散的分类与归类，发展成一项规模庞大的社区工作。在现实中，主要的工具供应商通常都有自己的漏洞识别规则“词典”，而合规要求和市场压力则促使他们将自有规则映射到标准化规则集。这一过程留下了相当多的空白、不匹配和混乱，原因在于人们对特定漏洞的概念和本质存在不同理解。

通用漏洞和暴露（CVE）的建立早于通用弱点枚举（CWE），用于记录已报告的漏洞和暴露，供公众查阅。与通用弱点枚举（CWE）相比，每个CVE条目都是一次软件故障，这种故障会导致安全问题，并可能被恶意攻击者利用。目前，CVE维护着超过16万个条目，涵盖专有项目和开源项目。已报告的漏洞会经过收集和审查，以进行核实，随后分配一个CVE编号。商业项目与开源项目在识别和报告方面可用资源的差异，可能导致此存储库内容不完整。

本文其余部分的结构如下：第二节介绍一些相关研究；第三节展示发展机遇；第四节阐述 SATriage 的初步构想；第五节概述全文并提出一些未来工作。

第二节. 文献综述

在 HASE 2015 的主题演讲中，Paul E. Black 指出，不同的静态分析器是为不同目的而设计的，通常不会报告相同的缺陷。”他在演讲中提出了对静态分析器的 12 项期望，涉及从处理复杂环境到支持更多编程语言等方面。Black 认为，尽管静态分析器无法为我们带来乌托邦，但持续改进静态分析器仍将为软件开发社区提供有力支持。作为一位在软件保障领域经验丰富的研究人员，Black等人还指出，提升静态分析工具对软件质量的影响，需要降低使用成本并增加收益。降低成本的措施包括减少误报，以及更透明地集成到软件开发流程中。增加收益的措施包括发现更高层次、更隐蔽、更关键的缺陷，并帮助程序员理解和修复问题。

Felderer等人 [1] 对软件开发生命周期不同阶段采用的各类安全测试技术进行了全面调查，涵盖需求阶段的基于模型的安全测试、开发阶段的基于代码的测试和静态分析（SA），以及维护阶段的安全回归测试。作者还提到了选择安全测试方法的六项标准，例如攻击面、应用程序类型和支持技术等。他们还以一个三层业务应用为贯穿示例，说明如何选择合适的安全测试工具。

对于 Black等人，[2], 目标是列出一系列具体的技术方法，这些方法有望显著减少漏洞：在漏洞出现前加以阻止、在漏洞遭到利用前将其发现，或

降低漏洞造成的影响”。文中讨论了三类信息：技术方法、措施与指标，以及非技术方法。其中，详细阐述了五种技术方法，将其视为具有发展潜力的领域：形式化方法、系统级安全、增量软件分析、更成熟的领域专用软件开发框架、移动目标防御（MTD）以及自动软件多样化。

在《数十亿行代码之后》中，[3], Nunes 等人提出了一种方法，用于设计基准测试，以评估能够根据真实工作负载和不同排名指标检测 Web 应用程序漏洞的静态分析工具（SAT）。为构建工作负载，作者收集了存在漏洞的应用程序，并根据代码行是否存在漏洞为其标注，然后依据软件质量将其划分到具有代表性的使用场景中。为证明该基准测试的可行性，作者评估了五种开源 PHP 静态分析工具，以检测由 134 个 WordPress 插件组成的工作负载中的 SQL 注入和跨站脚本攻击漏洞。实验结果表明，最佳工具会因场景而异，也取决于所检测的漏洞类别。最后，作者采用相同的方法，评估了基于其实现方案得到的排名，并将其与 OWASP 的 BSA 基准测试进行了比较。

Bessey 等人 [4] 从静态分析工具供应商之一 Coverity 的角度，提供了另一种看法。作者分享了将 Coverity 从研究项目商业化并转化为产品的经验。他们从一个鲜少被提及的角度提出了不少有意思的观点，而这些观点很容易被研究人员忽视。例如，复杂静态分析工具的正确配置和使用可能会对结果产生很大影响；又如，误报可能导致恶性循环，使开发人员开始厌恶静态分析工具，甚至弃用它们。信息过多至少和毫无信息一样糟糕。

有趣的是，Sadowski 等人 [5] 从另一方——软件用户的角度提供了经验。他们列出了工程师并不总是使用静态分析工具或忽略警告的六个主要原因：未集成；不可操作；不可信；在实践中并未显现；修复成本过高；无法理解警告。从用户的角度来看，无论是工具错误地报告了问题，还是开发人员未能理解实际故障而未采取行动，结果都一样：如果“开发人员看到问题后没有采取积极行动”，那么该问题就被视为“有效误报”。相反，作者指出，即使工具“错误地报告了问题，但开发人员仍进行修复，以提高代码可读性或可维护性，这也不属于有效误报”。

尽管软件测试在实际开发周期中发挥着重要作用，但对某些开发人员而言，它可能是一项容易被误解的任务。在漏洞识别过程中，人们应当像攻击者一样思考。Potter 和 McGraw [6] 建议，为了分析风险，从系统架构和攻击者的角度采用基于风险的方法是有价值的。此外，基于风险的安全测试基准高度依赖专业知识和经验。

在 Emanuelsson 和 Nilsson 的研究中，[7], 用于检测漏洞的静态分析工具被分为四类：采用语法模式匹配的字符串与模式匹配方法、结合语义信息但不可靠的数据流分析、可靠的数据流分析，以及路径和上下文敏感的工具，还有基于模型检测技术的工具。潜在的改进方向包括代码量、交互式使用、较高的定价，以及比已知情况更多的析取缺陷。评估应贯穿整个开发过程，但具体需求和技术会因所处阶段而异。

为区分用于自动捕获漏洞的特征，Sion 等人 [8] 根据问题集（例如通用弱点枚举）评估了现有的 19 项安全设计缺陷目录，以及可能存在缺陷的软件架构。人们希望通过系统化的数据流模型表示和检测规则，针对并跟踪各项内容，从而有条理地检测安全设计缺陷。通过将安全设计缺陷标准映射到特定元素类型，可以利用这些标准自动检测特定模型中的设计缺陷。

研究人员 Johnson 等人 [9] 调查了静态分析工具未被软件开发人员广泛使用的可能原因，并提出了对这类工具的其他期望。使用静态分析工具时，开发人员希望获得更多协同开发支持、将工具更好地集成到各个开发阶段、直观地显示缺陷并详细解释缺陷、自动提供代码修复建议，以及易于使用的交互功能，以便配置工具。

Imtiaz 等人 [10] 针对栈溢出警报开展了一项研究，探究开发人员如何使用静态分析工具提供的 SO 警报。研究发现，开发人员更倾向于通过筛选警报来控制静态分析工具。由于开发人员的领域知识与静态分析工具生成的报告结果/警报之间存在不匹配和差异，误报仍是一项重大挑战。根据他们的研究，静态分析工具应增加易于使用的功能，以便更方便地检查警报并针对具体条目进行自定义。

另一个有趣但通常被忽视的话题，是软件编码中的“假阴性”问题。由于假阴性的数量难以确定，很少有研究人员关注这一领域；然而，如果这些假阴性涉及关键问题，就可能对每一款软件产品构成重大威胁。Thung 等人 [11] 从选定的开源项目中收集了已报告并修复的缺陷，并使用三种静态分析工具对其进行分析，以考察这些工具的“假阴性”情况。实验设计得很好，但效度威胁相当明显，尤其是对已报告并修复缺陷的局限性：这些缺陷可能只是所有缺陷中的一小部分。此外，静态分析工具本身并不擅长识别某些类别的缺陷，例如规范错误。

III. 开发机会探索

软件让我们的生活更加便利，无数人从事软件相关行业。应持续提供安全支持，学术界也应更加关注漏洞检测领域的未来研究。如何规范安全环境，是所有人面临的一项挑战

从业人员。贯穿软件开发生命周期，在各个阶段保护软件制品免受因疏忽、使用不当、意外交互等造成的错误影响至关重要。接下来，我们将从七个方面讨论并探索一些发展机遇。

A. 隐私

健康的软件环境只允许用户拥有有限的操作访问权限。程序运行时，会使程序暴露于安全攻击者的威胁之下。这里存在一个悖论：通常，功能越灵活、越复杂，数据交换方式的安全性就越容易受到威胁。数据加密可以使这些数据交换管道更加安全。因此，我们需要考虑项目加密和数据加密。如果程序能够直接从加密数据中提取可疑甚至不合理的部分，安全性可能会更高。即使在安全扫描过程中，也可能出现安全问题。此外，开源工具和第三方资源缺少文档，且由于更新不频繁，更容易暴露于攻击者的威胁之下。人们往往会忽视静态分析工具运行所在的开发或实施环境中的漏洞。

B. 可扩展性

为了提高软件开发效率并实现更复杂的功能，工程师通常会使用各种开发和测试工具。许多工程师会共同参与同一个开源程序，但无法保证每个人都接受了充分的安全培训。因此，建议重视安全开发，让开发人员和管理人员都意识到软件安全的必要性。人们希望开发环境能够实时提供安全修复。这类修复的敏感度和偏好应持续满足个人需求。任何能够轻松融入前沿 DevOp 的安全分析工具，都可能是其成功的关键。

人工操作中可能存在许多不可控因素。如果一部分代码被许多开发人员共同使用，它就会在每次发布时不断产生隐蔽的安全问题。每次项目版本更新都会带来复杂且继承而来的问题。继承问题是指在上一次补丁或维护期间未被检测到或未得到彻底修复的缺陷；随着源代码的常规变更，无论是否存在尚未修复的遗留代码，这些问题都可能引发其他新缺陷，并因此演变为安全问题。针对每个项目进行定制化分析，应能提高安全判断的准确性和覆盖率。

C. 混合

在硬件限制下，结合静态分析与动态分析，通常足以在不过度消耗资源或遗漏过多缺陷的情况下，最大限度地提升代码安全水平。灵活满足需求并采用混合分析方法，正日益受到重视。不过，静态分析的误报率较高，而动态分析方法则会漏掉许多潜在漏洞。

如今，与传统方法相比，机器学习（ML）使人们不必过度依赖专家经验。用于分析、包含大量有效且均衡案例的数据集，与模型准确率密切相关。采用表现良好的机器学习方法后，便不再需要耗费大量人力进行冗余的测试分析。

尽管通用攻击模式枚举与分类（CAPEC）等社区倡议试图总结攻击模式，以识别攻击路径、技术等，但实际入侵通常具有独特性和不可预见性，并会采用各种技术和路径。为保持现有工具的准确性，静态分析工具（SATs）生成的报告旨在涵盖彼此重叠及未覆盖的缺陷。除了公开报告的攻击模式来源之外，还必须考虑这些尚未发现的模式，以提高漏洞检测率。借鉴大量无需领域专业知识的开源项目，有助于减少完善漏洞目录和数据库所需的工作量。从项目中提取已标注的缺陷函数后，可自然地将其构建为输入数据集；对不同版本的源代码进行预处理后，这也是一种完善漏洞识别的方法。学习过程的这一阶段可视为补充此前合并的缺陷，并提供进一步的思路。由于大多数漏洞检测技术都是在函数级别设计的，因此并未充分考虑代码依赖关系的能力。通过抽象语法树分解代码，并将其转换为其他层级的表示形式，或许能检测出更多潜在漏洞。

根据不同分析工具的各种扫描结果，集成平台可以获得大量安全建议，但并非所有建议都能在有限时间内得到修复。因此，应仔细对这些疑似漏洞的相关性和重要性进行分类与排序。借助标准化开发流程，可以评估软件开发周期和漏洞生命周期（缺陷如何产生、被利用及得到修复）。

D. 平衡

过去，我们依赖人类专家判断某个漏洞是否会对程序造成严重影响；未来，我们也可以依靠程序来确定是否需要优先处理该漏洞。例如，如果程序无法按照设计正确传递输入，而传递后的中间输出又会被用于其他不安全场景，那么这种情况是否严重将由机器来判定。开发人员会使用各种静态分析工具，但每种工具都有其独特的限制、覆盖范围和误报率。例如，有些工具设置用于资源管理，另一些则用于安全参与。它们的性能、资源开销和配置复杂度各不相同。用户可以选择最符合自身需求的工具，或汇总多个工具给出的发现。人们往往关注后果严重程度高的漏洞，但这类问题仅占总体故障的一半左右。在安全问题方面，还需要更多关注那些相对容易利用的漏洞。

E. 多样性

虽然许多分析技术会借助领域专家评估漏洞模式及其严重程度，但所有者和开发人员的反馈对于生成更贴合具体项目的分析报告至关重要。由于背景各异的人可能希望负责制定总体建议，而业务用户和技术用户对项目的目标以及安全需求各不相同。

从任何与项目相关的来源中提取见解，都有助于发现新的漏洞模式。例如，我们可以在源代码中函数附近的注释里寻找潜在漏洞，查看项目管理平台上任何用户发布的提交说明，查阅开发人员发布讨论的问题和缺陷，甚至分析个别开发人员特有的编码模式和开发偏好。未来关于结合多种方式检测漏洞的研究，必然应纳入编程语言和人类语言分析的结果。这种做法虽有优势，但也可能生成更加多样化的安全报告。

F. 回溯

不同类型的软件或操作系统需要采用不同种类的漏洞检测偏好。随着编程语言和平台数量的增加，故障的数量和种类也随之增长。研究新旧漏洞之间的联系及其成因，有助于提升检测性能。由于漏洞可能长期未被发现，却不会影响当前功能，因此，找到一种机制来映射这些潜在缺陷是值得付出努力的。

G. 准确性

假阴性率很少受到研究，甚至鲜少被提及，但从本质上说，它理应受到学术界和从业人员的重视。隐藏或遗漏的漏洞就像定时炸弹，随时可能爆发。对于任何非本机软件程序而言，确切的假阴性率都难以确定；而这类程序通常足够复杂，因此某些类型的缺陷比其他类型更容易被隐藏。另一方面，简单的测试用例虽然能给出明确的“是”或“否”答案，但通常被认为不切实际，因此其结果可能不为从业人员所接受。Imtiaz 等人的实地研究 [11] 颇具启发性，但繁重的手工工作以及大量效度威胁，限制了可复现研究或将研究应用于更大规模项目的潜力。一种可能的方向是，向相对简单、干净且中等规模的软件片段中注入漏洞，将其作为半合成测试用例，以检验和估算假阴性率。然而，考虑到设计层面以及与规范相关的软件漏洞是造成假阴性的主要来源，效度威胁仍然存在。针对这些假阴性，形式化方法可实现可追溯性的形式化证明；教育与培训则能提升意识、培养必要技能，并对设计人员和开发人员起到检查作用，因此都是减少设计和规范相关假阴性的理想途径。

IV. 基于预期的 SATriage

由于静态分析工具得到广泛使用，商业用户和技术用户可能需要试用几种不同的工具，才能找到最符合自身需求的工具。有些工具基于开源项目，也有些基于云服务。开发环境并非限制因素，反而提供了灵活性。有些工具能够在代码编写过程中检测问题，在某些情况下，实时支持会很有帮助。这些都是用于检测程序缺陷的软件测试工具的热门功能。用户可以根据预算限制，在免费的分析工具和付费选项之间进行选择。代码扫描后生成的报告通常有助于工程师检测并确定需要处理的潜在漏洞的优先级。现有工具支持大多数编程语言，并且可以同时使用，也可以在不同的开发阶段使用。由于开发人员和安全专业人员会使用各种不同的静态分析工具，建议相关用户通过搭配使用一个或多个工具并采用特定配置，以实现最佳漏洞检测效果。

SATriage 是一种新开发的分析工具，开创了应用安全测试（AST）缺陷分析与分诊流程的自动化，使开发人员能够全面了解每项缺陷及其上下文、重要性和风险因素。SATriage 是一个智能交互平台，采用一套创新的加权算法以及专有的缺陷关系数据，准确减少误报，识别最可能遭到利用的软件缺陷，并突出特定应用程序中最严重的安全问题。SATriage 根据应用程序的上下文快速检查缺陷，并为每项缺陷确定紧急程度。由于缺陷的重要性会因应用程序架构、功能和使用模式而有显著差异，因此在分诊过程中，上下文敏感性至关重要，有助于突出对应用程序而言确实紧急的问题。

SATriage流程始于导入各种应用安全测试工具的结果，随后对结果进行标准化、校正和合并。每种工具都有自己的术语和分类体系，必须确保这些内容保持一致。据 CyberSagacity 称，各应用安全测试工具报告的问题中，超过 60% 被错误分类和／或彼此不一致。

缺陷报告导入、规范化并修正后，SATriage 会对每项缺陷进行详细分析：

- 确定每项缺陷可能造成的所有后果的发生概率和重要性。
- 进行误报评估，并提供具体的快速检查验证信息。
- 分析攻击向量和可利用性，以确定攻击发生的可能性。
- 根据应用程序上下文、严重程度、后果发生的可能性、可利用性以及对结果的置信度，对结果进行分诊和优先级排序。

SATriage 完成缺陷分析后，会按照重要性对缺陷进行优先级排序。

SATriage 会以按排名排序的列表形式提供所有结果。为确定排名结果，SATriage 会采用基于相对概率估算值的加权机制。

SATriage 的方法与其他 DevOps 工具最显著的区别，在于区分了概率与可能性。除 SATriage 之外，其他工具根据可能性对缺陷进行排序——如果某个缺陷有可能造成严重后果，就会被判定为严重缺陷。这会形成一个包含三到五个等级的排序系统。

而 SATriage 则采用一组经过计算的相对概率。SATriage 可以根据估算出的相对概率，对各个缺陷逐一排序。通常，SATriage 的分析会生成 300 到 500 个不同的结果等级。

例如，SATriage 会考量某个缺陷导致严重后果的可能性，并据此进行排序。许多可能造成严重后果的缺陷会在排序中靠后，因为它们导致严重后果的可能性较低（同时导致无关紧要问题的概率较高）。采用可能性方法的工具会发现，30% 到 50% 的缺陷都很严重（也就是说，有数万到数十万个缺陷）。采用概率方法时，被认定为严重的缺陷只有几十个；但更重要的是，你会得到一份从 1 到 N 排列的清单，告诉你应优先处理哪些缺陷，而不是面对一大堆数以万计、未根据应用程序上下文区分重要性的缺陷。

SATriage 仍在开发中，相关内容将在另一篇完整论文中详细介绍。

V. 结论

软件安全是一个需要高度重视的领域，但真正致力于支持这项工作的人还不够多。个人信息很容易暴露给安全攻击者。因此，软件安全人员肩负着保护全球隐私的重大责任。本文总结并提出了有助于提升未来软件开发安全性的若干方面，尤其是针对 SA 软件测试工具。如何在安全性与便利性之间取得平衡，同时保持软件开发流程的高度灵活性，仍然是一项挑战。此外，参与任何开发阶段的技术人员和业务用户都应在安全开发过程中保持警觉。我们还简要介绍了 SATriage 及其开发预期：这是一款更准确、可定制的软件漏洞分析工具，可用于未来的开发周期。

致谢

作者感谢 Spectare Systems, Inc. 根据与鲍灵格林州立大学签订的合同，支持本研究工作。

参考文献

[1] Michael Felderer、Matthias Buchler、Martin Johns、Achim D. Brucker、¨ Ruth Breu 和 Alexander Pretschner。《安全测试：综述》。Advances in Computers，第101卷，第1-51页。爱思唯尔，2016年。[2] Paul Black、Mark Badger、Barbara Guttman 和 Elizabeth Fong。《大幅减少软件漏洞：致白宫科学技术政策办公室的报告》。美国国家标准与技术研究院内部或跨机构报告（NISTIR）8151（草案）。美国国家标准与技术研究院，2016年。[3] Paulo Nunes、Iberia Medeiros、Jos´ e C. Fonseca、Nuno Neves、Miguel´ Correia 和 Marco Vieira。《网络安全静态分析工具的基准测试》。IEEE可靠性汇刊，第67卷，第3期（2018年）：1159-1175。[4] Al Bessey、Ken Block、Ben Chelf、Andy Chou、Bryan Fulton、Seth Hallem、Charles Henri-Gros、Asya Kamsky、Scott McPeak 和 Dawson Engler。《数十亿行代码之后：使用静态分析在现实世界中发现缺陷》。ACM通讯，第53卷，第2期（2010年）：66-75。[5] Caitlin Sadowski、Edward Aftandilian、Alex Eagle、Liam Miller-Cushon 和 Ciera Jaspan。《在谷歌构建静态分析工具的经验教训》。ACM通讯，第61卷，第4期（2018年）：58-66。[6] Bruce Potter 和 Gary McGraw。《软件安全测试》。IEEE Security Privacy，第 2 卷，第 5 期（2004年）：81-85。[7] Par Emanuelsson 和 Ulf Nilsson。《工业静态分析工具的比较研究》。Electronic Notes in Theoretical Computer Science 217（2008年）：5-21。[8] Laurens Sion、Katja Tuma、Riccardo Scandariato、Koen Yskout 和 Wouter Joosen。《迈向自动化安全设计缺陷检测》。载于《2019年IEEE/ACM自动化软件工程国际会议研讨会（ASEW）》，第49-56页。IEEE，2019年。[9] Brittany Johnson、Yoonki Song、Emerson Murphy-Hill 和 Robert Bowdidge。《软件开发人员为何不使用静态分析工具来查找缺陷？》。载于《2013年第35届国际软件工程会议（ICSE）》，第672-681页。IEEE，2013年。[10] Nasif Imtiaz、Akond Rahman、Effat Farhana 和 Laurie Williams。《回应静态分析工具警报所面临的挑战》。载于《2019年IEEE/ACM第16届软件仓库挖掘国际会议（MSR）》，第245-249页。IEEE，2019年。[11] Ferdian Thung、David Lo、Lingxiao Jiang、Foyzur Rahman 和 Premkumar T. Devanbu。《我们在多大程度上能够检测现场缺陷？静态缺陷检测工具假阴性的实证研究》。载于《2012年第27届IEEE/ACM自动化软件工程国际会议论文集》，第50-59页。IEEE，2012年。