智能合约安全分析器的不足之处在哪里?

Tamer Abdelaziz 纽约大学阿布扎比分校
tamer.m@nyu.edu

Salma Alsaghir 纽约大学阿布扎比分校
saa9125@nyu.edu

Karim Ali 纽约大学阿布扎比分校
karim.ali@nyu.edu

摘要

智能合约支撑着去中心化金融（DeFi）等高价值生态系统，但反复出现的漏洞仍持续造成数十亿美元的损失。尽管已有众多用于检测此类缺陷的安全分析器，现实中的攻击仍屡见不鲜，这不禁令人质疑：这些工具究竟是否真正有效，还是仅仅因为开发者信任度低而未被充分使用？以往的基准测试是在合成数据集或仅包含易受攻击合约的数据集上评估分析器，因此难以衡量假阳性、假阴性以及影响采用的可用性因素。

为弥合这一差距，我们开展了一项混合方法研究，将大规模基准测试与从业者见解相结合。我们在 653 个真实世界的智能合约上评估了六种广泛使用的分析器（即 Confuzzius、Dlva、Mythril、Osiris、Oyente 和 Slither）。这些合约涵盖 OWASP智能合约十大风险中的三类高影响漏洞类别（即重入、自毁合约终止和整数算术错误）。结果显示，各分析器的准确性存在显著差异（F1 = 31.2–94.6%），假阳性率较高（最高达 32.6%），且每个合约的运行时间超过 700 秒。随后，我们对 150 名专业开发者和审计人员进行了调查，以了解他们如何使用这些工具以及对这些工具的看法。研究结果表明，过多的假阳性、含糊的解释以及过长的分析时间，是实际应用中影响信任和采用的主要障碍。通过将可测量的性能差距与开发者感知联系起来，我们为改进智能合约安全分析器的精确率、可解释性和可用性提出了具体建议。

CCS 概念

• **软件及其工程 → 软件验证与确认**。

关键词

区块链安全、漏洞检测、实证评估、开发者调查、分析器局限性

ACM 参考文献格式：
Tamer Abdelaziz、Salma Alsaghir 和 Karim Ali。2026年。《智能合约安全分析器有哪些不足？》。载于第*23*届国际软件仓库挖掘会议（*MSR '26*），*2026*年*4*月*13*日至*14*日，巴西里约热内卢。ACM，纽约，纽约州，美国，13页。https://doi.org/10.1145/3793302. 3793338

本作品采用知识共享署名 4.0 国际许可协议授权。*MSR '26*，里约热内卢，巴西© 2026 版权归所有者/作者所有。ACM ISBN 979-8-4007-2474-9/2026/04
https://doi.org/10.1145/3793302.3793338

1 引言

以太坊等区块链平台 [10] 推动了智能合约的普及。智能合约是支撑去中心化应用的自执行程序。去中心化金融（DeFi）[60] 和非同质化代币（NFT）市场 [59]等重要生态系统如今依靠智能合约实现透明、可验证的交易。这种采用推动了全球区块链行业的发展，预计到 2030 [38]年其规模将达到3930亿美元。然而，智能合约仍频繁成为安全攻击的目标。仅在2024年，就发生了303起攻击，造成超过22亿美元的损失，比上一年增加了21% [12]。从大规模重入攻击到隐蔽的算术错误，反复出现的漏洞持续削弱开发者和投资者的信心 [52, 53]。

为降低这些风险，研究界与业界推出了多种安全分析器，可在部署前自动检测漏洞。这些分析器采用多种技术，包括静态分析（例如 Slither [24],Vandal [9]）、符号执行（例如 Mythril [39], Oyente [37]）、模糊测试（例如 Echidna [29], Confuzzius [57]），以及最近的基于学习的方法（例如 GPTScan [55], Dlva [3], SCooLS [2]）。尽管已有多种工具可用，现实世界中的攻击仍在持续发生。这种脱节引出了一个核心问题：这些安全分析器是否不够有效，还是开发者没有有效地使用它们（甚至根本没有使用）？

要理解这一差距，需要从两个互补的方面入手。第一方面涉及现有分析器的技术有效性。换言之，这些工具在检测现实世界智能合约中的漏洞时有多准确、多高效。尽管已有研究（例如 Ghaleb 与 Pattabiraman [26], Durieux 等人 [20]）建立了初步基准测试，但它们往往依赖仅包含漏洞样本的合成数据集。这种设计无法对误报率和漏报率做出符合实际的估计，而这些是判断开发者能否信任并主动使用这些安全分析器的关键指标。第二方面涉及开发者感知与信任。即便技术上可靠的工具，如果用户觉得它们缓慢、不精确或没有帮助，在实践中也会失败。其他软件领域的研究 [19, 33] 表明，开发者经常忽略那些他们认为不具备可操作性的分析警报。然而，对于智能合约开发者是否面临类似问题，以及哪些因素影响他们对分析器输出的信任，目前所知甚少。

为解决这两个方面的问题，我们开展了一项收敛式并行混合方法研究 [54]，将定量基准测试与从业者的定性见解相结合。在定量研究方面，我们对 653 个真实世界的智能合约进行了系统评估，这些合约包含漏洞代码和良性代码；评估对象是六种广泛使用的分析器（Confuzzius、Dlva、Mythril、Osiris、Oyente 和 Slither）。为确保比较公平，我们聚焦于 OWASP 智能合约十大漏洞中的三种高影响漏洞类型

10 [5]：重入、合约自毁和整数算术错误，所有受研究的工具都能检测这些漏洞。我们的定量基准测试表明，没有任何分析器能够可靠地检测全部三种漏洞类型。Confuzzius 在检测重入方面表现出色，而 Osiris 在检测整数算术错误方面表现最佳。此外，Slither 和 Oyente 以牺牲一定准确性为代价，换取更快的扫描速度。在所有工具中，合约自毁最难检测（F1 分数低于 60%）。

在定性研究方面，我们调查了 150 名专业智能合约开发者和审计人员，以了解他们如何看待、采用并信任这些分析器。总体而言，调查参与者对传统漏洞类型（如重入）的熟悉程度高于较新的威胁（如闪电贷攻击 [6]）。超过三分之二的参与者期望在 10 分钟内获得分析结果，因为在部署合约或寻求安全专家进一步审计之前，立即做出决定至关重要。随后，我们从调查回复中总结出若干见解，帮助我们确定哪些设计因素能够促进安全分析器更广泛地融入现实开发工作流。具体而言，我们发现了阻碍工具采用的三大信任障碍：假阳性率高（70% 的参与者）、缺乏清晰的解释（ 65% 的参与者），以及与开发者工作流的集成有限（71% 的参与者）。对调查回复的进一步分析表明，大多数参与者更喜欢包含修复建议（82%）和精确漏洞位置（78%）的简短解释。只有少数参与者表示希望获得更多细节，例如情境化解释（17%）、现实案例（17%）或可执行的修复步骤（20%）。

通过将可衡量的工具性能与开发者信任及采用行为联系起来，我们的研究增进了对安全分析器如何从技术上前景可期的原型，发展为安全智能合约开发中值得信赖的组件的理解。具体而言，本文作出以下贡献：

- 我们提出了一套均衡的基准测试，涵盖六种广泛使用的智能合约分析器，揭示了它们在真实世界智能合约上的检测准确性、计算效率和漏洞覆盖率之间的权衡。
- 我们报告了一项针对 150 名开发者和审计人员的调查结果，首次以实证方式说明从业者如何看待这些分析器、哪些因素会促进或阻碍他们的信任，以及他们认为哪些解释方式最具可操作性。
- 我们综合两方面的发现，解释为何仅有较高的检测准确性并不足以带来广泛且有信心的采用。随后，我们提出具体建议，以使分析器设计更符合开发者需求。

可用性。所有数据、基准测试和匿名化调查答复均可在我们的复制包 [1] 中获取。

2 背景

2.1 智能合约

智能合约是部署在以太坊等区块链网络上的自动执行程序 [10]。它们无需中介即可执行协议，并在满足预设条件后自动执行交易。要执行一笔交易，首先需要将其存入内存池。内存池是一个公开的暂存区（池），用于存放所有待处理、尚未确认且等待验证者选取的交易 [23]。

由于智能合约的代码和状态在部署后不可更改，合约中的任何漏洞都可能造成不可逆的财务后果，因此自动化安全分析成为一项关键保障。所以，实时检测恶意交易（即在其仍处于内存池中时）对于在交易于区块链上确认之前防止被利用至关重要。

2.2 智能合约漏洞

尽管经过多年的研究与审计实践，智能合约中的漏洞仍然既普遍又代价高昂。Chainalysis [12] 报告称，仅在2024年，各大区块链上的安全攻击造成的损失就超过22亿美元。独立审计持续表明，超过三分之一已部署的合约至少存在一项已知弱点 [20]。这些反复出现的失败凸显了漏洞检测工具与开发者实践之间的差距。

尽管已编目了数十种智能合约漏洞类型，但并非所有类型都同样具有破坏性，或都能被自动化分析器检测出来。OWASP 智能合约十大漏洞 10 [5] 识别了影响最大的类型。然而，许多工具宣称仅覆盖其中的一部分。为确保各分析器之间的比较公平且可复现，本研究聚焦于三种影响大且得到广泛支持的类型：重入、合约自毁和整数算术错误。我们的选择基于四项实用标准：(i) 历史影响，因为每一类都曾造成有记录的数百万美元损失；(ii) 分析器覆盖范围，这使得能够在相同漏洞类型上对所有被评估工具进行有意义的比较；(iii)标注可靠性，因为程序化判定条件和链上证据使我们能够构建可复现的基准真值；(iv)代表性错误类别，确保我们的集合涵盖控制流（重入）、生命周期与访问控制（自毁合约）以及算术错误（整数溢出/下溢）。我们排除了闪电贷和预言机操纵攻击等新兴威胁，因为开发者对这些类别的熟悉程度仍然较低（见第4节），且当前大多数分析器几乎不提供或完全不提供检测支持。Chaliasos 等人 [13] 得出了类似结论，表明在127起有记录的高影响去中心化金融攻击中，有75%超出了被评估工具的检测能力。纳入此类漏洞将导致接近于零的检出率，从而无法进行有意义的比较。

2.2.1 重入。当合约在更新内部状态之前就向外部地址转账时，便会产生重入漏洞，使恶意被调用方能够递归地重新进入存在漏洞的函数并耗尽资金。2016年 DAO 事件利用了这一缺陷，盗取了 360 万 ETH（≈ 当时价值 6000 万美元），为追回这些资金，以太坊不得不进行一次备受争议的硬分叉 [52]。尽管现代框架（例如 OpenZeppelin 的`ReentrancyGuard`）有助于防范此类漏洞，但重入仍是去中心化金融（DeFi）平台中最常遭利用的弱点之一。

2.2.2 合约自毁。这种问题也称为自毁或自毁漏洞，指合约中能够调用`selfdestruct()` 的函数缺乏足够的访问控制，例如

所有权检查或函数修饰符 [22]。这会使未授权行为者能够销毁合约。一个典型案例是 2017 年 Parity 钱包攻击事件：攻击者销毁了一个关键的库合约，导致约 500,000 ETH 永久被冻结（按当时价值约为 1.5 亿美元）。尽管以太坊改进提案（EIP）6049 [48]已在较新的 Solidity 版本中弃用 `selfdestruct()` 操作码，但类似的终止开关逻辑在可升级代理合约中仍很常见，因此这依然是一个持续存在的安全问题。

我们将合约自毁视为访问控制失效中一种具体且边界明确的情况。它具有简单、程序化的检测判定条件（不受限制的 `selfdestruct()`），我们的研究中约有三分之二的分析器能够检测到这一问题。相比之下，一般的访问控制漏洞（例如复杂的授权逻辑或角色配置错误）高度依赖上下文，要想一致地进行标注，就需要手动规范，这会损害可复现性。

2.2.3 整数算术错误。 当运算超出数值边界时，就会发生算术溢出和下溢。这些问题可能导致意外数值 [49], ，使攻击者能够创建未经授权的代币或绕过余额检查。事实上，此类漏洞曾导致代币铸造和余额操纵攻击，包括 2023年 Poolz Finance 攻击 [53]。尽管 Solidity 0.8.0 引入了自动溢出检查，但仍依赖未检查算术运算的旧版合约和库依然存在漏洞。因此，安全分析器必须能够准确识别不同编译器版本中的算术问题，并提醒开发者使用安全算术运算（例如使用 OpenZeppelin 的 `SafeMath` [43]）。为确保评估涵盖此类可检测实例，我们将基准测试中与算术相关的部分限定为使用 0.8 之前的编译器编译的合约。

3 安全分析器的实证评估

为了评估当前分析器检测和报告智能合约漏洞的有效性，我们针对第 2 节介绍的三类漏洞进行了定量基准测试，重点关注影响开发者信任和工具采用的权衡。

3.1 数据集收集

现有的易受攻击智能合约公开数据集中存在重复项、标签不一致以及过度合成的样本 [17]。为实现公平比较，我们整理了 ScBench，这是一个包含真实世界合约及经验证真实标签的新数据集，可用于衡量三类目标漏洞中的假阳性和假阴性。

3.1.1 数据来源。 我们从学术数据集 [3, 14, 27, 31]以及 Trail of Bits [41], 、ConsenSys Diligence [18], 和 OpenZeppelin [42] 等主要公司的安全审计报告中收集了 717 份智能合约。我们采用这些来源中人工验证的漏洞标签作为真实标签。

我们的筛选流程对每个合约提出两项关键要求。首先，合约必须至少包含我们研究的三种漏洞类型之一。其次，必须能够公开获取经过验证的源代码，以便进行真实情况验证。对于每个选定的合约，我们都会获取其经过验证的源代码以及其元数据（包括编译器版本和部署区块），来源为 Etherscan [21]；并从 Google BigQuery [28]获取运行时字节码。源代码是否可用是首要筛选条件，因为一些学术数据集仅提供合约地址和标签，却不附带源代码或字节码。例如，Chen 等人 [14] 的数据集包含 756 个地址，但我们成功获取到经过验证的源代码的仅有其中 309 个。因此，我们的数据收集流程会自动排除未经验证的源代码对应的合约。

表 1：ScBench 中智能合约的分布。

漏洞	Safe	易受攻击	总计
重入	90	95	185
自杀攻击	89	220	309
整数溢出/下溢	96	63	159
总计	275	378	653

3.1.2 数据清理。 我们会自动移除非 ASCII 字符，使用为每个合约记录的元数据修正缺失或不一致的 `pragma` 声明。此外，我们会从复合 JSON 文件中分离出各个子合约，以确保能够成功编译。完成清理后，我们得到 653 个合约，它们均可在所有研究的分析器中正确编译和运行。

3.1.3 标注与验证。 遵循"存在漏洞并不意味着漏洞已被利用"这一原则， [46], 我们的标注会区分理论上的漏洞和已利用漏洞。为此，我们首先手动检查每个合约的源代码，以识别漏洞模式。对于重入，我们会检查函数是否在更新状态变量之前，使用 `call()` 方法向外部合约转移资金。这是因为此类模式通常会导致递归调用，而递归调用是重入攻击中的常见模式。随后，我们会审查合约的交易历史，以检测是否存在利用行为的链上证据。对于自杀攻击，我们会检查是否存在未经授权的 `selfdestruct()`调用，并审查交易日志，以寻找合约销毁的链上证据。对于算术错误，我们会识别那些可能超出变量类型取值范围、且未使用安全库（例如 SafeMath [43]）的算术运算。

这一过程生成了 ScBench，这是一个包含 275 个安全合约和 378 个易受攻击合约的数据集。表 1 展示了它们的分布，其中每个合约至少包含一种所研究的漏洞类型。ScBench 反映了真实的合约状态和漏洞模式，从而能够精确测量假阳性和假阴性。

3.2 安全分析器选择

我们对安全分析器的选择建立在 Ivanov 等人 [32] 与 Zhu 等人 [64], 分别于2023年和2024年发表的最新且最广泛的综述所汇编的全面、经同行评审的目录之上，而非基于流行度的选择。这两项综述合计列举了133篇和178篇不同的论文与工具，提供了方法多样且以实证为基础的根基。基于这一广泛清单，我们采用以下纳入标准：

• 公开可用性：该分析器必须可公开获取，并能通过命令行界面（CLI）运行，以支持大规模实验。

• 版本兼容性：该工具必须支持 Solidity 编译器 0.4.x–0.8.x 版本，以确保覆盖常见的已部署合约。

• 多类别覆盖：该分析器必须能够检测 ScBench 中三种漏洞类别里的至少两种。
• 实际采用：该工具应展现出在研究或审计实践中的活跃使用情况，可通过引用、社区参与或代码仓库活动（例如 GitHub > 1000 星标和 > 300 复刻）体现。

为确保不同分析范式之间的方法一致性和评估结果可复现，我们排除以下类别的工具：

• 专有或基于订阅的分析器（例如 Zeus [34]），其闭源特性妨碍了对结果进行透明评估和复现。• 针对过时 Solidity 版本的工具（例如 Sailfish [8]），与当前编译器不兼容，并会引入操作偏差。• 需要用户自定义规范的工具（例如 Echidna [29]），其性能高度依赖人工输入，因而有损客观、自动化的比较。• 仅检测单一漏洞类型的分析器（例如 eThor [51], SaferSC [56]），无法针对我们选定的漏洞类别进行全面评估。

我们还排除了基于大语言模型的分析技术，因为它们通常不支持本研究所需的自动化、可复现评估。目前大多数基于大语言模型的工具都是针对特定数据集定制的研究原型，不适合整合到标准化基准测试流程中。例如，我们尝试评估 BlockScan [62], ，但其实现与自身的交易数据集紧密耦合，无法适配我们的基准测试。我们曾联系作者寻求指导，但未收到回复。同样，由于缺少可重复使用且公开可用的工具，我们无法获得可运行的 BlockGPT [25] 版本。这些实际障碍妨碍了开展一致、公平比较所必需的条件。我们并未一概排除基于机器学习的工具；例如，我们纳入了 Dlva [3]，因为它符合我们的标准，但排除了其他缺乏广泛漏洞覆盖率的工具（例如 SaferSC [56]）。

应用我们的纳入和排除标准后，得到以下结果：
• Confuzzius [57]：一款面向以太坊智能合约的混合模糊测试工具，将数据依赖分析与传统模糊测试相结合。它利用污点跟踪和轻量级符号执行，引导输入生成触及可能存在漏洞的路径，从而提升对重入和算术漏洞等复杂问题的检测能力。Confuzzius 通过专注于依赖感知输入选择，在模糊测试的可扩展性与更深入的漏洞覆盖之间取得平衡。

• Dlva [3]：一种基于深度学习的分析器，它在以太坊虚拟机（EVM）字节码上运行，并使用图神经网络（GNN）提取结构特征。由于Dlva在EVM字节码上运行，因此无需访问合约的原始Solidity源代码。然而，Dlva 是在由Slither [24], 标注的合约上训练的，从而继承了这些标签中的偏差。

表2：重入的实证结果。

Tool	P	R	F1	FNR	FPR	时间（秒）
Confuzzius	**94.8%**	**94.6%**	**94.6%**	2.1%	8.9%	565.12
Dlva	30.2%	43.8%	31.2%	97.9%	12.2%	2.51
Mythril	64.6%	55.7%	49.0%	80.0%	**6.7%**	723.66
Osiris	86.5%	84.3%	84.0%	3.2%	28.9%	117.25
Oyente	87.3%	85.4%	85.2%	3.2%	26.7%	7.91
Slither	91.9%	90.8%	90.7%	**1.1%**	17.8%	**1.33**

• Mythril (v0.24.7) [39]：一款广泛采用的符号执行引擎，用于分析 EVM 字节码。Mythril 通过符号交易参数系统地探索执行路径，构建控制流图，并使用 Z3 SMT求解器 [16] 识别重入、自毁攻击和算术漏洞等问题。它会生成具体的利用轨迹，并在漏洞报告中提供可采取行动的结果。

• Osiris [58]：一款基于符号执行的工具，可检测整数相关漏洞（例如溢出和下溢）。Osiris 通过有针对性的 SMT查询检查算术特定不变量，即使在复杂控制流（例如循环密集合约）中也能精准检测。

• Oyente [37]：最早用于以太坊智能合约的符号执行框架之一。Oyente 可检测重入、交易顺序依赖和未检查的调用返回值等问题。它使用路径可行性检查来减少假阳性，并可生成具体测试输入。Oyente 的模块化设计为后续符号执行工具奠定了基础，至今仍对学术研究和安全审计具有重要影响。• Slither (v0.10.4) [24]：一款针对 Solidity 源代码运行的快速静态分析器。Slither 会构建自己的中间表示（IR）SlitherIR，随后应用 70 多项漏洞和质量检查（例如重入、整数漏洞以及访问控制执行不当）。其轻量级性能使其能够在 2–3 秒内分析合约，因此适用于大规模检测和集成到持续集成（CI）流水线中。

3.3 实验设置与指标

为了收集分析结果，我们在 ScBench 上运行每种工具，每份合约的超时时间为 15 分钟。每次运行时，我们都会记录报告的漏洞类型和运行时间。随后，我们将输出与真值标签进行比较，以评估其精确率（P）、召回率（R）、F1 分数（F1）、假阴性率（FNR）、假阳性率（FPR）以及每份合约的平均分析时间。这些指标有助于了解每种工具的准确性、识别漏洞的能力和计算效率。

3.4 结果

3.4.1 重入。表2显示，Confuzzius 凭借其混合模糊测试

表3：自杀攻击的实证结果。

Tool	P	R	F1	FNR	FPR	时间（秒）
Confuzzius	**78.1%**	51.1%	50.3%	66.8%	**4.5%**	481.01
Dlva	57.0%	40.1%	40.4%	70.9%	32.6%	2.20
Mythril	77.5%	49.2%	47.8%	69.5%	**4.5%**	491.71
Slither	70.0%	**57.6%**	**59.4%**	**49.1%**	25.8%	**1.71**

表4：整数算术错误的实证结果。

Tool	P	R	F1	FNR	FPR	时间（秒）
Confuzzius	47.5%	55.3%	47.9%	**12.7%**	14.6%	304.38
Mythril	77.8%	64.8%	54.7%	88.9%	**0.0%**	664.07
Osiris	**84.1%**	**83.0%**	**83.2%**	**12.7%**	19.8%	240.04
Oyente	75.7%	75.5%	75.6%	28.6%	21.9%	**17.61**

和污点跟踪设计，实现了最高的总体准确率（F = 26.6%）。Slither 位居其次（F1 = 90.7%），结果几乎即时得出（每个合约耗时 1.33 秒），但假阳性率（FPR）较高（17.8%）。符号执行工具（Oyente 和 Osiris）的精确率中等（分别为 F1 = 85.2% 和 84%），但比 Slither 更容易过度标记（假阳性率分别为 F = 26.7% 和 28.9%）。Mythril 的表现最差（F1 = 49%，假阴性率 F = 80%），将大多数易受攻击的合约错误分类。Dlva 也表现出类似的弱点，精确率平平（F1 = 31.2%），且由于其保守的分类阈值降低了假阳性率（12.2%），代价是召回率下降，几乎漏掉了所有真正例（假阴性率 F = 97.9%）。

3.4.2 合约自毁。表3显示，所有工具在检测这种漏洞类型时都表现不佳。Slither 表现最佳（F1 = 59.4%），但其 25.8% 的假阳性率提高了开发者进行人工分诊的成本。Mythril 和 Confuzzius 漏掉了约三分之二的真正例，表明它们的启发式方法无法很好地建模授权逻辑。Dlva 的精确率中等（P = 57.0%），召回率较低（R = 40.1%），漏掉了很大一部分真正例（假阴性率 = 70.9% ），同时还产生了大量假阳性（假阳性率 = 32.6% ）。

在分析时间方面，Slither 仍然是速度最快的工具，平均每份合约耗时 1.71 秒；而 Confuzzius 和 Mythril 则需要 >480 秒，凸显了它们较高的计算开销。Dlva 的速度比 Confuzzius 和 Mythril 快几个数量级，仅次于 Slither。不过，Dlva 是以明显牺牲检测质量为代价实现这一点的。

3.4.3 整数算术错误。表4显示，专门用于算术不变量检测的 Osiris 表现最佳，F1 值为 83.2%，假阴性率较低，为 12.7%；其次是 Oyente（F1 = 75.6%）。Confuzzius 和 Mythril 则远远落后（分别为 F1 = 54.7% 和 47.9%），凸显了它们在数值推理方面支持有限。

没有任何单一分析器能在所有漏洞类型上都占据优势。Confuzzius 擅长检测重入，而 Osiris 在检测整数算术错误方面表现最佳。Slither 和 Oyente 以牺牲准确性为代价换取更快的扫描速度。Dlva 将覆盖范围扩大到没有源代码的合约，但表现仍落后于具备源码感知能力的分析器。合约自毁终止是最难检测的问题（F1 <60%）。

3.5 讨论

ScBench 清晰地展现了准确率–效率前沿：精准检测往往伴随着高得令人难以接受的运行时间，而速度更快的工具则会对良性合约过度标记。这些发现揭示了开发者即使有工具可用，仍会犹豫是否将其整合到工作流中的原因。为应对这些挑战，我们根据研究结果提出以下针对性建议：

(1) 整合互补方法，以扩大覆盖范围。 Confuzzius 在重入检测方面的检出率最高，这体现了将模糊测试与符号执行相结合的有效性。将这种混合分析方法扩展到符号执行工具（例如 Mythril），或可通过改进复杂合约中的路径探索来减少假阴性。(2) 优化符号执行。 Mythril 的假阴性率较高，且由于采用穷举式符号执行，运行速度最慢。采用有界探索 [11], 可行性检查（类似 Oyente），或进行 SMT 查询优化 [15] 等技术，或可同时提升效率和漏洞覆盖率。(3) 扩充专门的漏洞模式。 得益于专门设计的算术谓词，Osiris 在检测整数算术错误方面表现最佳。将类似的定制漏洞模式纳入其他工具（例如 Slither 和 Confuzzius），或可提升它们检测整数算术错误和自杀攻击的能力。(4) 通过交叉验证减少假阳性。 Osiris 和 Oyente 的假阳性率较高，削弱了人们对其输出结果的信任。将其他工具（例如 Slither）的静态分析结果作为交叉验证步骤，有助于过滤不可信的漏洞报告，同时保持可接受的分析时间。(5) 投入资源提升可扩展性。 与其他工具相比，Confuzzius 和 Mythril 的分析时间显著更长。并行处理符号执行和模糊测试任务，并纳入静态预分析以优先检查高风险路径，有助于克服可扩展性有限的问题。还可采用有界符号执行 [11],可行性检查（类似 Oyente）或 SMT 查询优化 [15] 等技术，同时提升效率和漏洞覆盖率。提升可扩展性是让深入分析能够在开发者预期的运行时间预算（≤ 10 分钟）内切实可行的关键。(6) 开发基于集成的分析器。没有任何单一工具能在所有漏洞类型和性能指标上持续优于其他工具。开发集成框架，将多个分析器的输出结合起来（例如采用博弈论方式 [30]），能够发挥各自优势、扩大覆盖范围，并降低个别工具弱点的影响。(7) 开发并发布基于大语言模型的检测器。 研究人员应优先公开发布符合可复现性标准的基于大语言模型的漏洞检测器，并提供可用于基准测试的制品和评估脚本，以便进行公平、自动化的比较。(8) 将基准测试扩展至新兴威胁。 由于分析器支持有限且开发者对相关问题不够熟悉，我们的评估未涵盖闪电贷漏洞和预言机操纵漏洞。随着工具日趋成熟，未来的基准测试应纳入针对这些利用类型的精选数据集，以评估工具对现代攻击向量的检测能力。

4 智能合约开发者调查

第 3 节中的基准测试结果揭示，现有分析器在准确性、效率和可靠性方面存在显著差异。为了了解这些技术局限如何影响开发者信任和采用，我们在定量评估之外，还对专业智能合约开发者和审计人员开展了调查。本节报告该调查的定性和定量发现；这项调查构成了我们收敛式并行混合方法设计的第二条研究线索 [54]。

4.1 调查设计

4.1.1 问卷。 调查包含25个问题，用于收集人口统计和背景信息、工具使用模式、对准确性和可信度的看法，以及对解释形式的偏好。问题形式包括选择题、李克特量表题和开放式问题，旨在兼顾定量测量与定性深度。表5列出了所有调查问题。完整问卷和匿名化回复可公开获取 [1]。

4.1.2 招募。 我们通过 Prolific [47],进行预筛选，招募参与者，要求其参与过智能合约开发、审计或区块链安全工作。我们的纳入标准如下：

- 主要语言为英语，
- Prolific 认可率在 95–100% 之间，
- 目前的工作涉及编程语言，并且
- 在数据分析、软件工程或信息安全领域担任组织职务。

在 Prolific 上进行为期三周的数据收集期间，我们共收到 347 份已完成的调查答卷。按照该平台既定的质量控制实践 [44], ，我们从中筛选出 150 份高质量答卷，答卷者均在智能合约分析方面拥有真实经验。

为验证参与者的专业能力，我们首先使用 Prolific 的预筛选条件，招募自认在职业上从事智能合约开发、审计或区块链安全工作的人员。此外，我们还在调查中设置了能力检验问题。具体而言，Q6 测试参与者识别常见及较少见漏洞类型的能力，Q7 则考察其使用特定分析工具的实际经验。对于知识前后矛盾的答卷，例如声称精通相关领域却无法识别基本漏洞，或表现出类似机器人的模式，我们会予以剔除。这种方法评估的是实际应用能力，而非仅仅依赖参与者自我报告的经验。后续定性编码表现出的高评审者间信度（科恩系数 κ = 0.755–0.866）进一步证实，保留的参与者提供了连贯且达到专家水平的反馈。每位通过验证的参与者均获得 5英镑作为合理报酬，以补偿其投入的时间；这一金额符合 Prolific 上的平均报酬标准。

4.1.3 伦理考量。我们的研究已获纽约大学阿布扎比分校机构审查委员会（IRB）批准（研究编号：HRPP-2025-3）。本研究被认为对参与者的风险极低。在回答问题之前，我们会征得每位参与者的知情同意。我们会告知参与者本调查的目的，且 Prolific 允许他们随时退出调查。我们

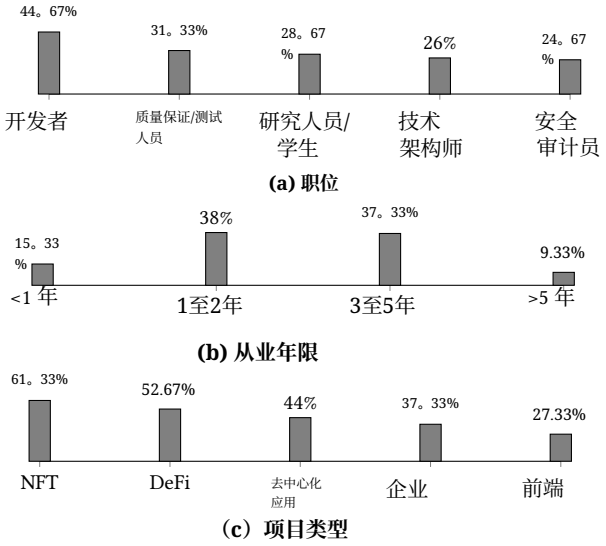


图1：调查参与者的背景。

确保收集的数据经过匿名化处理。我们还对调查进行了试测，以确保参与者能在合理的时间内完成（25–30 分钟）。

4.2 研究问题

促使我们开展这项工作的主要研究问题是：现有安全分析器是否不够有效，或者用户是否没有有效地使用它们（甚至完全没有使用）？我们将这一主要研究问题分解为若干子问题：

- **研究问题1**：用户最常认可哪些安全漏洞?
- **研究问题2**：最常使用哪些安全分析器?
- **研究问题3**：安全分析器可接受的运行时性能如何?
- **研究问题4**：哪些因素会影响用户对安全分析器的信任/信心?
- **研究问题5**：用户为什么会忽略安全分析器报告的漏洞?
- **研究问题6**：用户认为安全分析器标记的漏洞应如何解释，才能最有效地增强信任并促进修复?

4.3 研究发现

为分析定量调查答复，我们采用描述性统计。为获得定性见解，两位作者采用改良德尔菲卡片分类法 [45]，对开放式答复进行系统分类。

4.3.1 参与者背景。 图1显示，参与者涵盖多种职位：开发者（44.7%）、测试人员（31.3%）、研究人员/学生（28.7%）、技术架构师（26%）和审计人员（24.7%）。大多数参与者（75.33%）拥有1–5年从业年限，并且

表5：按章节分组的调查问题。

章节	ID	问题
参与者背景	Q1	您在智能合约开发流程中担任什么职位?
	Q2	您从事智能合约开发多少年了?
	Q3	您主要使用哪些编程语言进行智能合约开发?
	Q4	您主要参与哪种类型的智能合约项目?
	Q5	安全与保障在您的智能合约开发流程中有多重要?
	Q6	以下哪些智能合约安全漏洞是您熟悉的?
安全分析器的使用分析器	Q7	您在智能合约开发过程中使用过哪些安全分析器?
	Q8	开发智能合约时，您使用安全分析器的频率如何?
	Q9	您通常在开发的哪些阶段使用安全分析器?
	Q10	您使用安全分析器的主要原因是什么?
	Q11	您更喜欢使用哪种界面来操作安全分析器?
	Q12	您通常会使用安全分析器分析哪种输入?
	Q13	您更喜欢安全分析器采用哪种定价模式?
	Q14	您通常最多会允许安全分析器运行多长时间，之后才会期待获得结果?
	Q15	平均而言，您会花多少时间核实报告的漏洞是否为真正例?
对安全分析器输出	Q16	您对安全分析器报告的漏洞的准确性有多大信心?
	Q17	以下哪些因素会增强您对安全分析器结果的信心?
	Q18	以下哪些因素会降低你对安全分析器结果的信心?
	Q19	你是否曾忽略过安全分析器报告的漏洞?
	Q20	你能否简要说明一种你选择忽略报告的漏洞的情况?
影响解释对信心	Q21	安全分析器报告漏洞时，您认为哪类解释最有帮助?
	Q22	解释的质量在多大程度上会影响您对分析器结果的信心?
	Q23	如果安全分析器报告了漏洞，并提供了详细解释，其中包含代码片段或建议的修复方案，您有多大信心认为该漏洞是真正例?
	Q24	如果安全分析器报告了漏洞，但没有提供解释，您有多大信心认为它是真正例?
	Q25	对于安全分析器使用的解释格式，您有什么改进建议吗?

大多数人从事非同质化代币（61.33%）和去中心化金融（52.67%）项目。最后，大多数参与者（96%）认为智能合约安全很重要。

4.3.2 常见漏洞（研究问题***1***）。图2显示，参与者最熟悉重入（62.7%）和整数算术错误（54.7%）等经典漏洞，而对闪电贷攻击或预言机操纵攻击等新兴威胁的认知仍低于40%。这种认知差异表明，教育和工具文档仍然着重强调已知漏洞，导致新的漏洞利用类别未得到充分关注。

研究问题1：用户最熟悉重入等经典漏洞，而对闪电贷和预言机操纵等新兴威胁的认知较少，这凸显了人们对新兴攻击类型的认识有限。

4.3.3 使用模式（研究问题***2***）。由于大多数参与者重视智能合约安全，超过半数（54.66%，Q8）的参与者在超过 75% 的工作中使用安全分析器。使用场景（Q9）涵盖审计期间（78%）、编码（75.33%）和测试（74%）。主要原因（Q10）包括识别漏洞（89.33%）、确保代码质量（82%）、了解潜在安全问题（68%），以及遵守组织或监管政策（53.33%）。参与者对工具界面（Q11）没有明确偏好。34% 的参与者偏好命令行界面，28% 偏好集成开发环境（IDE）插件，21.33% 则更倾向于使用基于网页的界面。关于工具成本，大多数参与者倾向于使用免费增值定价模式（52%，Q13）。

图3显示，Slither 是最常用的分析器（77.33%），其次是 Mythril（51.3%）。它们的受欢迎程度反映了实际考量。Slither 的持续维护、快速分析和广泛的漏洞覆盖率（如第3节所示），使其成为许多开发者的默认选择。尽管运行时间较长，Mythril 凭借其符号执行能力仍颇具吸引力。相比之下，需要用户定义大量输入（例如 Echidna）或不再积极维护（例如 SmartCheck）的分析器，采用率明显较低。

尽管文献中包含许多基于机器学习（ML）的漏洞检测器 [64]，，但很少有作者发布可用于生产的工具或数据集，而且我们的调查参与者中没有人报告

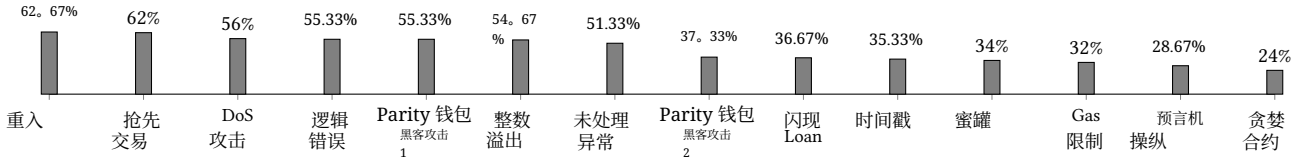


图2：对智能合约安全漏洞的熟悉程度。

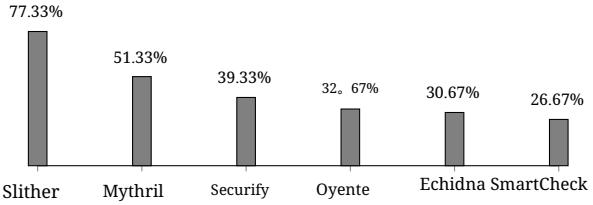


图 3：最常用的安全分析器。

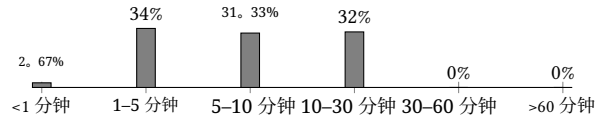
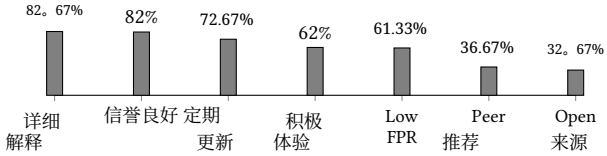
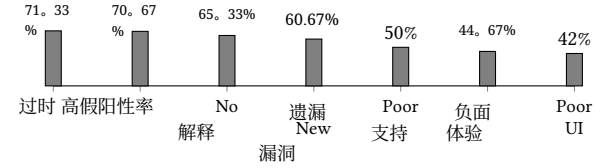


图4：安全分析器可接受的运行时间。



（a）提高置信度的因素



（b）降低信心的因素

图5：影响对安全分析器信心的因素。

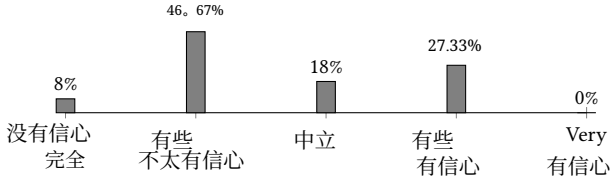
使用基于机器学习的工具。这种缺失凸显了研究与实践之间的差距，其原因可能是工具可用性有限、集成与部署方面的挑战，以及对机器学习驱动的发现的可解释性和信任方面的顾虑。

> **研究问题2**：Slither 和 Mythril 是使用最广泛的分析器，这反映了开发者更倾向于选择积极维护、运行速度快且易于集成到其工作流中的工具。

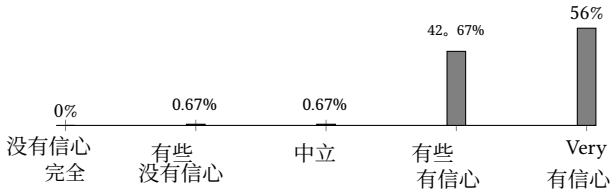
4.3.4 可接受的运行时间性能（研究问题***3***）。图4显示，超过三分之二的参与者希望在10分钟内获得分析器结果（34%偏好1–5分钟，31%偏好5–10分钟），无人能接受超过30分钟的运行时间。这些期望与我们在第 3 节中观察到的实际运行时间限制相一致，凸显了在准确性与及时反馈之间取得平衡的重要性，以便将其集成到开发工作流中。

> **研究问题3**：68%的参与者希望在10分钟内获得分析器结果，这与我们在实证评估中观察到的时间限制相一致。

4.3.5 影响对分析器信心的因素（研究问题***4***）。大多数参与者（73%）表示对分析器输出持部分信任（问题16），并且通常会手动复核严重问题。图5展示了参与者认为有助于增强其对工具结果信心的主要因素：详细解释（82.67%）和分析器声誉（82%）。相反，过时工具（71.33%）、高假阳性率（70.67%）以及缺少解释（65.33%）会削弱信心。对58.67%的参与者而言，解释质量会影响其信心



(a) 没有解释



(b) 附带解释

图 6：对安全分析器的信心。

参与者。这些发现印证了我们在第 3 节中指出的技术缺陷。

图 6 进一步显示，当分析器结果附有解释或支持性证据时，用户的信任度会大幅提高：98.67% 的参与者表示，警报附有解释时，他们会更有信心；一半的参与者表示，按五分制量表计算，信心至少提升了两个等级。

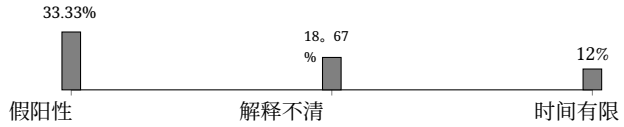


图 7：忽略分析器警告的主要原因。

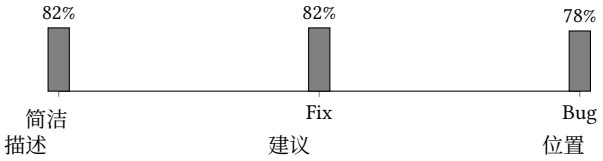


图8：偏好的解释格式。

研究问题4：影响分析器置信度的主要信任障碍包括假阳性率过高（71%）、工具支持过时（71%）以及缺乏解释（65%）。绝大多数用户（98%）表示，当工具对结果提供清晰、详细的解释时，他们会更有信心。

4.3.6 忽略已报告漏洞的原因（研究问题***5***）。为探究用户为何忽视智能合约安全分析器标记的漏洞，我们在调查中设置了封闭式问题（Q19）和开放式问题（Q20）。约三分之一的参与者（34.67%）从不忽略警告。然而，图7显示，用户忽略警告主要是因为假阳性过多（33.33%）或解释不清（18.67%）。另有较少比例的受访者（12%）将时间有限列为忽略警告的原因。

为深入了解这一现象，我们对Q20的开放式回答进行了定性分析，对常见主题进行编码，以识别普遍的理由。这些回答的编码者间信度较高（Cohen's $\kappa = 0.866$），超过了表示几乎完全一致的0.81阈值 [36],，确保了定性分析的稳健性。在Q20的回答中，最常见的主题是"手动验证"（14.67%）和"假阳性"（14%）。在包含多种主题的回答中，排名前两位的主题同样是"假阳性"（29.17%）和"手动验证"（27.78%）。由于这两个主题表明用户不信任工具的准确性，我们的定性分析进一步证实，导致用户忽略已报告漏洞的是信任度下降，而非漠不关心。

研究问题5：用户忽略安全警告，主要是因为假阳性过多（33.3%）和解释不清（18.7%）。虽然三分之一的参与者从不忽略警报，但对大多数人而言，忽略警报是应对工具不准确以及验证分析结果所需工作量过大的务实做法。

4.3.7 最有用的解释类型（研究问题***6***）。为了解用户对漏洞解释格式的偏好，我们分析了调查问题 Q21 和 Q25 的回答。图8显示，参与者对简洁的漏洞描述和修复建议的偏好程度相同（均为 82%），其次是对精确漏洞位置的偏好（78%）。

为确定具体偏好，我们进一步对 Q25 的回答进行了定性分析。对这些回答进行编码时，评分者间信度相对较高（Cohen's $\kappa = 0.755$），接近通常表明独立标注者之间具有实质性一致性的 0.81 阈值 [36]。回答中单一的偏好并不常见，包括"结构化解释"（8%）、"情境化解释"（6.67%）、"可执行的修复步骤"（6%）和"可视化辅助"（3.33%）等特征。大多数回答（58.67%）体现了混合理由，表明用户希望结合多种解释特征。在这些混合回答中，最常见的是"可执行的修复步骤"（20.15%）和"情境化解释"（17.16%），其次是"真实世界示例和文档"（17.16%）、"可视化辅助"（13.81%）和"结构化解释"（11.94%）。这些发现表明，用户重视可执行且清晰的解释；这类解释不仅简洁，还应实用，并能直接用于修复已识别的漏洞。

研究问题6：开发者优先考虑可操作且高效的漏洞解释。大多数用户偏好简短描述和修复建议（82%），以及精确的漏洞位置（78%）。需要更多细节时，参与者通常更偏好可操作的修复步骤（20.2%）、情境化解释（17.2%）和现实案例（17.2%）。

4.4 对工具设计的启示

将用户反馈与基准测试结果相结合，可以得出若干针对智能合约安全分析器设计的可行建议。这些建议直接反映了用户的需求和期望，有助于打造更有效、更可信且更广泛采用的工具。

(1) 兼顾准确性与反馈速度。超过三分之二的参与者希望在10分钟内获得分析结果，没有人能接受工具耗时30分钟或更久才生成结果。为融入迭代开发工作流且不影响生产效率，分析器应力争在10分钟内完成。(2) 清晰优先于冗长。所报告解释的质量会影响超过半数参与者对分析器结果的信心。此外，超过80%的参与者偏好简短描述和可操作的修复建议。因此，分析器应着重提供简洁、可验证的解释，以及针对所标记漏洞的具体修复步骤。(3) 提高精确率并加强过滤。参与者忽略警告的主要原因是高假阳性率（33.33%）。由于高假阳性率会直接导致分类审查工作量浪费并降低信任，安全分析器必须着力生成精确的结果。(4) 支持灵活的界面。参与者偏好以命令行工具、集成开发环境插件或Web仪表板形式提供的分析器，这反映了工作流的多样性。为尽可能提高采用率，分析器应支持灵活的部署选项，以适应这些不同的工作流。

(5) 通过基于模式的解释，在简洁性与上下文深度之间取得平衡。 我们的研究结果表明，开发者更偏好简洁的描述（82%），但也会要求详细的上下文，例如

可执行的修复步骤（20.2%）。这些偏好并非相互矛盾，而是反映了不同的工作流程阶段。在分诊阶段，开发者需要包含漏洞位置的简短单行描述，以便快速做出继续或放弃的决定。在修复阶段，当某条警告被判定为可信时，他们需要可展开的详细修复建议。理想的分析器应同时支持两种模式：一种摘要模式，提供简要描述以便浏览大量警告；以及一种 详情 模式，为已确认的问题提供代码片段和修复模式，类似于带有可选 `-explain` 标志的编译器错误消息。

这些从业者见解与定量基准测试相结合，揭示了可衡量的工具缺陷在实践中如何表现为可用性和信任障碍。两者共同为设计新一代分析器提供了基于证据的依据，使开发者既愿意也能够使用这些工具。

5 效度威胁

5.1 内部效度

一个主要的威胁来自我们精选数据集中漏洞真实标签的准确性。这些标签中的不准确之处可能会使精确率和召回率等评估指标产生偏差，从而可能扭曲我们对工具性能的评估。为缓解这一问题，我们进行了人工代码分析，以识别漏洞模式；审查了交易历史记录，以查找利用的证据；并将标签与安全审计和学术文献中报告的事件进行交叉验证 [3, 14, 27, 31]，同时确保数据集包含已遭利用和未遭利用的合约，遵循以下原则：漏洞并不意味着遭到利用 [46]。

另一项威胁是为工具评估设置的15分钟超时，这可能会截断较慢工具（例如 Mythril 和 Confuzzius）的分析，从而影响性能指标。不过，该时限比开发者期望收到分析结果的10分钟窗口高出50%。因此，我们认为这在完整性与实际约束之间取得了合理平衡。

最后，我们调查中的自我报告数据可能受到社会期许偏差的影响，从而可能夸大所报告的专业水平或工具使用情况。为解决这一问题，我们通过 Prolific [47] 确保了匿名性，并面向经验丰富的从业者，以鼓励坦诚作答。

5.2 外部效度

我们的以太坊合约数据集可能无法完全代表其他区块链（例如币安智能链 [7]）上的合约，从而可能限制我们结果的适用性。为应对这一威胁，我们确保数据集包含多样的漏洞类型以及反映真实场景的合约状态。

同样，我们的调查聚焦于以太坊开发者，这可能会限制研究结果在其他区块链生态系统中的适用性。为缓解这一威胁，我们招募了职位和经验水平各不相同、具有广泛代表性的从业者，涵盖了以太坊社区的多个方面。本研究聚焦于以太坊，因为它拥有规模最大、最成熟的智能合约生态系统，提供了大量现实世界事件、多样化的已部署字节码以及成熟完善的分析工具。这些条件是进行全面、可复现评估的必要基础。我们的研究发现可直接适用于兼容 EVM 的区块链（例如，币安智能链 [7], Polygon [35]），因为它们采用相同的字节码、编译器工具链，且大多数漏洞类别相同。将研究扩展到非 EVM 平台（例如 Solana [61]），则需要针对不同的编译器、语言语义、漏洞分类体系和特定区块链的标注规则重新构建流程；这将留待未来工作完成。

此外，我们对所评估的六种工具得出的发现可能无法推广到其他工具或未来版本。为应对这一威胁，我们选择了广泛使用、持续维护且采用多样分析技术的工具。

5.3 构念效度

一个关键威胁是，从业者解读之间对漏洞的定义可能存在差异，导致我们的指标与现实世界的认知不一致。为缓解这一威胁，我们采用了既有文献中的标准化定义 [4, 63]。此外，我们还聚焦于有充分文档记录且经安全审计验证的问题。

另一个值得关注的问题是，通过自我报告的信心和定性回答衡量的用户信任，可能无法充分反映实际行为。为解决这一问题，我们将定量量表与定性开放式问题相结合。

虽然精确率、召回率和 F1 分数等指标可用于评估技术性能，但可能无法充分反映可用性或现实世界中的有效性。为解决这一问题，我们纳入了多项指标，包括假阴性率、假阳性率和分析时间。随后，我们结合有关信任和可用性的调查见解，以提供全面评估。

6 相关工作

6.1 智能合约漏洞

关于智能合约漏洞的研究为理解其性质和影响奠定了基础。Atzei 等人 [4]提出了具有开创性的常见漏洞分类体系，其中包括重入、整数溢出/下溢和未经检查的外部调用，这些漏洞至今仍威胁着合约完整性。Zhou 等人 [63]在此基础上将其应用于智能合约安全，识别出 20 多种漏洞类型，并强调了自动化检测机制的必要性。现实世界事件（例如 2016 [52] 中的 DAO 黑客攻击，以及 2017 年冻结了 1.5 亿美元的自杀攻击 [50]）展现了这些缺陷可能造成的严重后果。本研究利用了一个包含 653 个真实世界合约的数据集，这些合约存在重入、自毁和整数溢出/下溢漏洞，反映了这些已有充分记录的问题，从而对安全分析器进行全面评估。

6.2 智能合约安全分析器的实证评估

多项大规模实证评估为工具性能确立了基准。Durieux 等人 [20] 对 47,587 个真实世界未标注合约和 69 个已标注的脆弱合约进行了广泛评估，并建立了可复现的 SmartBugs 框架。尽管这项工作揭示了各工具之间的检测重叠和不一致之处，但评估仅限于正样本（即易受攻击的合约），这不可避免地

排除了数量均衡的非易受攻击（即负样本）合约，因此限制了对假阳性率（FPR）和假阴性率（FNR）等工具性能指标进行全面评估的能力。

为通过系统化测试解决工具的局限性，Ghaleb 和 Pattabiraman [26] 提出了 SolidiFI，这是一种变异测试框架。该方法向 50 个合约中注入 9,369 个有针对性的漏洞，以衡量六种静态分析器的假阴性和假阳性。尽管结果揭示了工具的若干局限性，但该研究仅使用合成注入漏洞，可能无法涵盖现实世界漏洞和开发者代码的全部复杂性与多样性。

其他研究关注被标记的漏洞与实际利用之间的差异。Perez 和 Livshits [46] 利用 Datalog 查询，分析了 23,327 个存在漏洞的合约及 2,000 万条以太坊交易轨迹，发现被标记的合约中实际遭到利用的仅占 1.98%。这项研究为利用率为何偏低（例如资金集中）提供了重要见解。然而，该研究依赖汇总的既有漏洞数据，既未重新运行工具，也未纳入负样本，因此无法计算假阳性率和假阴性率等标准性能指标。

最近，Chaliasos 等人 [13] 评估了五款知名分析工具在 127 起有记录、造成重大影响的去中心化金融攻击中的表现，这些攻击造成的损失总额超过 23 亿美元；他们还对 49 位从业者开展了调查。作者发现，这些工具只能预防其中 8% 的攻击，且主要针对基于重入的问题，这凸显了行业在检测逻辑漏洞方面存在显著不足。然而，他们的评估聚焦于高度专业化的攻击（例如与闪电贷相关的攻击），其中 75% 超出了其基准测试工具目前的适用范围。此外，报告中的评估并未使用全面且均衡的数据集来评估假阳性、假阴性或效率指标。

与以往工作不同，我们的研究基于包含 653 个经人工标注的真实世界合约的定制数据集，对六款广泛使用的工具进行了全面的性能与效率基准测试。我们针对三种关键且可能造成重大经济损失的漏洞类型——重入、自毁和整数溢出/下溢——精确测量了精确率、召回率、假阳性率、假阴性率和效率。测量结果显示，假阳性率较高（例如，Slither 在重入检测中的假阳性率为 17.8%，Oyente 为 26.7%），给开发者带来了沉重且难以持续的验证负担。我们对 150 位从业者开展的大规模调查也直接印证了这一发现：33.33% 的受访者表示，假阳性是他们忽视安全警告的原因之一。我们的研究有意聚焦于三类漏洞，并采用定制标注的数据集，从而能够严谨、一致且综合地分析工具性能及由此产生、以人为中心的挑战。

6.3 智能合约安全中的用户信任与可用性

尽管安全工具的技术能力至关重要，但用户信任和可用性也是关键的人为因素，会直接影响这些工具在开发工作流中的实际采用和有效性。Chaliasos 等人 [13] 通过调查 49 位从业者，探讨了开发者对工具的整体使用体验以及工作流方面的挑战。然而，该研究并未关注定量性能指标（例如假阳性率、假阴性率、分析时间）如何影响用户对已报告漏洞的信任。尽管已有许多关于智能合约工具和技术的其他综述与编目工作 [32], , 但它们很少涉及以人为中心的采用或信任挑战。

在更广泛的软件工程领域，Noller 等人 [40]调查了 100 多名从业者，以确定哪些因素能够提升人们对通用编程语言自动生成补丁的信任，重点关注交互偏好和支持性证据。这些发现能否迁移到智能合约这一高风险、特定领域的环境中，仍有待探索；在该环境下，遗漏漏洞可能造成灾难性后果。我们的工作通过专门研究开发者对智能合约安全工具的信任，填补了这一空白。我们不再局限于一般的工具使用情况，而是量化了高误报率、低检出率、缺乏清晰的漏洞解释以及工具集成不佳如何削弱开发者的信心，并经常导致警告疲劳以及对关键警报的忽视。这种将技术性能数据与关于开发者信任的深入定性见解相结合的综合方法，是我们工作的独特之处。

7 结论

本文对智能合约安全分析器进行了大规模、以从业者为基础的评估，结合了定量基准测试与开发者的定性见解。我们对 653 个经过人工验证的合约（ScBench）进行评估后发现，分析器在准确性、效率和可用性方面存在显著差异。F1 分数介于 31.2% 和 94.6% 之间，假阳性率最高达到 32.6%，部分分析时间超过每个合约 700 秒。没有任何单一工具能够在所有漏洞类别中占据优势。例如，Confuzzius 在重入检测方面表现出色，而 Slither 的假阴性率最低，但两者在合约自毁和算术错误方面的表现都不佳。

我们对 150 名专业开发者开展的调查，通过揭示低采用率背后的人为因素，为这些实证发现提供了补充。用户指出，假阳性过多、解释不清以及反馈缓慢，是他们难以信任工具的主要障碍。综合来看，这些发现表明，基准测试中测得的技术局限，在实际使用中直接体现为可用性和信心方面的问题。我们的评估聚焦于三类成熟的漏洞，这些领域现有工具的性能可以量化。我们承认，由于分析器支持有限且开发者对此不够熟悉，闪电贷和预言机操纵攻击等新兴威胁仍未得到充分研究。随着检测能力日趋成熟，将这些现代攻击向量纳入基准测试，是未来研究的重要方向。

根据这些结果，我们明确了未来研究和工具开发的重点。短期内，分析器应着力减少假阳性、提升解释质量，并优化运行时间，以适应开发者的工作流。长期方向包括整合混合分析技术，并扩大对新兴攻击向量的检测覆盖范围，例如闪电贷攻击和预言机操纵。我们还呼吁社区协作，共同建立共享基准测试和开放数据集，以确保进展可复现。

参考资料

[1] Tamer Abdelaziz、Salma Alsaghir 和 Karim Ali。2025年。《“安全分析器的实证评估”和“智能合约开发者调查”的复制包》。https://github.com/blockchain-security-artifacts/sc-developer- study-1。[2] Tamer Abdelaziz 和 Aquinas Hobor。2023年。《训练以利用愚蠢合约》。载于2023年第五届区块链计算与应用国际会议（BCCA）。IEEE，388–395。[3] Tamer Abdelaziz 和 Aquinas Hobor。2023年。《智能学习以发现愚蠢合约》。载于第32届USENIX安全研讨会（USENIX Security 23）。1775–1792。[4] Nicola Atzei、Massimo Bartoletti 和 Tiziana Cimoli。2017年。《以太坊智能合约攻击综述（SOK）》。载于《安全与信任原理：第六届国际会议（POST 2017）论文集》，该会议作为欧洲软件理论与实践联合会议（ETAPS 2017）的一部分于瑞典乌普萨拉举行，时间为2017年4月22日至29日。施普林格，164–186。[5] Jinson Varghese Behanan 和 Shashank。2025年。《OWASP 智能合约十大风险》。网页。https://owasp.org/www-project-smart-contract-top-10/ 访问日期：2025年10月20日。[6] Jinson Varghese Behanan 和 Shashank。2025年。《SC07:2025——闪电贷攻击》。OWASP 智能合约十大风险。https://owasp.org/www-project-smart-contract-top-10/2025/en/src/SC07-flash-loan-attacks.html 访问日期：2025年10月20日。[7] 币安。2020年。《币安智能链：币安链的并行区块链》。https://docs.bnbchain.org/。访问日期：2025年7月18日。[8] Priyanka Bose、Dipanjan Das、Yanju Chen、Yu Feng、Christopher Kruegel 和 Giovanni Vigna。2022年。《Sailfish：数秒内审查智能合约状态不一致错误》。载于2022年IEEE安全与隐私研讨会（SP）。IEEE，161–178。[9] Lexi Brent、Anton Jurisevic、Michael Kong、Eric Liu、Francois Gauthier、Vincent Gramoli、Ralph Holz 和 Bernhard Scholz。2018年。《Vandal：面向智能合约的可扩展安全分析框架》。arXiv预印本 arXiv:1809.03981（2018）。[10] Vitalik Buterin 等。2013年。《以太坊白皮书》。GitHub代码库1，22–23（2013），5–7。[11] Cristian Cadar、Daniel Dunbar、Dawson R Engler 等。2008年。《Klee：无需辅助且自动生成复杂系统程序的高覆盖率测试》。载于OSDI，第8卷。209–224。[12] Chainalysis。2025年。《2025年加密货币犯罪报告》。https://go.chainalysis.com/2025-Crypto-Crime-Report.html。访问日期：2025年3月1日。[13] Stefanos Chaliasos、Marcos Antonios Charalambous、Liyi Zhou、Rafaila Galanopoulou、Arthur Gervais、Dimitris Mitropoulos 和 Benjamin Livshits。2024年。《智能合约和DeFi安全工具：它们满足从业者的需求吗？》。载于第46届IEEE/ACM国际软件工程会议论文集。1–13。[14] Jiachi Chen、Xin Xia、David Lo 和 John Grundy。2021年。《智能合约为何会自毁？对以太坊SELFDESTRUCT函数的调查》。ACM软件工程与方法学汇刊（TOSEM）31卷2期（2021），1–37。[15] Leonardo De Moura 和 Nikolaj Bjørner。2007年。《SMT求解器的高效E匹配》。载于自动演绎国际会议。施普林格，183–198。[16] Leonardo De Moura 和 Nikolaj Bjørner。2008年。《Z3：高效的SMT求解器》。载于系统构建与分析工具和算法国际会议。施普林格，337–340。[17] Monika Di Angelo 和 Gernot Salzer。2023年。《整合用于智能合约弱点检测的真实值数据集》。载于金融密码学与数据安全国际会议。施普林格，439–455。[18] ConsenSys Diligence。 [n. d.]。《ConsenSys Diligence——智能合约审计》。https://consensys.net/diligence/。访问日期：2025年7月18日。[19] Lisa Nguyen Quang Do、James R Wright 和 Karim Ali。2020年。《软件开发者为何使用静态分析工具？一项针对开发者需求和动机的以用户为中心的研究》。IEEE软件工程汇刊48卷3期（2020），835–847。[20] Thomas Durieux、João F Ferreira、Rui Abreu 和 Pedro Cruz。2020年。《对47,587份以太坊智能合约自动化分析工具的实证评估》。载于第42届ACM/IEEE国际软件工程会议论文集。530–541。[21] 以太坊。2025年。《Etherscan：以太坊区块链浏览器》。https://ethersca n.io/。[22] 以太坊基金会。2025年。《Solidity语言文档：函数修饰符》。Solidity语言文档。https://docs.soliditylang.org/en/latest/contracts.html#function-modifiers 访问日期：2025年10月20日。[23] Etherscan。2025年。《待处理交易内存池》。https://etherscan.io/txsPending。https://etherscan.io/txsPending 访问日期：2025年10月16日。[24] Josselin Feist、Gustavo Grieco 和 Alex Groce。2019年。《Slither：智能合约静态分析框架》。载于2019年第二届IEEE/ACM区块链软件工程新兴趋势国际研讨会（WETSEB）。IEEE，8–15。[25] Yu Gai、Liyi Zhou、Kaihua Qin、Dawn Song 和 Arthur Gervais。2023年。《区块链大语言模型》。arXiv预印本 arXiv:2304.12749（2023）。[26] Asem Ghaleb 和 Karthik Pattabiraman。2020年。《智能合约分析工具的效果如何？使用错误注入评估智能合约静态分析工具》。载于第29届ACM SIGSOFT软件国际研讨会论文集。测试与分析。415–427。[27] GitHub。2022。SB Curated。https://github.com/smartbugs/smartbugs-curated 访问日期：2025年7月16日。[28] 谷歌。2025。BigQuery bigquery-public-data.crypto_ethereu m.contracts。https: //console.cloud.google.com/bigquery。[29] Gustavo Grieco、Will Song、Artur Cygan、Josselin Feist 和 Alex Groce。2020。Echidna：面向智能合约的高效、易用且快速的模糊测试。载于《第29届ACM SIGSOFT软件国际研讨会软件测试与分析论文集》。557–560。[30] Jiaqi He、Revan MacQueen、Natalie Bombardieri、Karim Ali、James R. Wright 和 Cristina Cifuentes。2023。寻找检测软件漏洞的最优静态分析器组合。载于《软件维护与演化国际会议》（ICSME）。IEEE，463–473。[31] Tianyuan Hu、Jingyue Li、Bixin Li 和 Andre Storhaug。2024。为何被报告为存在漏洞的智能合约并未遭到利用？《IEEE可靠与安全计算汇刊》（2024）。[32] Nikolay Ivanov、Chenning Li、Qiben Yan、Zhiyuan Sun、Zhichao Cao 和 Xiapu Luo。2023。智能合约的安全威胁缓解：全面综述。《计算机综述》55，14s（2023），1–37。[33] Brittany Johnson、Yoonki Song、Emerson Murphy-Hill 和 Robert Bowdidge。2013。软件开发者为何不使用静态分析工具查找漏洞？载于《2013年第35届国际软件工程会议》（ICSE）。IEEE，672–681。[34] Sukrit Kalra、Seep Goel、Mohan Dhawan 和 Subodh Sharma。2018。Zeus：分析智能合约的安全性。载于《NDSS》。1–12。[35] Jaynti Kanani、Sandeep Nailwal 和 Anurag Arjun。2021。Matic白皮书。Polygon，印度班加罗尔，技术报告（2021）。[36] J. R. Landis 和 G. G. Koch。1977。分类数据观察者一致性的测量。载于《生物统计学》。159–174。[37] Loi Luu、Duc-Hiep Chu、Hrishi Olickel、Prateek Saxena 和 Aquinas Hobor。2016。让智能合约变得更智能。载于《2016年ACM SIGSAC计算机与通信安全会议论文集》。254–269。[38] MarketsandMarkets。2025。按组件、供应商、类型、组织规模、应用和地区划分的区块链市场——2030年全球预测。https: //www.marketsandmarkets.com/Market-Reports/blockchain-technology-market-90100890.html。访问日期：2025年10月15日。[39] Bernhard Mueller。2018。以太坊智能合约的趣味攻击与真实获利。HITB SECCONF 阿姆斯特丹 9（2018），54。[40] Yannic Noller、Ridwan Shariffdeen、Xiang Gao 和 Abhik Roychoudhury。2022。程序修复中的信任增强问题。载于《第44届国际软件工程会议论文集》。2228–2240。[41] Trail of Bits。 [无日期。]。Trail of Bits——安全审计与研究。https://www.tr ailofbits.com/。访问日期：2025年7月18日。[42] OpenZeppelin。 [无日期。]。OpenZeppelin——审计与开源安全。https: //openzeppelin.com/security-audits/。访问日期：2025年7月18日。[43] OpenZeppelin。2025。OpenZeppelin Contracts。https://github.com/OpenZeppeli n/openzeppelin-contracts。[44] Stefan Palan 和 Christian Schitter。2018。Prolific.ac——在线实验的受试者库。《行为与实验金融杂志》17（2018），22–27。[45] Celeste Lyn Paul。2008。针对新型卡片分类方法的改良德尔菲法。《可用性研究杂志》4，1（2008），7–30。[46] Daniel Perez 和 Benjamin Livshits。2021。智能合约漏洞：存在漏洞不代表遭到利用。载于《第30届USENIX安全研讨会》（USENIX Security 21）。1325–1341。[47] Prolific。2025。Prolific：研究参与者招募。https://www.prolif ic.com/。访问日期：2025年2月1日。[48] 以太坊改进提案。2022。EIP-6049：弃用SELFDESTRUCT。https://eips.ethereum.org/EIPS/eip-6049 访问日期：2025年10月22日。[49] SWC注册表。2020。整数溢出与下溢。https://swcregistry.io/docs/SWC-101/ 访问日期：2025年7月16日。[50] SWC注册表。2020。自杀攻击。https://swcregistry.io/docs/SWC-106/ 访问日期：2025年7月16日。[51] Clara Schneidewind、Ilya Grishchenko、Markus Scherer 和 Matteo Maffei。2020。eThor：对以太坊智能合约进行实用且可证明可靠的静态分析。载于《2020年ACM SIGSAC计算机与通信安全会议论文集》。621–640。[52] David Siegel。2016。了解DAO攻击。https://www.coindesk.com/learn/understanding-the-dao-attack。[53] SolidityScan。2023。Poolz Finance攻击分析：溢出问题是否仍在持续？SolidityScan博客。https://blog.solidityscan.com/poolz-finance-hack-analysis-still-experiencing-overflow-fcf35ab8a6c5 访问日期：2025年10月20日。[54] Margaret-Anne D. Storey、Rashina Hoda、Alessandra Maciel Paz Milani 和 Maria Teresa Baldassarre。2025。软件工程混合方法研究的指导原则。《实证软件工程》（EMSE）30，5（2025），138。[55] Yuqiang Sun、Daoyuan Wu、Yue Xue、Han Liu、Haijun Wang、Zhengzi Xu、Xiaofei Xie 和 Yang Liu。2024。GPTScan：结合GPT与程序分析检测智能合约中的逻辑漏洞。载于《IEEE/ACM第46届国际软件工程会议论文集》。1–13。

[56] Wesley Joon-Wie Tann、Xing Jie Han、Sourav Sen Gupta 和 Yew-Soon Ong。2018。迈向更安全的智能合约：一种用于检测安全威胁的序列学习方法。*arXiv*预印本 *arXiv:1811.06632*（2018）。[57] Christof Ferreira Torres、Antonio Ken Iannillo、Arthur Gervais 和 Radu State。2021。Confuzzius：一种具备数据依赖感知能力的智能合约混合模糊测试器。载于*2021*年*IEEE*欧洲安全与隐私研讨会（*EuroS&P*）。IEEE，103–119。[58] Christof Ferreira Torres、Julian Sch¨utte 和 Radu State。2018。Osiris：搜寻以太坊智能合约中的整数漏洞。载于第*34*届年度计算机安全应用会议论文集。664–676。[59] Qin Wang、Rujia Li、Qi Wang 和 Shiping Chen。2021。非同质化代币（NFT）：概述、评估、机遇与挑战。*arXiv*预印本 *arXiv:2105.07447*（2021）。[60] Sam Werner、Daniel Perez、Lewis Gudgeon、Ariah Klages-Mundt、Dominik Harz 和 William Knottenbelt。2022。综述：去中心化金融（DeFi）。载于论文集

第四届 *ACM* 金融技术进展大会. 30–46. [61] Anatoly Yakovenko. 2018. Solana：一种高性能区块链的新架构 v0. 8.13. (2018). [62] Jiahao Yu, Xian Wu, Hao Liu, Wenbo Guo, and Xinyu Xing. 2025. BlockScan：检测区块链交易中的异常。发表于第三十九届神经信息处理系统年会。[63] Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. 2023. Sok：去中心化金融（defi）攻击。发表于*2023 IEEE* 安全与隐私研讨会 *(SP)*。IEEE，2444–2461. [64] Huijuan Zhu, Lei Yang, Liangmin Wang, and Victor S Sheng. 2024. 智能合约安全分析方法综述。*IEEE* 服务计算汇刊 (2024)。