中国科学：信息科学

SCIENTIA SINICA Informationis

综述

区块链上智能合约安全综述

胡玉斌[1]

1. 北京邮电大学, 网络空间安全, 北京 100876
E-mail: yubin.hu@bupt.edu.cn

摘要 本文主要介绍了区块链上智能合约的常见的漏洞，并结合现实世界，探究如何检测漏洞。在此基础上，进一步考虑如何自动修复相关漏洞。相关研究发现，在现实世界中，各种区块链上的智能合约存在多种类型的漏洞，目前对相关漏洞的定义，检测方法也较为成熟，而自动修复方法的研究也在进行中。

关键词 区块链, 智能合约，漏洞

1 引言

区块链是链接在一起的公共记录列表。由于底层加密机制，区块链中的记录可以抵抗修改。随着加密货币的繁荣繁荣（例如，比特币，以太坊），区块链技术已经变得更具吸引力，并在许多领域采用。

然而，与传统代码程序一样，智能合约受基于代码的漏洞，可能导致巨额的财务损失并阻碍其应用。考虑到在网络上部署智能合约后，修改智能合约是比较困难的。换句话说，在部署之前，必须保证智能合约是健壮的。

下面本文将从区块链平台，智能合约等背景，相关漏洞的攻击面，如何检测相关漏洞以及漏洞自动修复的相关研究作出综述。

2 背景

Ethereum [1] 和 EOSIO[] [2] 是支持智能合约的最受欢迎的公共区块链平台。但是，Ethereum 和 EOSIO 智能合约中的漏洞导致其用户的巨大的财务损失。对于 EOSIO 智能合约，相关漏洞导致了大约 380K EOS 令牌 [3] 的损失。这些漏洞的累计损失金额在攻击时为 190 万美元。对于 Ethereum 智能合约，DAO 合约中的漏洞 [4] 导致 6000 万美元的损失。

2.1 Ethereum

以太坊（Ethereum）是一个开源的有智能合约功能的公共区块链平台，通过其专用加密货币以太币（Ether，简称"ETH"）提供去中心化的以太虚拟机（Ethereum Virtual Machine）来处理点对

点合约。以太坊的概念首次在 2013 至 2014 年间由程序员 Vitalik Buterin 受比特币启发后提出，大意为” 下一代加密货币与去中心化应用平台”，在 2014 年通过 ICO 众筹开始得以发展。截至 2018 年 2 月，以太币是市值第二高的加密货币，仅次于比特币。

2.2 EOSIO

作为最具代表性的 DPOS 平台之一和第一分散式操作系统之一，EOSIO 已成为最活跃的全球社区之一。EOSIO 采用基于其 DPO 共识协议的多线程机制，能够实现数百万 TPS。EOSIO 的性能优势使其成为分散应用程序（DAPPS）开发人员的流行。EOSIO 在 2018 年 6 月在推出后三个月内成功超过了 DAPP 交易的 Ethereum [5]。目前，平均 EOSIO 的交易量超过了以太坊的百倍 [6]。截至 2019 年，EOSIO 的连锁交易总值已达到 60 亿美元。

2.3 智能合约

这个术语是跨领域法律学者尼克 · 萨博（Nick Szabo）提出来的，他对智能合约的定义是” 一个智能合约是一套以数字形式定义的承诺（promises），包括合约参与方可以在上面执行这些承诺的协议。” 换成更加通俗的描述就是” 智能合约是一个在计算机系统上，当一定条件被满足的情况下，可以被自动执行的合约。”

智能合约是一段可以在区块链上执行的代码，并将合约的执行状态作为该区块链实例不可改变的历史的一部分。因此，开发者可以依靠该区块链作为一个可信的计算环境，其中智能合约的输入、执行和结果是独立的，不受外部影响。

以太坊上智能合约是用 Solidity 编写的。Solidity 是一门面向合约的、为实现智能合约而创建的高级编程语言。这门语言受到了 C++，Python 和 Javascript 语言的影响，设计的目的是能在以太坊虚拟机（EVM）上运行。Solidity 是静态类型语言，支持继承、库和复杂的用户定义类型等特性。

EOSIO 上的智能合约是用 WebAssembly 编写的。WebAssembly（缩写为 Wasm）是一种基于堆栈的虚拟机的二进制指令格式。Wasm 被设计为编程语言的可移植编译目标，能够在网络上部署客户端和服务器应用程序。WebAssembly 虚拟机可以被嵌入到网络浏览器或区块链平台。EOSIO 区块链已经支持 Wasm。此外，在以太坊 2.0 中，Wasm VM 是以太坊虚拟机（EVM）的替代品。

使用 Wasm 的设计选择使 EOSIO 能够重用经过优化和实战检验的编译器和工具链，这些编译器和工具链正在被一个更广泛的社区维护和改进。此外，采用 Wasm 标准也使编译器开发人员更容易将其他编程语言移植到 EOSIO 平台上。

WebAssembly 有两种可转换和等价的表示方法。我们有一种二进制格式——以”.wasm ” 为后缀。为了使 WebAssembly 能够被人类阅读和编辑，有一个 Wasm 二进制格式的文本表示——”.wast ” 作为后缀。

3 漏洞

下面我们将介绍在 Ethereum 和 EOSIO 平台上所披露的漏洞。

3.1 重入漏洞

2016 年 7 月，TheDao 合约中的错误允许攻击者窃取 50M。事物的原子性和顺序可能使开发人员认为在终止之前无法重新进入非递归函数。但是，对于智能合约，这种想法并不是如此。

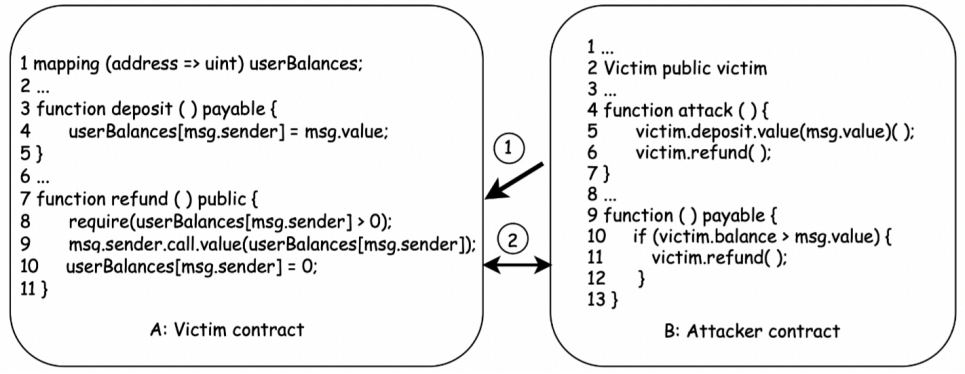

图 1 重入漏洞漏洞的利用

图1显示了重入漏洞的利用方法。首先，调用攻击者智能合约中的 *attack()* 函数向受害者合约存一些代币来调用受害者的脆弱 *refund()*) 函数。然后，*refund()*) 方法向攻击者合约（a 中的第 9 行）发送申请的项目，也触发攻击合约中未命名的 *fallback* 函数（B 中的第 9 行）。接下来，*fallback* 函数再次调用受害者合约中的 *refund()*) 函数（B 中的第 11 行）。由于受害者契约更新了 *userBalances* 变量（A）之后的 *userbalances* 变量（第 10 行）后，当攻击者重新进入 *refund()*) 函数时，*userBalances* 保持不变，因此仍然可以通过余额检查（a 中的第 8 行）。因此，攻击者能够从受害者合约中反复转账，以此来盈利。

3.2 缺少输入验证

一个函数的参数在使用前应该被验证。如果开发者忘记给参数分配正确的值，EVM 将使用基于参数类型的默认值来执行函数。这种机制使智能合约容易受到对函数参数的攻击。

3.3 锁定代币

2017 年，一个脆弱的合约导致了百万美元的冻结。原因是这个合约依赖另一个库合约来提取其资金（使用 delegatecall）。不幸的是，一个用户意外地从区块链上删除了该库合约（使用 *kill* 指令），因此钱包合约中的资金无法再被提取。

3.4 未处理的异常

在 Solidity 中，有多种情况可能会引发异常。未处理的异常会影响智能合约的安全性。2016 年 2 月，一个易受攻击的合约因为调用指令中存在未处理的异常，迫使所有者要求用户不要向所有者发送以太坊代币。

3.5 算术漏洞

以太坊虚拟机（EVM）为整数指定固定大小的数据类型。这意味着一个整型变量只能有一定范围的数字表示。例如，一个 *uint8* ，只能存储在范围 [0,255] 的数字。试图存储 256 到一个 *uint8* 将变成 0。不加注意的话，只要没有检查用户输入又执行计算，导致数字超出存储它们的数据类型允许的范围，Solidity 中的变量就可以被用来组织攻击。

整数溢出的类型包括乘法溢出，加法溢出，减法溢出三种。

3.6 假 EOS

任何人都可以创建和发行名为 EOS 的令牌，因为在 EOSIO 中，令牌的名称和符号并不要求是唯一的。

由于 *eosio.token* 的源代码是完全公开的，任何人都可以复制它的源代码并发行一个具有相同名称、符号和代码的令牌。然而，假的 EOS 和官方的 EOS 有不同的发行者。因此，如果攻击者通过复制合约的转移功能将假的 EOS 转移到赌博的 DApp，DApp 方收到的通知代码将不是 *eosio.token*。此外，如果 DApp 恰好没有检查代码的值，那么调度器中的验证将被绕过。

为了缓解上述问题，一些开发者缩小了接受代码的范围。*self* 或 *eosio.token* 都可以被当作代码的有效输入值。然而，如果攻击者直接调用转移函数，这样的缓解措施也可以被绕过。由于条件 *code == self* 总是被满足，即使没有来自 eosio.token 的通知，转移函数也会被调用，这表明有转移请求。

这两种情况都属于假 EOS 令牌漏洞 [7]。

3.7 假收据

如果 DApp 开发者对代码进行了全面的检查，那么通知就会被调度器转发。但是，如果开发者在这一步没有进行核查，DApp 就会被攻击。需要强调的是，通知可以被转发，而代码不会改变。因此，DApp 可能被同时扮演发起人和共犯双重角色（账户）的攻击者所欺骗。具体来说，发起人通过 *eosio.token* 向共犯（用 *to* 表示，转移函数的参数）调用常规转移。当帮凶收到 *eosio.token* 的通知时，它将立即把通知转发给 DApp，而不作任何修改。通过这种方式，代码没有改变，这仍然是官方的发行者：*eosio.token*。因此，调度器将不知道任何异常情况。然而，如果在转移中不检查参数 *to*，DApp 将被愚弄，因为代币转移在攻击者控制的两个账户之间完成。这可能会导致 DApp 开发者的直接经济损失。

3.8 回滚漏洞

在 *reveal* 函数中，DApp 处理与玩家的转移一起收到的赌注。在转移函数中，开发者通常使用各种链上状态值作为种子（例如，*current_time*，表示执行动作时的时间戳）来生成一个伪随机数 3，最后通过 *modulo operation* 将生成的数字与玩家的输入进行比较来获得结果，这是由 Wasm 字节码中的 *rem* 操作符实现的。注意，一般来说，回滚的情况只能在赌博的 DApps 中找到。我们假设它总是在那里，并且可以从调度器中到达。即使开发者对输入的每个参数都做了彻底的检查，并在任何敏感动作之前检查调用者的权限，一个符合图 3 模型的游戏仍然可能被攻击。具体来说，所有的

动作都是内联调用的，也就是说，在一个交易中定位。因此，当玩家在第 8 步后收到通知时，他可以立即调用另一个内联动作到 *eosio.token* 来检查他的余额。如果他的余额减少，那么就意味着他没有赢得这一轮。他可以使用一个断言语句来迫使当前的动作失败。一个动作的失败可能会导致整个交易的逆转。通过这种方式，玩家可以继续尝试，直到他中了大奖。我们把这种恶意的回滚称为回滚漏洞。

4 漏洞检测

以上我们列出来在 Ethereum 和 EOSIO 平台上的常见漏洞，下面我们将介绍如何检测以上漏洞。

4.1 通过比较历史版本检测漏洞

对于以太坊来说，在区块链系统上摧毁智能合约的唯一方法是在攻击者发现漏洞时使用 *Selfdestruct* 函数。与以太坊不同，部署在基于 EOSIO 的区块链上的应用程序是可以更新的。这意味着只要提供足够的权限，你就可以部署代码修复的补丁，增加功能，并改变应用程序的逻辑。作为一个开发者，你可以迭代你的应用程序，而不会有被永久锁定在一个软件错误中的风险。我们可以很容易地收集同一智能合约的几个版本。一般来说，开发者更新的代码可能是漏洞、新功能，甚至是人为植入的欺诈代码。

4.2 符号执行

EOSafe 将符号执行引擎设计为泛型框架，以模拟基于堆栈的 EOS VM 上的智能合同的执行。它接受 CFG 和拆卸的 WASM 指令作为输入，并在基本块中进行符号执行，以便以获取所有可行的路径。在此过程中，相应地生成路径约束。具体地，模块维护两个关键组件：路径树和状态。路径树由路径组成，可以进行当前智能合同的可控制流程。当遇到某些条件指令（如 *br_table*）时，路径将分为两条路径。沿着每条路径，我们不仅记录相应的约束（即控制流的输入值的条件导致该结果），还有调用的导入功能的所有签名，这有助于分析漏洞检测。至于该状态，我们维护了一些必要的状态相关信息，包括本地/全局变量，线性存储器，堆栈和随后的指令，以及其对应的程序计数器。

4.3 模糊测试

一些研究者设计了一个用于自动分析智能合同的模糊工具 ETHPLOIT [8]。其中包括五个步骤：

1. 静态分析。鉴于智能合同的稳定性代码，Ethploit 编译了提取应用程序二进制接口（ABI）和字节码的代码。它还适用 Taint 分析来提取变量之间的依赖性。

2. 测试案例。Ethploit 最初生成测试用例。Ethploit 而不是提供任何预定义的代码模式，而是进行模糊以优化测试用例。基于前一轮模糊测试产生的静态分析结果和反馈结果，进一步更新测试案例。

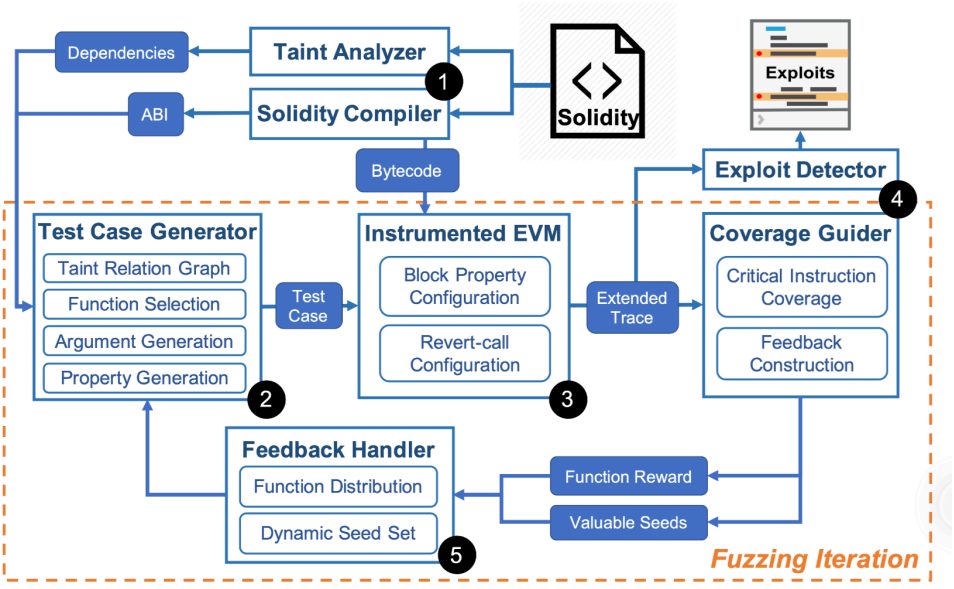

图 2 ETHPLOIT 的工作流

3. 测试案例执行。Ethploit Instruments 是模拟区块链效应的 EVM。它还在仪表 EVM 和输出执行跟踪上应用每个测试用例。由于指令，迹线涵盖了更多的执行概率。

4. 追踪分析。Ethploit 分析每个执行跟踪。如果 Exproit 检测器标识漏洞利用，则 Ethploit 将当前的测试用例报告为有效的漏洞利用。同时，指令的覆盖范围作为反馈构建优化测试用例。ETHPLOIT 将反馈区分为两类：修改功能分布和用于更新动态种子集的有价值种子的功能奖励。

5. 反馈处理。Ethploit 将功能奖励视为测试案例发生器的功能分布，以选择更有可能通过过去模糊迭代的经验利用漏洞的功能。

反馈处理程序还将有价值的种子添加到种子集中，该种子集可用于生成能够解决一些严格约束的参数。

4.4 根据现有工具进行投票制筛选

在漏洞修复框架 EVMPATCH [9] 中，在能够应用修补程序之前，EVMPATCH 框架需要识别和检测漏洞。为此，EVMPATCH 框架利用现有的漏洞检测工具，例如 [[10], [11], [12], [13], [14], [15], [16]]。对于任何现有工具未检测到的漏洞，要求开发人员或安全顾问创建漏洞报告。在 EVMPATCH 的系统中，漏洞检测组件负责标识漏洞所在的指令的确切地址，以及漏洞的类型。然后将该信息传递给字节码重写器，相应地修补合同。

5 漏洞自动修复

目前调研有大概这样的漏洞自动修复方法。

- 使用基于模板的修复模式生成补丁 [17]，并利用静态程序分析。
- 二进制重写 [9]。二进制重写也被应用于改造安全加固技术，如控制流的完整性，以编译的二进制文件，但也可以动态地将安全补丁应用于运行的程序。对于传统架构上的二进制重写，已经开发了两种类型的方法：静态和动态重写。

不过漏洞自动修复还不算成熟，泛用性不够。目前市面上的区块链公司仍采用专业人员代码审计。

6 评估

大体上，相关工作的评估方面有以下几种：

- 假阳性
- 是否会引入新的漏洞
- 运行时性能
- 额外 Gas 开销

7 总结

区块链上的智能合约仍存在着大量漏洞，漏洞的检测和修复手段也在发展中，相关漏洞带来的经济损失惨重，未来这个方面仍是重要的科研方向。

参考文献

1 Ethereum[EB/OL].https://ethereum.org/en/.

2 EOSIO Developer Portal[EB/OL].https://developers.eos.io/.

3 EOSRoyale Smart Contract. https://www.eosroyale.com, Last access, 2020.

4 Analysis of the DAO exploit. http://hackingdistributed.com/2016/06/18/analysis-of-the-daoexploit/. Last access, 2020.

5 CRAIG RUSSO, "EOSIO surpasses Ethereum in transaction volume," Sep. 2018. [Online]. Available: https://sludgefeed.com/eos-surpasses-ethereum-in -daily-dapp-users-and-transaction-volume/

6 Alfredo de Candia, "Increase of EOSIO transaction volumes," Sep. 2019. [Online]. Available: https://en.cryptonomist.ch/2019/09/03/eos-porn-transaction-volumes

7 He, N. , Zhang, R. , Wu, L. , Wang, H. , Luo, X. , & Guo, Y. , et al. (2020). Security analysis of eosio smart contracts.

8 Zhang, Q. , Wang, Y. , Li, J. , & Ma, S. . (2020). EthPloit: From Fuzzing to Efficient Exploit Generation against Smart Contracts. 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE.

9 Rodler, M. , Li, W. , Karame, G. O. , & Davi, L. . (2020). Evmpatch: timely and automated patching of ethereum smart contracts.

10 Christof Ferreira-Torres, Julian Schütte, and Radu State. "Osiris: Hunting for Integer Bugs in Ethereum Smart Contracts". In: Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC).2018. DOI: 10.1145/3274694.3274737.

11 Shelly Grossman, Ittai Abraham, Guy Golan-Gueta, Yan Michalevsky, Noam Rinetzky, Mooly Sagiv, and Yoni Zohar. "Online detection of effectively callback free objects with applications to smart contracts".In: Proceedings of the ACM on Programming Languages POPL (2018).DOI: 10.1145/3158136.

12 Johannes KruppandChristian Rossow. "teEther: Gnawing atEthereum to Automatically Exploit Smart Contracts". In: 27th USENIX Security Symposium. USENIX Association, 2018. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/ krupp.

13 Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. "Making smart contracts smarter". In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS). ACM. 2016.

14 Ivica Nikolic, Aashish Kolluri, Ilya Sergey, Prateek Saxena, and Aquinas Hobor. "Finding The Greedy, Prodigal, and Suicidal Contracts at Scale". In: 34th Annual Computer Security Applications Conference (ACSAC). 2018. DOI: 10.1145/3274694.3274743.

15 Michael Rodler, Wenting Li, Ghassan O. Karame, and Lucas Davi."Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks". In: Proceedings of the Network and Distributed System Security Symposium (NDSS). 2019. DOI: 10.14722/ndss.2019.23413.

16 Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. "Securify: Practical security analysis of smart contracts". In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018.

17 Nguyen, T. D. , Pham, L. H. , & Sun, J. . (2021). Sguard: towards fixing vulnerable smart contracts automatically.