基于区块链的智能合约漏洞分析与风险防范

文◆山东经贸职业学院　白　翔

引言

区块链智能合约是一种基于区块链技术的可执行计算代码[1]，区块链智能合约具有透明性、不可篡改性、可验证性、自动执行性、去中心化等特点，被广泛应用于各个领域，如金融、供应链管理、数字资产交易等，为合约执行带来了更高的效率和安全性[2]。然而，智能合约同样存在诸多漏洞，且漏洞一旦暴露，会引发资产盗窃、系统瘫痪等灾难性后果。因此，应结合智能合约的漏洞类型，分析智能合约漏洞产生的原因，并建立健全智能合约漏洞风险防范机制。

1 基于区块链的智能合约漏洞类型

智能合约漏洞包括多种类型，付梦琳等人总结了 10 种出现频率最高的智能合约漏洞类型[3]，倪远东等人提出智能合约安全三层威胁模型并介绍了 15 类主要漏洞[4]。综合以上研究成果来看，智能合约漏洞主要包括以下 8 种。

（1）重入攻击。重入攻击的核心逻辑是利用外部调用的控制权转移特性，通过递归调用突破合约的状态一致性约束[5]。在 EVM（Ethereum Virtual Machine，以太坊虚拟机）中，调用外部合约时，控制权会暂时转移至被调用合约。若被调用合约的回退函数中再次调用原合约的敏感函数，则会形成调用循环。此时，原合约的状态变量尚未更新，攻击者可重复触发转账操作，导致超额资产转移。重入攻击的本质是状态更新滞后于外部调用的时序错误。例如，在提现逻辑中，若先执行转账操作，再扣减用户余额，则攻击者可在转账回调中再次触发提现，利用未更新的余额重复转账。

（2）整数溢出与下溢。Solidity 的无符号整数类型采用模运算规则，当运算结果超出类型取值范围时，会自动回绕而非抛出异常。若代码未对关键算术运算进行范围校验，则溢出 / 下溢会导致状态变量的异常变化。

（3）时间戳依赖。区块链的区块时间戳由矿工节点提议并共识确定，存在 15s 的可操控空间。若合约将时间戳作为随机数源、奖励触发条件或关键业务逻辑的判断依据，矿工可通过调整出块时间影响结果，导致不公平或可预测的行为。

（4）权限控制缺陷。权限控制是合约对升级、铸币、资金划转等敏感操作的访问限制机制。常见缺陷包括未限制访问权限、权限转移逻辑漏洞、过度授权，如关键函数未使用 `onlyOwner` 等修饰符，导致任意地址可调用。未验证新管理员地址的有效性或未设置二次确认机制，导致权限被恶意转移。

（5）Gas 限制攻击。EVM 通过 Gas 机制限制单次交易的计算量，若合约设计未考虑 Gas 消耗上限，可能导致循环耗尽 Gas、存储操作 Gas 优化不足等风险。以循环耗尽 Gas 为例，在遍历数组或映射的循环操作中，若集合规模过大，单次交易的 Gas 消耗可能超过区块 Gas 上限。

（6）业务逻辑漏洞。业务逻辑漏洞源于对实际需求的错误抽象，常见于奖励计算、条件判断、状态转移等场景。其核心特征是代码逻辑与业务目标不一致，具体表现为条件判断不完整以及竞态条件。

【作者简介】白翔（1983—），女，山东济宁人，本科，讲师，研究方向：区块链在管理会计上应用及数据分析。

（7）预言机操控。预言机是智能合约获取外部数据的桥梁。若预言机设计存在单点依赖或验证缺失，攻击者可操控数据输入，导致合约错误执行。以数据源攻击为例，若预言机仅依赖单一数据，攻击者可通过“闪电贷 + 砸盘 / 拉盘”临时操控价格，触发合约的错误清算或奖励。

（8）函数可见性错误。Solidity 函数的可见性决定其调用范围。常见错误包括默认可见性、错误使用 external 函数、internal 函数暴露等。以错误使用 external 函数为例，external 函数只能被外部调用，若在合约内部调用则会浪费 Gas，且会因重入风险被利用。

2 基于区块链的智能合约漏洞成因

基于区块链的智能合约漏洞是多方面因素综合作用的结果，涉及技术特性、开发流程、外部环境等。

（1）智能合约语言特性限制。Solidity 作为主流智能合约语言，其内在的设计缺陷为智能合约埋下状态可变性与外部调用冲突、类型安全缺失、继承与覆盖风险等隐患。从状态可变性与外部调用冲突的角度而言，合约状态与外部调用的执行顺序敏感，若状态更新滞后于外部转账，如重入攻击，会导致资产重复转移。从类型安全缺失的角度而言，早期版本无自动溢出检查，需手动引入 SafeMath 库，且 address 与 uint 类型可隐式转换，易引发数值操作错误。从继承与风险覆盖的角度而言，合约继承会导致函数覆盖冲突，如子合约未正确 override 父合约函数或 delegatecall 调用恶意库合约时篡改状态。

（2）开发者安全意识与经验不足。多数开发者，仅注重转账、铸币等业务逻辑的实现，对输入校验、异常处理等边界条件缺乏足够的重视。重功能轻安全的开发理念增加了智能合约的漏洞风险，如 BEC 代币漏洞中，开发者未考虑 cnt * _value 的溢出风险。部分开发者，对区块链底层机制理解不足，不熟悉以太坊虚拟机 Gas 消耗、交易顺序、revert 处理等运行机制，导致出现逻辑漏洞。例如，未意识到 call 函数的异步性会触发重入攻击。此外，部分开发者过于依赖未审计的第三方代码，将其直接复用于开源合约，未审查其安全缺陷，导致漏洞传染。

（3）测试与验证体系不完善。测试与验证体系不完善也是智能合约安全漏洞高发的重要因素。首先，测试覆盖不全。传统单元测试仅验证正常流程，忽视恶意输入、并发调用等异常场景。例如，The DAO 的测试未模拟递归调用场景，导致重入漏洞未被发现。其次，形式化验证门槛高。形式化验证需将合约逻辑转换为 Coq、F* 语言等数学模型，对开发者数学和编程能力要求极高，实际应用率较低。最后，动态测试效果不佳。模糊测试虽能生成随机输入，但难以覆盖所有可能的数值组合，如 uint256 的极大值。此外，符号执行受路径爆炸问题限制，复杂合约的分析时间可达数小时。

（4）区块链底层机制的固有风险。除开发、测试、监控层面的原因，区块链底层机制的固有风险，如交易顺序可操纵性、不可篡改与升级矛盾也是智能合约漏洞的重要因素。一方面，矿工可通过调整交易打包顺序影响合约执行结果。另一方面，合约部署后无法直接修改代码，漏洞修复需通过复杂的迁移或代理模式，如 TransparentProxy，但升级逻辑本身会引入新漏洞。

3 基于区块链的智能合约漏洞风险防范

针对漏洞成因，需构建覆盖开发、测试、运行全生命周期的安全防护体系，重点做好编码规范、工具链支持以及运行时监控。

（1）遵循安全编码规范。遵循安全编码规范是开发阶段智能合约漏洞风险防范的关键。首先，采用标准化安全库。优先复用 OpenZeppelin、ConsenSys Diligence 等经过审计的开源库。OpenZeppelin 涵盖 Ownable、Reentrancy

```
function transfer(address to, uint256 value) public {
    require(to != address(0), "Invalid address"); // 校验地址非零
    require(value > 0, "Value must be positive"); // 校验数值有效
    require(balances[msg.sender] >= value, "Insufficient balance");
    //...
```

```
function withdraw() public {
    uint256 amount = balances[msg.sender];
    require(amount > 0);
    balances[msg.sender] = 0; // 先更新状态
    (bool success, ) = msg.sender.call{value: amount}(""); // 后转账
    require(success, "Transfer failed");
}
```

图 1 防御性编程图示

Guard、Safe Math 等模块，能够满足绝大部分的基础安全需求。Solmate 作为轻量级安全库，能优化 Gas 消耗，并支持 EIP-20、EIP-721 等标准实现。其次，开展防御性编程实践。通过输入校验（见图 1），对 sg.sender、_value 等外部输入进行严格检查，遵循“检查—生效—交互”模式，先更新本地状态再执行外部调用，防止重入攻击。Solidity 0.8.0+ 默认启用溢出检查，并支持 unchecked 块，避免 delegatecall 调用不可信合约。最后，坚持权限最小化原则。mint、pause 等关键函数需通过 onlyOwner、onlyRole 限制调用者。避免使用 public 修饰符暴露内部函数，减少攻击面。对管理员权限设置时效性，如通过多签合约限制 owner 的操作，防止单点失效。

（2）多维度验证安全。在测试与审计阶段，可通过自动化测试工具链、形式化验证、第三方审计来多维度验证智能合约安全，防范智能合约漏洞。自动化测试工具链是由单元测试、模糊测试、符号执行等测试工具构成的链条。单元测试采用 Hardhat、Truffle 框架编写测试用例，覆盖正常流程、异常流程以及边界条件。模糊测试依托 Echidna、Foundry 生成地址、函数等随机输入，监控 revert、panic 等合约异常，发现逻辑漏洞。以 batchTransfer 函数为例，模糊测试可自动生成 _ receivers.length 和 _value 的组合，触发溢出场景。符号执行采用 Manticore、MythX 遍历所有可能的执行路径，验证条件分支的完整性。形式化验证通过数学证明确保合约行为符合规范，适用于清算、铸币等高风险场景，如将 Solidity 代码转换为 F* 语言模型，以验证无重入、无溢出、转账原子性等安全属性。第三方审计由专业安全团队负责，围绕代码逻辑、交互安全以及合规性 3 个方面，展开代码审计。

（3）动态监控与应急响应。运行阶段的智能合约漏洞风险防范主要通过动态监控与应急响应来实现，具体包括代理模式、链上监控与预警以及应急响应机制。代理模式通过 Proxy 合约分离逻辑与状态，支持安全升级，具体的实现方式包括 OpenZeppelin Transparent Proxy 和 UUPS Proxy，前者可以防止代理合约与逻辑合约的函数冲突，后者则可以为逻辑合约自身控制升级权限。链上监控与预警通过 Tenderly、Dune Analytics 等监控工具的部署，实施检测大额转账、未授权操作、预言机异常等风险。例如，设置单笔转账的阈值，一旦单笔转账超出阈值，自动触发预警。监控、验证 onlyOwner 函数的调用者是否为管理员地址。对比 Chainlink、Band Protocol 等多个预言机数据源，检测数据操纵问题。应急响应机制能在漏洞检出后，第一时间采取暂停、资产冻结、保险赔付等应急措施，降低漏洞的危害。例如，通过 Pausable 模块，暂停转账、铸币等关键操作，为漏洞修复争取时间。提前于合约中设计 freeze 函数，并限制调用权限，对恶意地址的资产进行冻结。为 DeFi 协议等高风险合约购买区块链保险，降低资产损失风险。

结语

智能合约漏洞的本质是代码逻辑与 EVM 运行环境、业务需求的不匹配。从技术原理看，漏洞可分为重入攻击、整数溢出与下溢、时间戳依赖、权限控制缺陷、Gas 限制攻击、业务逻辑漏洞、预言机操控、函数可见性错误等。理解这些漏洞的形成机制不仅是构建安全开发流程的基础，还是抵御潜在攻击的关键，应根据漏洞成因，构建覆盖开发、测试、运行全生命周期的安全防护体系，提高智能合约的安全性。

引用

[1] 李芝.基于交易记录和智能合约的区块链安全检测[D].江苏:南京邮电大学,2022.

[2] 薛伟.区块链智能合约安全性分析与改进方法[J].科学与信息化,2025(5):70-72.

[3] 付梦琳,吴礼发,洪征,等.智能合约安全漏洞挖掘技术研究[J].计算机应用,2019,39(7):1959-1966.

[4] 倪远东,张超,殷婷婷.智能合约安全漏洞研究综述[J].信息安全学报,2020,5(3):78-99.

[5] 董伟良,刘哲,刘逵,等.智能合约漏洞检测技术综述[J].软件学报,2024,35(1):38-62.