bits lab

# BitsLab 重磅

# 2024

# 新兴生态公链全景观察及安全研究报告

# 目录 CONTENTS

# 1 引言

## 1.1 报告背景

2024年，新兴生态公链正在快速演进，以其技术创新性、网络性能优化和安全性增强，逐渐成为下一代 Web3 基础设施的中坚力量。这些公链不仅提供了更高效的交易处理能力，还承载了多样化的应用场景，为用户和开发者带来了全新的去中心化体验。

然而，随着新兴公链生态的扩展和应用落地，安全问题也愈发凸显。安全漏洞、智能合约攻击、网络共识机制的潜在风险等问题不仅影响了整个区块链生态的健康发展，也对链上资产的安全性构成了威胁。因此，针对新兴生态公链的全面安全研究显得尤为重要，既是为了保护用户的资产，也是为了保障区块链生态的稳定性与持续性发展。

BitsLab 作为全球领先的区块链安全与基础设施建设公司，始终以推动区块链行业的安全发展为己任。自团队通过 MoveBit 进入 Move 生态以来，BitsLab 迅速成为 Move 语言和 Move 生态的早期贡献者，参与了众多安全审计服务、标准制定及形式化验证的最佳实践，得到了Aptos、Sui等主流生态的广泛认可。BitsLab 凭借其安全解决方案，覆盖了 Move 生态项目的80%以上，并成为该生态中核心建设的一部分。随着行业的扩展与技术的发展，BitsLab不断提升自身的技术实力，并将目光扩展至其他新兴公链生态，包括 TON、比特币拓展生态等，为这些生态提供安全保障与技术支持。

本报告的目的在于帮助行业内外的参与者，尤其是开发者和投资者，了解2024年新兴公链生态的全景，并深入剖析其安全性与技术趋势，助力用户和开发者在不断变化的区块链世界中做出更为明智的决策。

## 1.2 报告聚焦

在新兴的 Web3 世界中，公链生态日益复杂化，各种技术堆栈和创新层出不穷，普通用户和开发者面对这一复杂环境往往难以全面掌握其内在逻辑及技术安全要点。

**报告将聚焦于以下几个关键前沿新兴生态：**

**(1) Move 生态：**
作为新兴公链语言中备受瞩目的开发语言之一，Move 生态凭借其独特的安全性和灵活性，吸引了大量开发者和项目方的参与。BitsLab 通过其核心产品 MoveBit，已经成为 Move 生态安全的领军者，本报告将详细分析 Move 生态的安全性和未来发展趋势。

**(2) TON 生态：**
TON 作为 Telegram 支持的区块链项目，近年来发展迅速。其高扩展性和快速交易处理能力使其在新兴公链中占据了一席之地。本报告将分析 TON 的技术优势和其在 Web3 生态中的安全挑战。

**(3) 比特币拓展生态：**
随着比特币拓展生态解决方案的不断成熟，如闪电网络（Lightning Network）等扩展技术逐渐成为关注焦点。BitsLab 将通过其专业审计和基础设施建设，为比特币拓展生态提供安全研究和技术趋势分析。

**除了对这些生态的安全现状和技术趋势进行详细分析外，本报告还将提供针对开发者的技术指导，涵盖智能合约审计、漏洞检测、形式化验证等多个方面。本报告将以这些方面为基础，帮助开发者理解和应对新兴生态中的安全挑战。**

# 2 前沿新兴生态公链

## 2.1 新兴生态公链的技术创新：Move 生态

### 2.2.1 Move 编程语言：区块链智能合约的革新力量

Move 语言最初由 Facebook（现Meta）为 Diem（Libra）项目开发，旨在解决传统智能合约语言的性能和安全瓶颈。Move 的设计强调资源的明确性与安全性，确保区块链上每个状态变化的可控性。这一创新编程语言具备以下显著优势：

**(1) 资源管理模型：**
Move 将资产视为资源，使其不可复制或销毁。这种独特的资源管理模型避免了智能合约中常见的双重支付或意外销毁资产问题。

**(2) 模块化设计：**
Move 允许智能合约以模块化的方式构建，提高代码复用性，并且降低了开发复杂度。

**(3) 高安全性：**
Move 在语言层面内置了大量的安全检查机制，防止常见的安全漏洞，比如重入攻击（reentrancy attacks）等。

### 2.2.2 Aptos 与 Sui：新兴生态公链的智能合约架构

#### Aptos：高效与安全并重

Aptos 作为基于 Move 语言开发的全新公链项目，专注于提供高性能和高安全性的区块链基础设施。
其创新智能合约架构具有以下特点：

- 并行执行引擎：Aptos 采用并行执行技术，大幅提高交易吞吐量。通过并行化交易处理，避免了传统区块链单线程执行的瓶颈，从而实现更高的可扩展性。
- 模块化设计与定制：Aptos 支持智能合约的模块化设计，开发者可以根据需求对合约进行灵活定制和扩展。
- 用户友好性：Aptos 致力于简化智能合约开发流程，降低门槛，吸引更多开发者参与其生态建设。

#### Sui：以可扩展性为核心

Sui 同样基于 Move 生态，专注于优化区块链的可扩展性和用户体验。其智能合约架构具备以下技术创新：

- 对象中心设计：Sui 的智能合约架构围绕“对象”进行设计，允许开发者灵活定义资产的状态与行为。这种对象导向的架构提高了合约的可编程性与可扩展性。
- 水平扩展性：Sui 通过创新的共识算法，允许区块链随着节点数量的增加而线性扩展，从而支持更大规模的用户和应用。
- 分片技术：Sui 使用分片技术来提升网络吞吐量，确保智能合约的执行效率不受全网交易量的影响。

## 2.2 TON 生态

### 2.2.1基本介绍与架构

TON（The Open Network）是由 Telegram 创建的区块链和数字通信协议，旨在构建一个快速、安全和可扩展的区块链平台，为用户提供去中心化的应用和服务。通过结合区块链技术和 Telegram 的通信功能，TON 实现了高性能、高安全性和高可扩展性的特点。它支持开发者构建各种去中心化应用，并提供分布式存储解决方案。与传统的区块链平台相比，TON 具有更快的处理速度和吞吐量，并采用了 Proof-of-Stake 共识机制。

#### 灵活且可分片的 PoS 架构

TON 采用了权益证明共识机制，并通过其图灵完备的智能合约和异步区块链实现了高性能和多功能性。TON 的闪电般快速且低成本的交易由链的灵活且可分片的架构支持。这种架构允许其在不损失性能的情况下轻松扩展。动态分片涉及初步开发的具有各自目的的单独分片，这些分片可以同时运行并防止大规模积压。TON 的区块时间为5秒，最终确定时间少于6秒。
现有基础设施分为两个主要部分：

- 主链（Masterchain）：负责处理协议的所有重要和关键数据，包括验证者的地址以及验证的币量。
- 工作链（Workchain）：连接到主链的次级链，包含所有交易信息及各种智能合约，每个工作链可以有不同的规则。

### 2.2.2扩展用例和优势

TON 基金会是由 TON 核心社区运营的 DAO，为 TON 生态系统中的项目提供各种支持，包括开发者支持和流动性激励计划。
2024年，TON 社区在多个方面取得了显著进展：

- TON Connect 2.0的推出：提供了一种直观的方式连接钱包和应用程序，改善用户体验。
- TON Verifier：由 Orbs 团队创建的智能合约检查器，提升了合约的可靠性。
- Blueprint 开发工具：帮助开发者编写、测试和部署智能合约。
- Sandbox 开发者工具包：适用于从企业到政府的各种用例。
- Tact、Func 及其他新支持语言：促进更强大的编程环境。
- 开发者支持：TON 基金会与 DoraHacks 合作，推出了为期三个月的线上黑客松。
- TON Hubs 国际化：在全球多个城市启动了国际中心。
- DeFi 流动性激励计划：为项目提供资金，促进 TON DeFi 领域的可持续性。

## 2.3 比特币拓展生态

### 2.3.1什么是比特币拓展生态?

比特币拓展生态主要指的是围绕比特币基础网络开发的各种扩展解决方案和应用生态系统。比特币最初设计主要用于点对点支付和价值储存，但随着区块链技术的发展，比特币社区和开发者们也在探索如何在其基础上增加更多的功能，特别是在智能合约、去中心化金融（DeFi）、NFT 和更高效的交易扩展上。

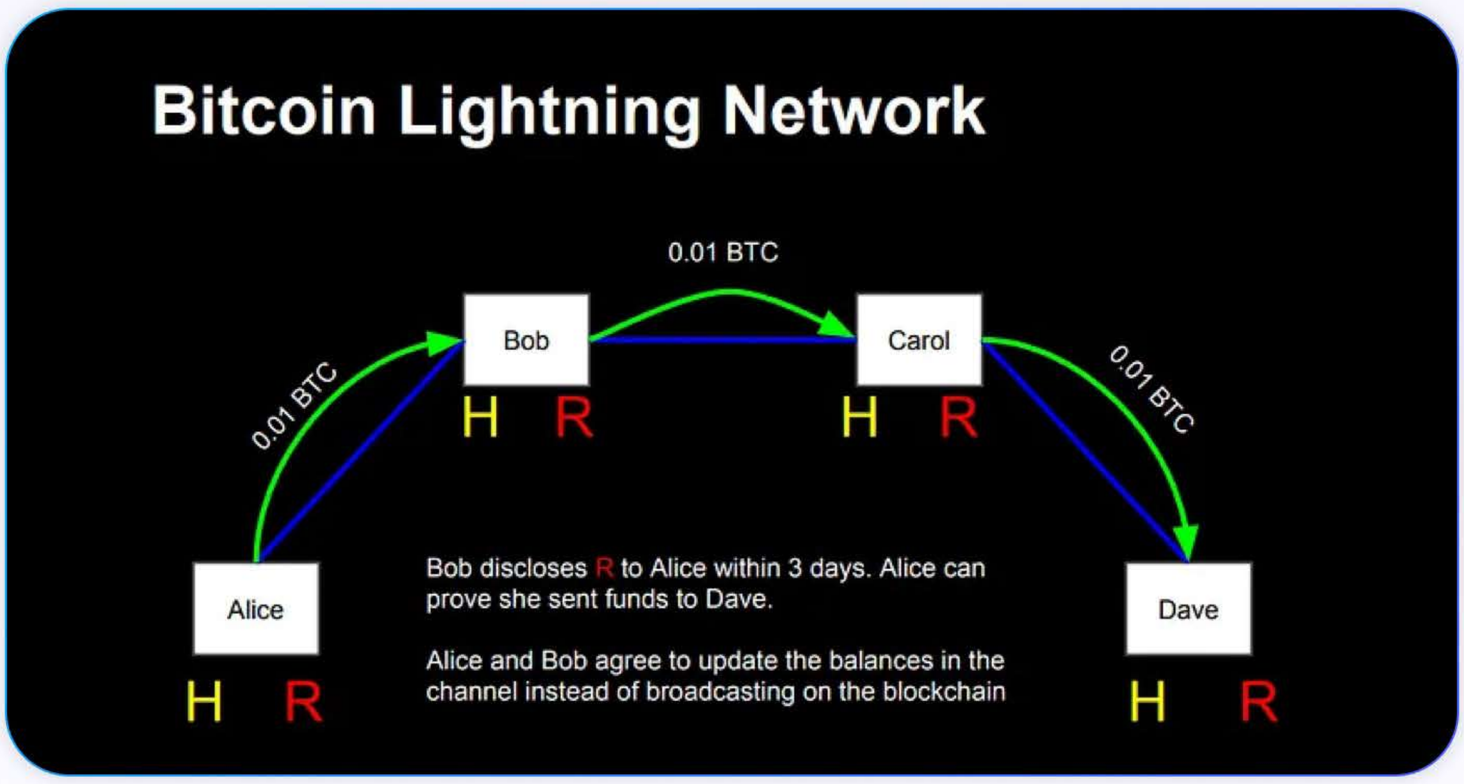
## 2.3.2比特币拓展生态如何运作?

比特币拓展生态的运作主要依赖于在比特币主链之上或之外构建的扩展技术和协议，这些技术和协议使得比特币能够支持更多样化的应用场景。以下是比特币拓展生态中几种关键技术的运作原理：

### (1) 闪电网络（Lightning Network）

闪电网络是比特币 Layer 2 最成熟、应用最广泛的解决方案之一。它通过建立支付通道，将大量小额交易从主链移到链外，从而大幅提升比特币的交易速度和降低手续费。

**趋势：**闪电网络的基础设施在不断改进，用户体验也在提升，越来越多的商户开始支持闪电支付。
**挑战：**流动性问题和路由效率仍需进一步优化，尤其是在大额交易场景下。



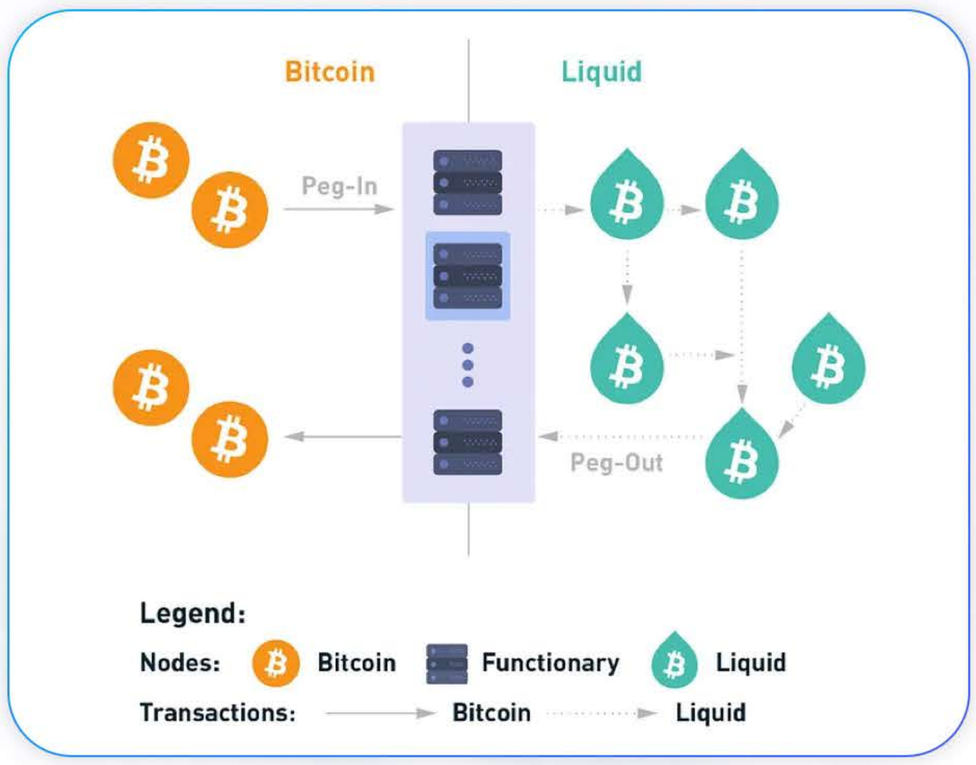

### (2) Liquid Network（LQ）

Liquid Network 是一个基于开源的 Elements 区块链平台运行的侧链，专为在交易所和机构之间实现更快的交易而设计。它由比特币公司、交易所和其他利益相关者组成的分布式联盟治理。Liquid 使用双向锚定机制，将 BTC 转换为 L-BTC，反之亦然。

Liquid 支持保密交易和代币化，因此适用于企业应用。如果比特币是互联网的价值层，Lightning 是比特币驱动的金融系统中的点对点支付网络，那么 Liquid 则是金融层，增加了多资产支持和金融工具，如证券和大宗商品。

与 Lightning 相比，Liquid 是一个比特币的 Layer 2 解决方案，专注于促进更大、更复杂的交易，例如发行和交易资产（如证券和稳定币）。Liquid 内置了保密交易功能，隐藏了交易金额和资产类型，而 Lightning 主要通过其链下交易提供隐私。虽然 Lightning 擅长于小额支付和日常交易，但 Liquid 更适合机构金融、资产发行和跨境交易。

目前已有超过 50 家交易所采用了 Liquid Network，它已经促成了数十亿美元的交易，证明了其在提升比特币用于机构交易中的效用方面的有效性。Liquid Network 能够为交易所提供更快的结算时间，从而提高了比特币市场的流动性，使机构能够更高效、安全地运营。




图片来源：https ://docs.liquid.net/docs/technical-overview

### (3) Rootstock 基础框架（RBTC）

Rootstock 自 2015 年诞生以来，是运行时间最长的比特币侧链，并在 2018 年启动了其主网。它的独特之处在于将比特币的工作量证明（PoW）安全性与以太坊的智能合约相结合。作为一个开源、兼容 EVM 的比特币 Layer 2 解决方案，Rootstock 为不断增长的 dApp 生态系统提供了入口，并致力于实现完全去信任化。

与 Liquid 类似，Rootstock 也使用双向锚定机制，因此用户可以轻松在 BTC 和 RBTC 之间进行交换。RBTC 是 RSK 区块链上的原生货币，用于支付矿工处理交易和合约的费用。Liquid 侧重于快速、私密的交易和资产发行，而 Rootstock 则通过智能合约扩展了比特币的 DeFi 和 dApp 生态系统。

截至撰写本文时，Rootstock 的总锁仓价值（TVL）超过 1.7 亿美元，市值为 3.8 亿美元。

### (4) $B^2$ Network

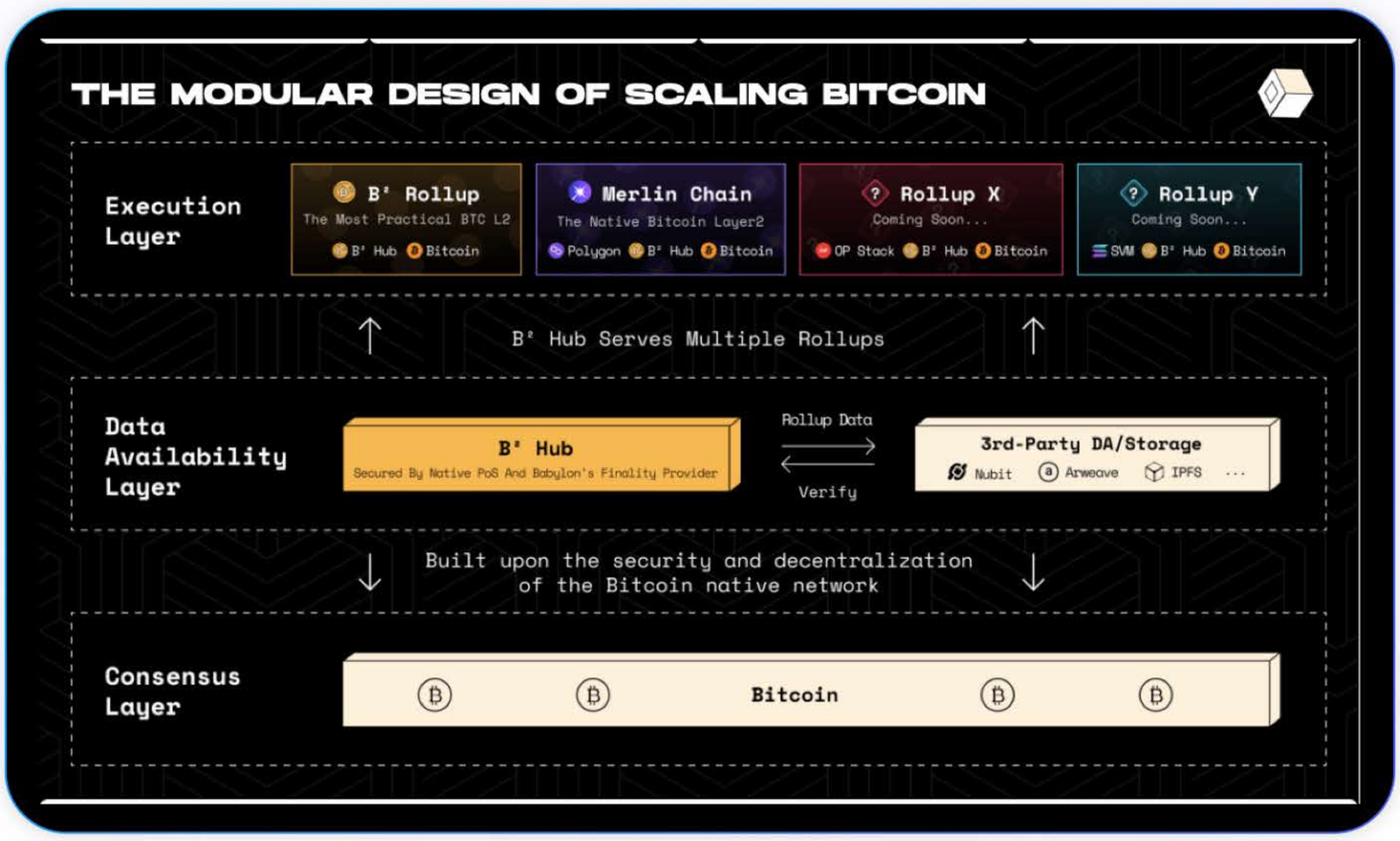
$B^2$ Network 的技术架构包括两层结构：Rollup 层、数据可用（DA）层。$B^2$ Network 旨在重新定义用户对比特币第二层解决方案的看法。

$B^2$采用 ZK-Rollup 作为 Rollup 层。ZK-Rollup 层采用 zkEVM 解决方案，负责在第二层网络内执行用户交易和输出相关的证明。用户的交易被提交并在 ZK-Rollup 层进行处理。用户的状态也存储在 ZK-Rollup 层。批量提案和生成的零知识证明被转发到数据可用层进行存储和验证。

数据可用性层包括分布式存储、$B^2$节点和比特币网络。该层负责永久存储 Rollup 数据的副本，验证 Rollup 的零知识证明，并最终在比特币上执行最终确认。

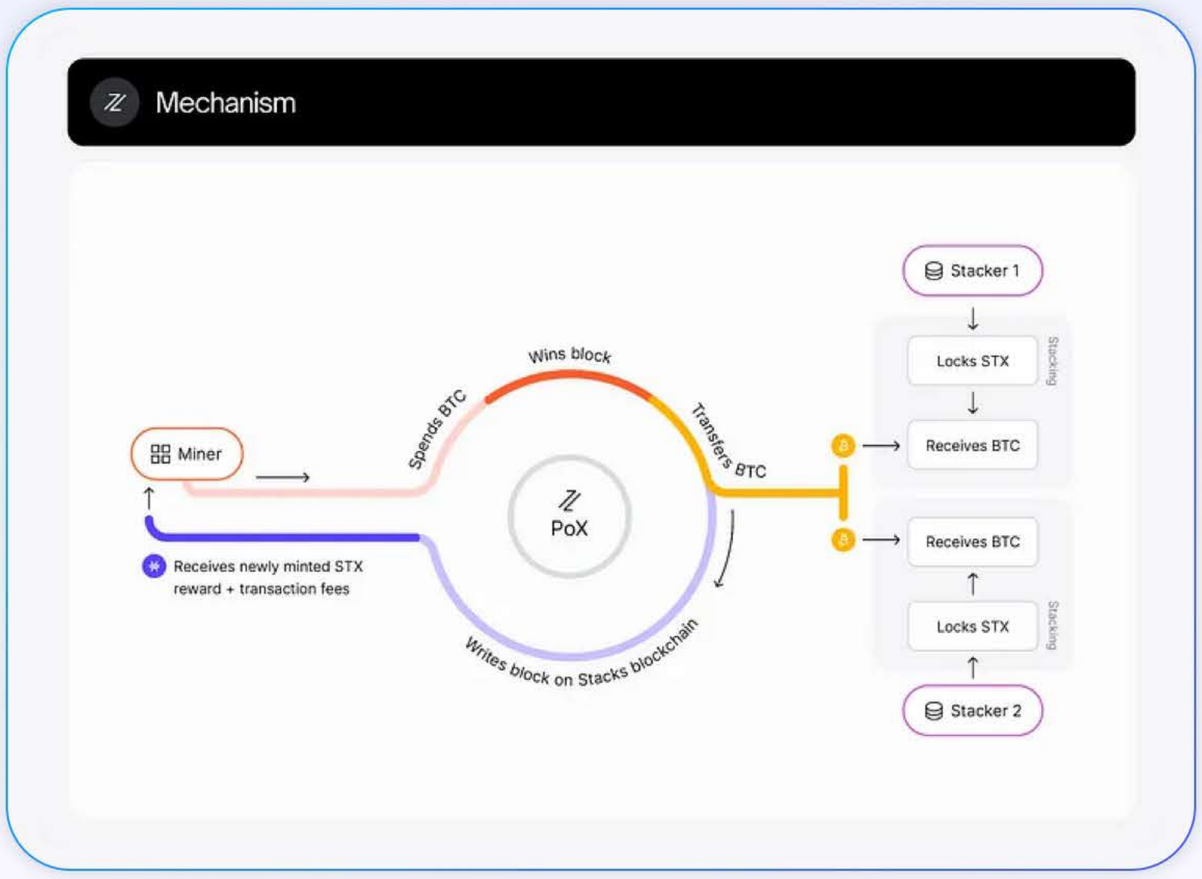
分布式存储是$B^2$ Network 的一个关键方面，它作为 ZK-Rollup 用户交易及其相关证明的存储库。通过分散存储，网络从根本上提高了安全性，减少了单点故障，并确保了数据的不可变性。

为了保证数据可用性，$B^2$还在比特币的每个区块中向比特币网络写入一个 Tapscript 脚本，如下图。该脚本锚定了这一时段的 Rollup 在去中心化存储中有效存储的数据路径和零知识证明，这一过程成本可控，每小时产生 6 笔交易。因此，用户在验证时会逐一比对交易和比特币 Layer1 上的 Taproot 脚本数据，以确保 Rollup 数据的最终可信性。



## (5) Stacks 协议（STX）

自 2018 年以 Blockstack 名义在主网上推出以来，Stacks 已成为领先的比特币 Layer 2 解决方案。




图片来源：https ://docs.stacks.co/stacks-101/proof-of-transfer

Stacks 直接连接到比特币，允许在比特币上构建智能合约、dApps 和 NFT，显著扩展了比特币的功能，使其不仅仅是一个价值存储工具。它采用了一种独特的转移证明（PoX）共识机制，将其安全性直接与比特币挂钩，而无需修改比特币本身。

Stacks 拥有超过 9900 万美元的总锁仓价值（TVL），其已建立的基础设施和不断壮大的开发者社区，使其成为该领域中不容忽视的项目。

## (6) Babylon

Babylon 的愿景是将比特币的安全性扩展到保护去中心化世界。通过利用比特币的三个方面 — — 其时间戳服务、区块空间和资产价值 — — Babylon能够将比特币的安全性传递到众多权益证明（PoS）链，从而创建更强大、统一的生态系统。

Babylon的比特币质押协议采用远程质押方法，通过密码学、共识协议创新和优化使用比特币脚本语言的方式克服了智能合约的缺失。Babylon的质押协议可以允许比特币持有者在无需桥接到 PoS 链的情况下，可信地质押比特币，并为该链提供完整的可削减安全性保证。Babylon的这一创新性协议消除了对已质押比特币进行桥接、封装、托管的需求。

Babylon 的一个关键方面是其BTC时间戳协议。它将其他区块链的事件时间戳到比特币上，使这些事件能够像比特币交易一样享受比特币的时间戳。这有效地借用了比特币作为时间戳服务器的安全性。BTC 时间戳协议实现了快速权益解绑、可组合的信任、降低安全成本,以最大限度地提高比特币持有者的流动性。协议被设计成模块化插件，可用于多种不同的 PoS 共识算法之上，并提供了可构建重置协议的基础。

## 2.4 应用链生态 Cosmos

Cosmos 在区块链领域引入了多项技术创新，以下是一些关键的创新点：

### 2.4.1 Tendermint 共识机制

Cosmos 使用的 Tendermint 是一种创新的共识算法，它将拜占庭容错（BFT）算法与传统区块链结合，提供了快速、低成本、且高效的共识解决方案。Tendermint 共识机制的优势包括：

**(1) 高吞吐量：**与比特币、以太坊等基于工作量证明（PoW）的网络相比，Tendermint 共识机制可以处理更多的交易。

**(2) 低延迟：**区块生成速度更快，交易确认时间短。

**(3) 安全性强：**能够容忍多达 1/3 的恶意节点，具备良好的抗攻击性。

### 2.4.2 Cosmos SDK

Cosmos SDK 是一个模块化、可定制的框架，开发者可以利用它轻松创建专属的区块链。这种框架支持开发者根据具体应用的需求进行灵活的开发，而不必从头构建区块链。Cosmos SDK 的创新包括：

**(1) 模块化设计：**开发者可以选择使用 SDK 中的现成模块，例如治理、代币、质押等功能，也可以开发自定义模块。

**(2) 轻松升级：**由于 Cosmos SDK 的模块化结构，项目可以在不影响核心网络运行的情况下升级和改进。

### 2.4.3 IBC（跨链通信协议）

Cosmos 的 IBC 协议（Inter-Blockchain Communication Protocol）是其最具创新性的技术之一。IBC 允许独立的区块链之间进行安全、去中心化的通信和资产转移，实现了真正的跨链互操作性。其关键功能包括：

**(1) 跨链资产转移：**区块链可以通过 IBC 协议转移资产，如代币、NFT 等。

**(2)跨链信息传递：**区块链之间可以共享数据或状态，从而实现更复杂的跨链应用场景，如去中心化交易所（DEX）和跨链智能合约。

**(3) 灵活性：**BC 协议支持不同共识机制和技术堆栈的区块链之间进行交互，这使得其成为多链生态中的基础通信层。

### 2.4.4 跨链区块链网络（Hub & Zone 架构）

Cosmos 的架构采用了 Hub 和 Zone 的模式，Hub（中心）作为跨链的核心节点，连接并协调多个 Zone（独立区块链）。这种架构的创新点在于：

**(1) 去中心化管理：**每个 Zone 都是独立的、自治的区块链，不需要依赖于单个中心化的管理节点。

**(2)高效的跨链连接：**通过 Hub，Zone 之间可以无缝进行跨链通信和资产流动，实现了真正的互联互通。

### 2.4.5 PoS（权益证明）和跨链质押

Cosmos 使用了 PoS（权益证明） 作为其共识机制的核心，确保网络的安全和运行。除了基础的 PoS 机制，Cosmos 的创新还包括：

**(1) 跨链质押：**通过 IBC，未来可能实现跨链质押，即用户可以在多个链上同时进行资产质押，从而提高质押资产的利用率。

**(2)灵活的治理机制：**Cosmos 的 PoS 机制结合了链上治理功能，持币者可以参与链上决策和网络升级等重要事项。

# 3 安全漏洞类型列表

## 3.1 L2/L1 跨链通信漏洞

跨链通信是提升区块链生态系统互操作性的重要手段，但在其实现过程中也存在诸多安全隐患。以下是主要的关注点：

**L2 未考虑 L1 的区块回滚**

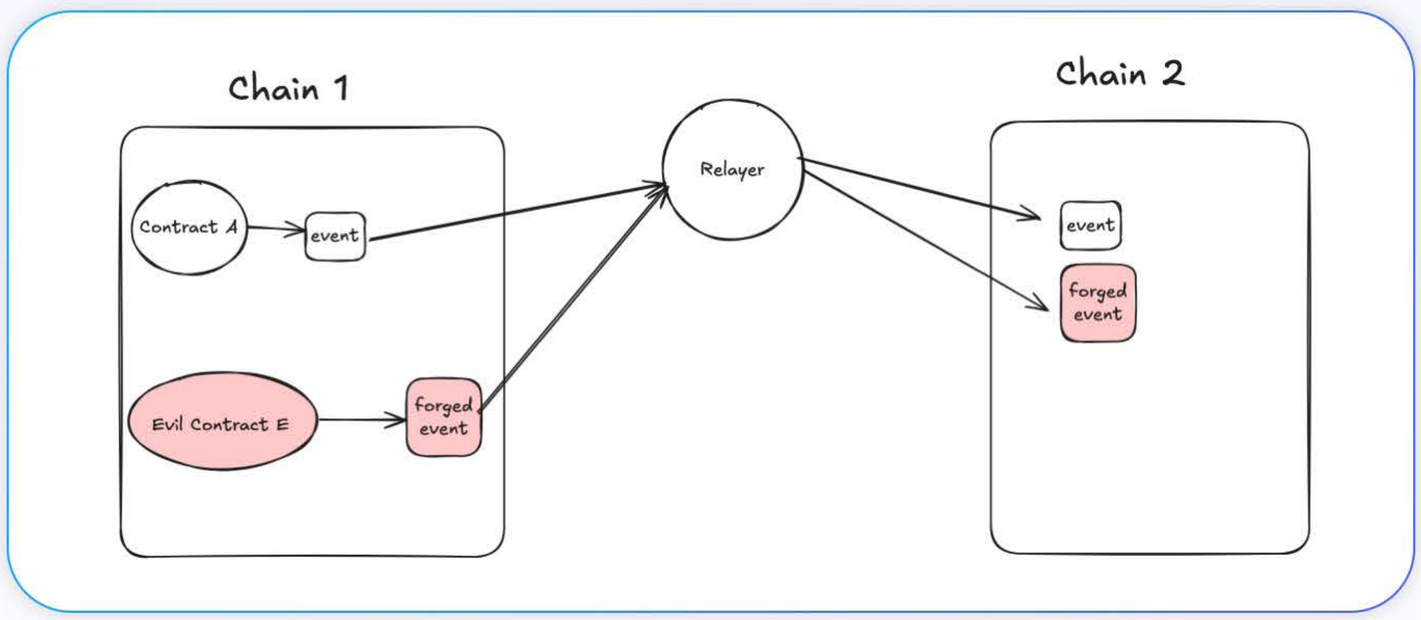
在发送 L1 交易，或者获取 L1 链上数据的时候，如果没有考虑这种情况，可能会导致资产损失。

**L2 未检测发送给 L1 的交易是否成功**

由于网络、gas 费用等问题，发送的交易可能会失败。如果没有考虑这种情况，可能导致项目或者用户的资产损失。

**链上事件伪造**

跨链桥在监听链上事件的时候，没有校验事件是否来自于指定的合约地址，其他合约可以伪造事件。




**同一个交易包含多个链上事件**

一个交易里面是可以包含多个事件，如果没有考虑这种情况，可能导致项目或者用户资产损失。

**轻客户端验证漏洞**

1、PoW 链未考虑私自挖矿攻击

2、没有采用官方推荐的算法

**中间人劫持**

在 L2/L1 跨链通信中，消息传递机制至关重要，需确保消息在传递过程中的完整性和保密性。消息在不同链之间的传递可能面临被篡改或监听的风险，因此需要使用加密手段保护信息的安全。此外，确保消息在链间转发时的不可否认性也是关键，以防止信息被恶意篡改。

**延迟和最终性问题**

跨链通信常常面临延迟和最终性的问题。由于不同链的共识机制和确认时间不同，跨链交易的确认时间可能不一致，导致状态更新的延迟，从而增加了潜在的安全风险。在设计跨链协议时，需明确最终性的定义，确保交易状态的一致性，防止因确认延迟而造成的双花攻击或状态不一致问题。

## 3.2 Cosmos 应用链漏洞

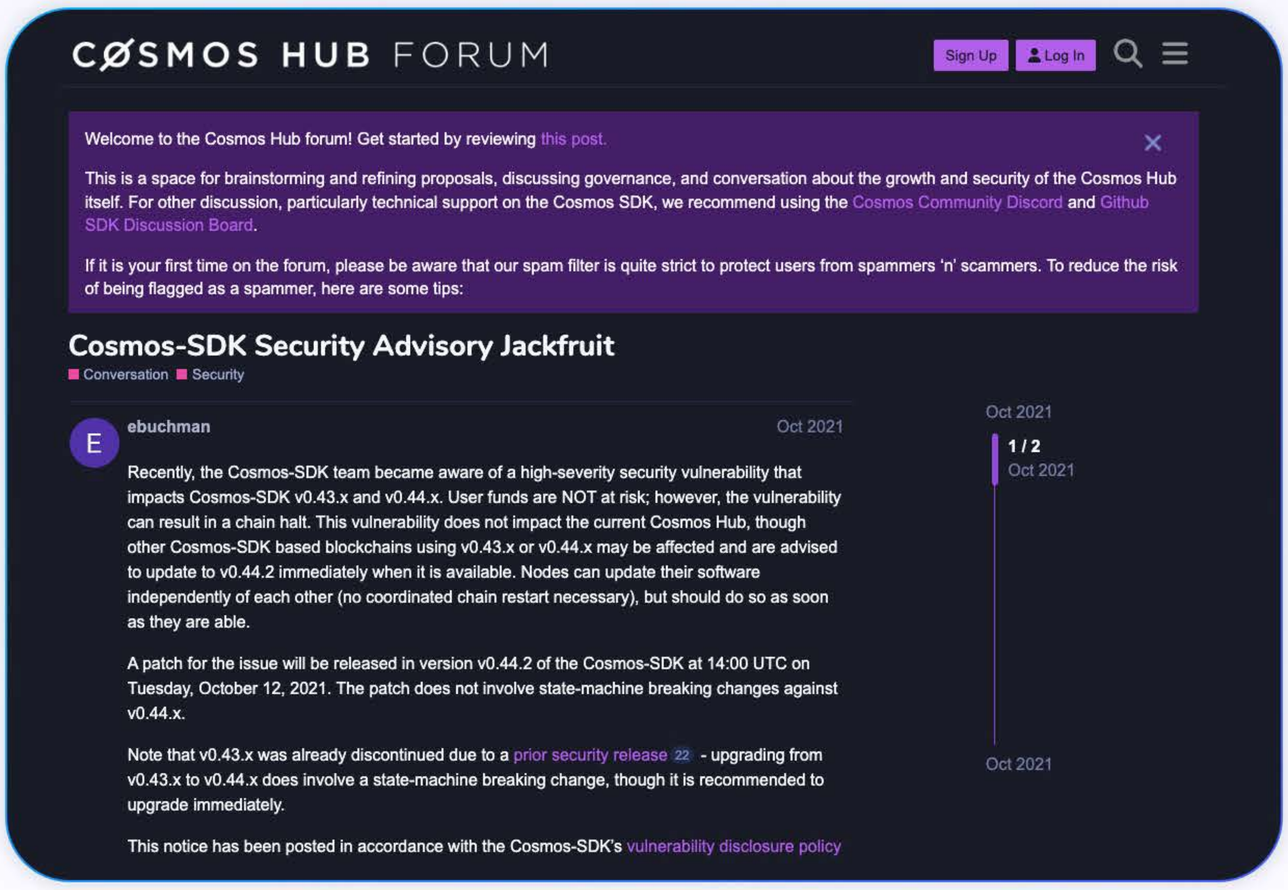
Cosmos 作为一个以区块链互操作性为核心的生态系统，允许不同的区块链通过IBC（跨链通信协议）进行连接。然而，Cosmos 应用链在实现过程中也可能存在一些漏洞和安全隐患。以下是主要的关注点：

**BeginBlocker 和EndBlocke r崩溃漏洞**

BeginBlocker 和EndBlocker 是模块开发人员可以在其模块中实现的可选方法。它们分别在每个区块的开始和结束时触发。
在BeginBlock 和 EndBlock方法中使用崩溃来处理错误可能会导致链在出现错误时停止运行。

**本地时间使用不正确**

由于不同节点的本地时间会有差异，如果在生成共识的时候，没有考虑到这点，会导致共识问题。




图片来源https://forum.cosmos.network/t/cosmos-sdk-security-advisory-jackfruit/5319

**随机数使用不正确**

由于不同节点生成的随机数不一样，如果在生成共识的时候，没有考虑到这点会导致共识问题。

**Map迭代功能的不正确使用**

go 语言的 map 的迭代是非确定性的，如果在生成共识的时候，没有考虑到这点，会导致共识问题。示例代码如下：

```
313
314  func (k Keeper) BuildDependencyDag(ctx sdk.Context, txDecoder sdk.TxDecoder, anteDepGen sdk.AnteDepGenerator, txs [][]byte) (*types.
315          defer MeasureBuildDagDuration(time.Now(), "BuildDependencyDag")
316          // contains the latest msg index for a specific Access Operation
317          dependencyDag := types.NewDag()
318          for txIndex, txBytes := range txs {
319                  tx, err := txDecoder(txBytes) // TODO: results in repetitive decoding for txs with runtx decode (potential optimizat
320                  if err != nil {
321                          return nil, err
322                  }
323                  // get the ante dependencies and add them to the dag
324                  anteDeps, err := anteDepGen([]acltypes.AccessOperation{}, tx)
325                  anteDepSet := make(map[acltypes.AccessOperation]struct{})      map
326                  for _, dep := range anteDeps {
327                          anteDepSet[dep] = struct{}{}
328                  }
329                  // pass through set to dedup
330                  if err != nil {
331                          return nil, err
332                  }
333                  for accessOp := range anteDepSet {                             iterate map
334                          err = types.ValidateAccessOp(accessOp)
335                          if err != nil {
336                                  return nil, err
337                          }
338                          dependencyDag.AddNodeBuildDependency(ANTE_MSG_INDEX, txIndex, accessOp)
339                  }
340
341                  msgs := tx.GetMsgs()
342                  for messageIndex, msg := range msgs {
343                          if types.IsGovMessage(msg) {
344                                  return nil, types.ErrGovMsgInBlock
345                          }
346                          msgDependencies := k.GetMessageDependencies(ctx, msg)
347                          dependencyDag.AddAccessOpsForMsg(messageIndex, txIndex, msgDependencies)
348                          for _, accessOp := range msgDependencies {
349                                  // make a new node in the dependency dag
350                                  dependencyDag.AddNodeBuildDependency(messageIndex, txIndex, accessOp)
351                          }
352                  }
```

**模块顺序不合理导致的问题**

Cosmos 应用链是由多个模块组成的，在某些事件处理中，模块之间是由顺序的。如果顺序设置不合理，可能会导致安全问题。

**交易失败但数据未回滚**

在 Cosmos 应用链中，如果一笔交易执行失败，但由于设计缺陷，数据状态未能回滚到交易执行前的状态，就可能导致链上的数据不一致。这种情况不仅影响用户的信任，还可能造成资金损失。因此，设计时必须确保智能合约能妥善处理失败的交易，确保状态的一致性和可靠性，必要时实现自动回滚机制。

**错误的状态验证**

Cosmos 应用链中的状态验证逻辑需要非常严谨。如果状态验证不准确，可能导致不合法的交易被确认，从而影响链的安全性。开发者需仔细设计状态转移逻辑，并进行全面的测试，以防止因为错误的状态验证导致的漏洞。

**跨链消息传递安全**

Cosmos 的 IBC 机制使得不同应用链之间可以传递消息，但这也引入了潜在的安全风险。

例如，如果跨链消息在传递过程中被篡改，可能导致错误的状态更新或攻击者利用该消息进行恶意操作。应采取加密和签名机制，确保消息的完整性和真实性，防止消息在传递过程中的篡改。

**合约升级与版本管理问题**

Cosmos 应用链的合约在使用过程中可能需要进行升级，但合约升级不当可能导致旧合约状态的不兼容或安全漏洞。开发者需制定明确的合约升级策略，包括版本管理和迁移方案，确保升级过程中不会影响链的正常运行。

**经济模型与激励机制**

Cosmos生态中的经济模型设计直接影响到链的安全性和稳定性。如果激励机制设置不合理，可能导致参与者行为失衡，出现恶意行为或经济攻击。需要对经济模型进行全面评估，确保激励机制能有效维持网络的安全性和健康性。

**治理机制的漏洞**

Cosmos应用链的治理机制允许持币者参与链的决策，但若治理机制设计不当，可能导致治理攻击或集权化问题。应确保治理机制的公平性和透明性，以防止少数人操纵链的决策过程。

## 3.3 比特币拓展生态漏洞

**比特币脚本构造漏洞**

比特币脚本很多情况下是实时生成并部署到 Bitcoin 上，而且脚本的内容包含用户提供的数据。如果脚本构造不安全，会导致资产损失。

**未考虑衍生资产导致的漏洞**

常见的比特币衍生资产是符文和 Runes ，如果L2在操作用户资产的时候，只考虑原生的 BTC ，没有考虑衍生资产，会导致用户资产损失。

**UTXO 金额计算错误漏洞**

1、计算交易费的时候失误

2、计算找零金额失误

比如在如下代码中，找零输出计算实现包含一个条件：只有当找零金额大于或等于 546 聪时才会添加找零输出。这个值对应于传统 P2PKH 交易的尘埃限制。然而，不同地址类型的尘埃限制有所不同。特别是，对于 Taproot 地址，尘埃限制为 330 聪。

这个硬编码的值没有考虑到 Taproot 地址较低的尘埃限制，可能会导致涉及 Taproot 地址的交易中小额资产的损失。关于各种地址类型尘埃限制的详细信息，可以在 Bitcoin Core 源代码和 BitcoinTalk 论坛的讨论中找到。

https://github.com/bitcoin/bitcoin/blob/e9262ea32a6e1d364fb7974844fadc36f931f8c6/src/policy/policy.cpp#L26

https://bitcointalk.org/index.php?topic=5453107.msg62262343#msg62262343

```
if (change >= 546) {
  psbt.addOutput({
    address: lenderPaymentAddress,
    value: change,
  });
} else {
  fee += change;
}

```

**SPV 验证漏洞**

1、没有校验区块头时间戳

2、没有校验区块头的工作量证明

**未考虑回滚情况**

由于比特币是基于 PoW 的，所以经常发区块重组。在发送比特币交易，或者获取比特币链上数据的时候，如果没有考虑这种情况，可能会导致资产损失。

**未考虑比特币交易是否发送成功**

比特币交易发送之后不一定会被矿工打包，所以需要检测交易是否成功上链。同时，第三方可以构造交易费更高的交易，来让L2发送的交易在比较长的时间内一直在交易内存池里面而不是被打包。

**绝对时间锁和相对时间锁混淆漏洞**

在构造比特币脚本的时候，如果混淆这2种时间，严重情况下会导致资产损失。

**哈希时间锁合约（HTLC）的时间设置不合理**

常见情况有如下3种：

- 时间设置太大
- 时间设置太小
- L1和L2时间不同步

# 3.4 编程语言常见漏洞类型

## 3.4.1 整数溢出

整数溢出发生在数值超出类型所能表示的范围时，会导致数值回绕或逻辑错误，影响程序数据的准确性。Rust 在调试模式下自动检测溢出，而 Go 需要手动添加边界检查。

```rust
fn integer_overflow() -> u8 {
    let x :u8 = u8::MAX;
    let y :u8 = 1;
    let result = x + y;
    println!("{} + {} = {}", x, y, result);
    result
}
```

## 3.4.2 死循环

死循环是指程序在特定条件下进入无限循环，消耗系统资源，导致程序卡死或无法响应。避免此问题的关键在于设置合理的循环退出条件，并使用Rust的超时机制或 Go 的 context 包控制长时间运行的循环。

```rust
fn infinite_loop_demo() {
    let mut counter = 0;
    loop {
        println!("Counter: {}", counter);
        if counter == -1 {
            break;
        }
        counter += 1;
    }
}
```

## 3.4.3 无限递归调用

无限递归调用指递归函数缺少终止条件，导致栈溢出并引发崩溃。确保递归有明确的基准条件，并根据需求限制递归深度，可以有效防范该问题。

```rust
fn infinite_recursion() {
    println!("Entering infinite recursion...");
    infinite_recursion();
}

fn main() {
    infinite_recursion();
}
```

Rust 会在调试信息中显示栈溢出错误，类似于以下输出：

```
thread 'main' has overflowed its stack
fatal runtime error: stack overflow
```

## 3.4.4 竞争条件

当多个线程未同步地访问共享资源时，会发生数据竞争，从而导致数据不一致。Rust 通过所有权机制和线程安全库来避免竞争条件，Go 则通过 channel 和 sync包提供并发支持。

**如下 Unsage Rust 示例，演示了两个线程同时访问和修改同一个共享变量，导致未定义的行为。这个代码会造成数据竞争，因为没有使用任何同步机制保护共享资源：**

```rust
use std::thread;
use std::sync::Arc;
use std::time::Duration;

fn race_condition_demo() {
    // Wrap a shared integer variable using Arc for reference counting
    let counter = Arc::new(0);
    let mut handles = vec![];

    // Create 10 threads, each incrementing the counter variable by 1
    for _ in 0..10 {
        let counter = Arc::clone(&counter);

        let handle = thread::spawn(move || {
            for _ in 0..1000 {
                // Read and modify the counter, causing a race condition
                unsafe {
                    let raw_ptr = Arc::as_ptr(&counter) as *mut i32;
                    *raw_ptr += 1; // Race condition: multiple threads modifying the same data simultaneously
                }
            }
        });

        handles.push(handle);
    }

    // Wait for all threads to complete
    for handle in handles {
        handle.join().unwrap();
    }

    // Pause briefly to ensure all thread output is complete
    thread::sleep(Duration::from_millis(10));
    println!("Final counter value: {}", *counter);
}

fn main() {
    race_condition_demo();
}

```

虽然 Safe Rust 防止数据竞争，但逻辑上的竞争仍然可能发生。**逻辑条件竞争**（Race Condition）指的是代码在特定执行顺序下可能产生非预期的行为，这种情况是逻辑上的竞争。比如在以下场景中：

1.**时间敏感的操作：**两个线程之间的操作顺序可能影响最终的逻辑结果。例如：

a.一个线程检查某个条件是否成立，而在此期间另一个线程修改了该条件。这在 Rust 中可以通过 Arc<Mutex<T>> 这样的同步机制来协调访问顺序，但并不能防止逻辑上的条件竞争。

2.**双重检查锁（Double-Checked Locking）**：如果多个线程尝试初始化一个共享资源并且都认为资源未被初始化，则可能导致逻辑错误。这种情况下，虽然不会发生数据竞争，但可能会产生意外的逻辑错误。

3.**不正确的锁顺序:** 如果使用多个锁，线程可能会以不一致的顺序获取锁，从而可能造成死锁。Rust 的类型系统并不能防止这种死锁类型的竞争条件。

以下是 Safe Rust 的条件竞争漏洞演示。在这个例子中：

- 两个线程分别对共享变量 data 加 1。每个线程都会锁住 data，然后增加其值。
- 因为操作是分步完成的，即使数据受 Mutex 保护，线程 1 和线程 2 的执行顺序会影响最终的打印结果。理论上，最终值应为 2，但具体的线程打印输出顺序可能不固定。

```rust
use std::sync::{Arc, Mutex};
use std::thread;

fn main() {
    let data = Arc::new(Mutex::new(0));

    let data1 = Arc::clone(&data);
    let handle1 = thread::spawn(move || {
        let mut num = data1.lock().unwrap();
        *num += 1;
        println!("Thread 1 incremented value to: {}", *num);
    });

    let data2 = Arc::clone(&data);
    let handle2 = thread::spawn(move || {
        let mut num = data2.lock().unwrap();
        *num += 1;
        println!("Thread 2 incremented value to: {}", *num);
    });

    handle1.join().unwrap();
    handle2.join().unwrap();

    println!("Final value: {}", *data.lock().unwrap());
}

```

## 3.4.5 异常崩溃

异常崩溃通常由未处理的错误引发，导致程序意外终止。Go 使用 defer/panic/recover 捕获异常，Rust则使用 Result 和 Option类型系统，提供更健壮的错误处理。
如下是没有捕获 panic 导致程序崩溃的例子：

```rust
fn main() {
    // Call a function that may cause a panic
    cause_panic_with_unwrap();
}

fn cause_panic_with_unwrap() {
    // Create an Option with a None value
    let value: Option<i32> = None;

    // Use unwrap to attempt to get the value inside None, which will cause a panic
    let _unwrapped_value = value.unwrap();
}
```

## 3.4.6 除0漏洞

除0漏洞指程序执行除法操作时分母为零，可能引发异常或崩溃。建议在进行除法前检查分母值，防止零值操作并确保程序的稳定性。

```
fn divide_by_zero_demo() {
    // Define a numerator
    let numerator = 10;
    // Define a denominator as zero
    let denominator = 0;
    // Attempt to divide by zero, which will cause a panic
    let _result = numerator / denominator;
}
```

## 3.4.7 类型转换

类型转换错误通常由不安全或不兼容的转换导致，可能引发不可预料的行为。Go 和 Rust 在转换时会提示类型不兼容，Rust 在转换上更严格，需要使用“as”操作显式进行。

在如下示例代码中，用户添加了amount 数量代币的流动性，然后系统记录其流动性数量。如果 amount = (u128::MAX << 64) | 1，那么实际只支付了1的代币，但是流动性余额记录为340282366920938463444927863358058659841。

```
fn stake(amount :u128, staker :Address) {
    let liquid_max : u64 = 1000;
    let trs = amount as u64;
    if trs < liquid_max
    {
        transfer_token(to,trs);
    }else
    {return;}
    staker.liquid = amount;
}
```

## 3.4.8 数组越界

数组越界指访问数组中无效索引位置，导致内存访问错误或程序崩溃。

如下示例代码演示了数组越界导致程序崩溃：

```
fn main() {
    // Call a function that will attempt to access an out-of-bounds e
    array_out_of_bounds_demo();
}
fn array_out_of_bounds_demo() {
    // Define an array with 3 elements
    let array = [1, 2, 3];
    // Attempt to access an element beyond the array's length
    let out_of_bounds_element = array[5];
    // Print the out-of-bounds element (this line will not execute d
    println!("Out-of-bounds element: {}", out_of_bounds_element);
}

```

## 3.5 p2p 网络漏洞

P2P（点对点）网络在区块链系统中用于分布式节点间的直接连接与通信。尽管 P2P 网络为去中心化系统提供了网络基础，但其也面临一系列安全漏洞与攻击风险。

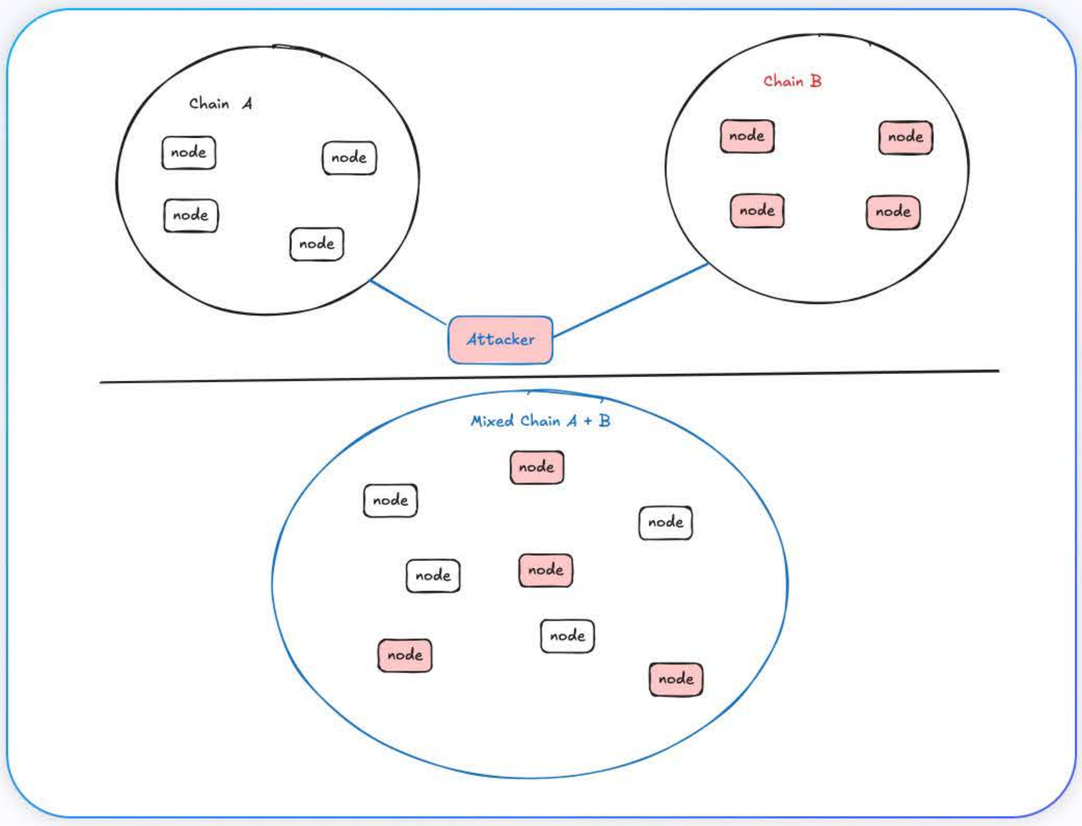
从安全角度来说，p2p 网络类型可以分为2种，一种是无需许可的，这种在L1上使用比较多。
一种是需要许可的，这种在L2比较多。

无需许可的 p2p 网络里面，很多漏洞的攻击是基于 sibil 攻击的。需要许可的网络里面，我们不能假设所有节点都是可信的，从安全的角度来说，要假设至少有1个恶意节点。

**p2p网络的常见漏洞类型如下：**

**(1) 异形攻击漏洞：**

异形攻击又称地址池污染，是指诱使同类链的节点互相侵入和污染的一种攻击手法，漏洞的主要原因是同类链系统在通信协议上没有对非同类节点做识别。
以太坊某些同类链曾经出现过类似的漏洞。以太坊同类链（具体的说是使用以太坊 P2P discv4 节点发现协议的公链，
包括以太坊、以太经典）由于使用了兼容的握手协议，无法区分节点是否属于同个链，导致地址池互相污染，节点通信性能下降，最终造成节点阻塞。
攻击过程如下图：




**(2) 缺乏信任模型机制**

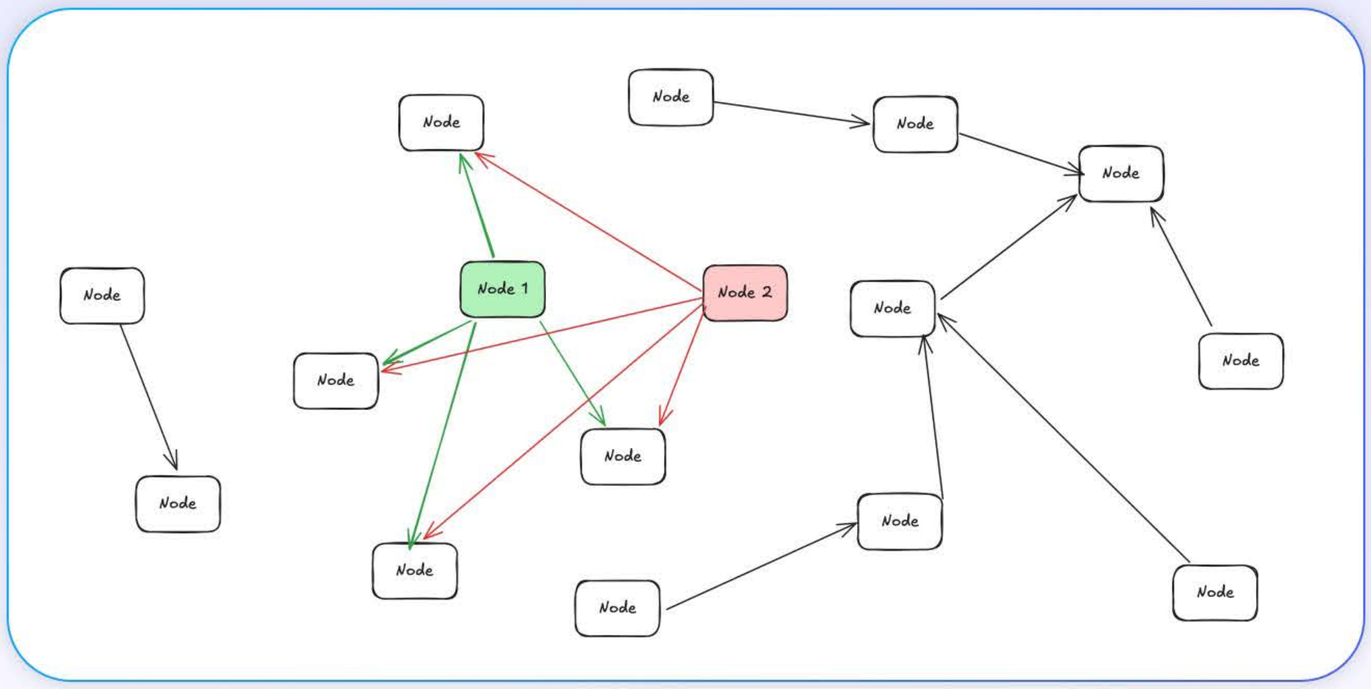
为每个节点建立信誉分数，根据节点的历史行为来调整信任度。例如，如果节点频繁发送无效数据，则降低其信誉。高信誉的节点更受信任，而低信誉的节点需要受到限制。

**(3) 缺乏节点数量限制机制**

限制新节点的创建速度或每个ip的连接数，防止短时间内创建大量虚假节点。

**(4) 节点发现算法问题**

节点发现与选择算法负责在 P2P 网络中定位新节点并建立连接。若算法设计不合理，如距离算法不均衡，容易导致网络拓扑失衡，部分节点过载。需要保证算法的均衡性与不可预测性，以提高节点分布的安全性和网络的抗攻击性。
如下图片展示了网络拓扑失衡的一种极端情况：




**(5) 易受攻击的节点选择机制**

如果节点使用易受攻击的节点选择机制，那么可能其连接的所有其他节点都是恶意节点，导致 Eclipse Attack。

下面是常见的节点选择安全机制：

随机节点选择：将节点连接的目标随机化，使攻击者难以控制节点的连接结构。

局部连接策略：让节点优先与物理上或网络拓扑上较近的节点建立连接，使得攻击者难以渗透整个网络。

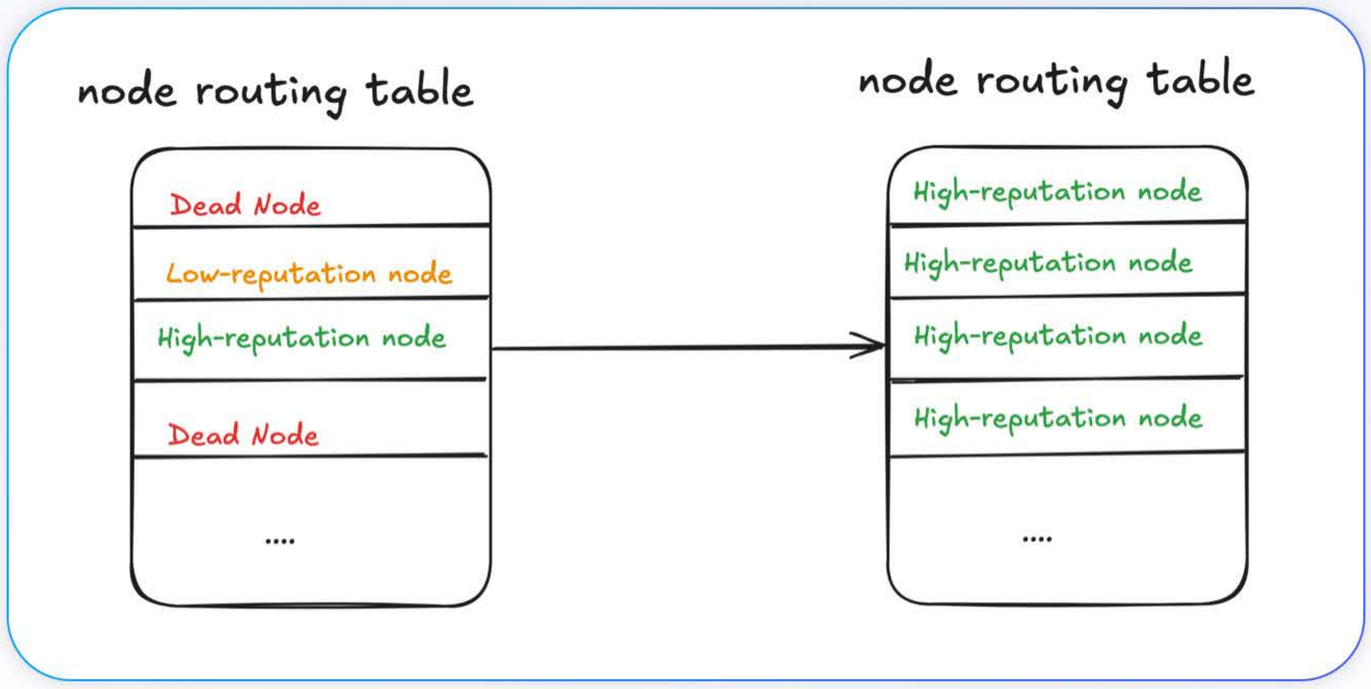
**(6) 缺乏身份验证**

在需要许可的 p2p 网络中，需要对节点的身份进行验证

**(7) 缺乏路由表定期更新机制**

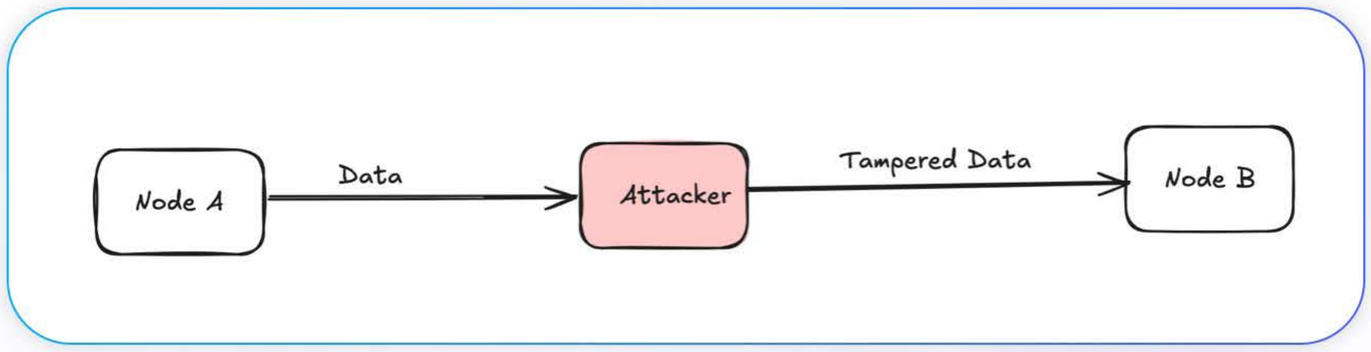
如果发现一个节点返回的数据是非法的，就需要考虑是否删除其在路由表里的记录

节点应该定期地从路由表中移除长时间未活跃的节点，替换成新的节点，从而避免因路由表失效导致的网络孤立。




**(8) 中间人劫持漏洞**

p2p 数据在传输的过程中，必须要保持数据的完整性。如果使用的加密算法不正确或者有漏洞，可能会让数据被篡改。



## 3.6 DoS 漏洞

DoS（拒绝服务）漏洞会导致系统资源耗尽，阻碍合法用户的正常访问，以下是关键类型：

### 内存耗尽攻击

利用大量内存需求拖垮系统，建议通过设置资源上限防范。
一个比较常见的内存耗尽攻击是“zip炸弹”。**zip 炸弹的基本原理是，我们生成一个非常大的内容全是 0（或者其他值）的文件，然后压缩成 zip 文件，由于相同内容的文件的压缩比非常大，此时生成的 zip 文件非常小。被攻击目标在解压 zip 文件之后，需要消耗非常多的内存来存储被解压之后的文件，内存会被快速耗尽，目标因为 OOM 而崩溃。**
其他的压缩算法也会出现同样的问题。下面是压缩1GB大小的内容全部为0的文件，常见算法的压缩比列表：

| File Format | Compressed Size | Compression Ratio |
| --- | --- | --- |
| .bzip2 | 1KB | 1048576:1 |
| .7z | 154KB | 6808:1 |
| .xz | 155KB | 6765:1 |
| .gzip | 1231KB | 851:1 |
| .zip | 1231KB | 851:1 |

Sui 链之前出现过这种漏洞，在 Sui 的 p2p 协议中，为了减少带宽压力，有部分 RPC 消息是用 snappy 算法压缩，代码如下：

```
impl<U: serde::de::DeserializeOwned> Decoder for BcsSnappyDecoder<U> {
    type Item = U;
    type Error = bcs::Error;
fn decode(&mut self, buf: bytes::Bytes) -> Result<Self::Item, Self::Error> {
    let compressed_size = buf.len();
    let mut snappy_decoder = snap::read::FrameDecoder::new(buf.reader());
    let mut bytes = Vec::with_capacity(compressed_size);
//Decompress
    snappy_decoder.read_to_end(&mut bytes)?;
//Deserialize
    bcs::from_bytes(bytes.as_slice())
}
}
```

如果恶意攻击者使用 snappy 压缩炸弹，就可以让节点因为内存耗尽而崩溃。

还有一种漏洞类型就是数据包大小没有限制。如果目标节点在接受数据的时候，没有对接受数据的大小做校验，那么攻击者就可以使用超大数据包占用内存资源。

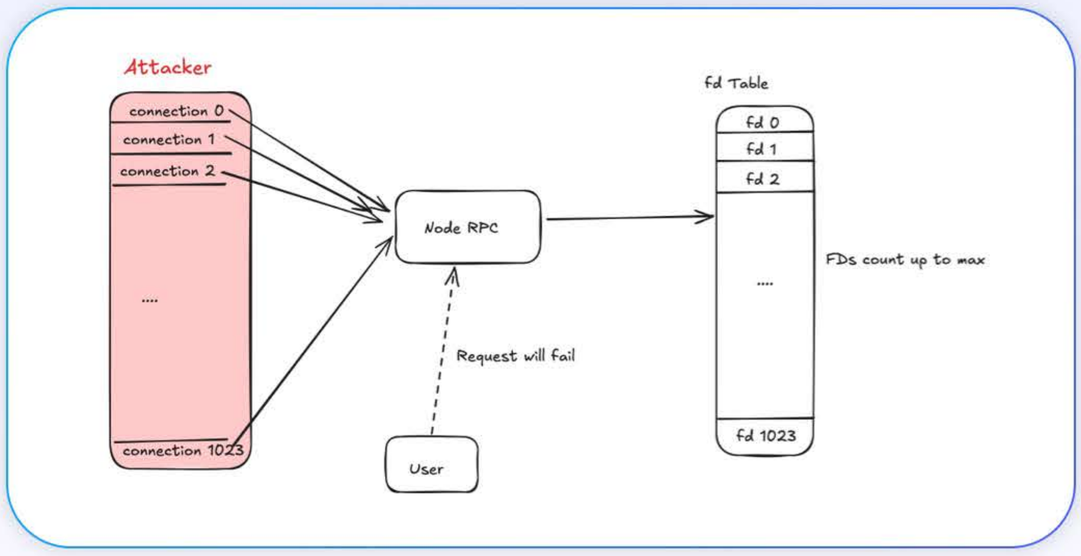
### 硬盘耗尽攻击

写入无用数据占满存储空间，通过磁盘配额管理防止资源不足。
硬盘耗尽攻击常见的有如下2种情况：

- zip 炸弹。攻击方法和上面的“内存耗尽攻击”一样，只不过目标程序将 zip 解压缩到磁盘，而不是直接到内存。
- 无成本或低成本写入大量数据到磁盘，就可以将磁盘数据耗尽

## 内核句柄耗尽攻击

大量资源请求使句柄耗尽，建议控制句柄分配并监控异常。
此攻击的大概原理是：攻击者通过漏洞，使目标节点的内核句柄耗尽或者接近耗尽，让其无法对正常的业务请求进行响应。




常见的一种攻击是“ Socket 压力攻击”。如果目标节点没有限制连接数量和并发数量，那么攻击者可以发送并维持大量连接请求，耗尽系统 Socket 资源。

以下是 Linux 系统常见内核句柄**（也称为内核对象）类型和数量限制**：

1. 文件描述符（File Descriptor）：通过 `ulimit –n` 命令查看和调整当前进程的最大文件描述符数量限制。常见限制是1024。

2. 信号量（Semaphore）：信号量的数量限制通常在系统级别配置，例如 `/proc/sys/kernel/sem` 文件中设置的系统信号量限制。

3. 共享内存段（Shared Memory Segment）：共享内存段数量和大小受到系统参数的限制，可以通过 `/proc/sys/kernel/shmmax` 和 `/proc/sys/kernel/shmall` 等参数进行调整。

4. 消息队列（Message Queue）：消息队列的数量限制可以通过 `/proc/sys/kernel/msgmni` 和 `/proc/sys/kernel/msgmax` 等参数进行配置。

5. 线程和进程句柄（Thread/Process Handle）：进程的总数受到系统限制，通常可以通过 `ulimit –u` 查看和设置单个用户的最大进程数量。

6. 定时器（Timer）：定时器的数量通常依赖于系统资源，没有特别的硬性限制，但进程资源限制会间接影响定时器的数量。

7. 网络套接字（Socket）：网络套接字数量通常受到文件描述符限制的影响，也受系统网络配置（如 `net.core.somaxconn` 和 `net.ipv4.ip_local_port_range`）影响。

## 持续性内存泄露

内存无法正常释放导致资源枯竭，需定期检测内存使用防止问题恶化。
内存泄露在一般情况下不算作安全漏洞。但是如果攻击者可以主动出发目标节点的内存泄露，并能重复这一操作，日积月累，目标节点程序就应为内存不足而崩溃。

## 3.7 密码学漏洞

密码学漏洞会破坏数据的保密性和完整性，给系统带来潜在的安全威胁。以下是主要的密码学漏洞类型：

### 使用已经被证明不安全的哈希算法

哈希算法用于生成数据的唯一标识，确保完整性不被篡改。常见哈希算法如 MD5 和 SHA-1 已被证明存在碰撞风险，可能被恶意攻击者利用。因此，建议使用更安全的 SHA-256 或 SHA-3 等现代哈希算法，并保持算法更新，避免易遭破解的旧算法带来的数据完整性风险。

### 使用不安全的自定义哈希算法

有些项目使用自己定义的哈希算法，一般情况下这些算法的都没有公开的知名算法安全。
比如我们在审计过程中，遇到如下自定义的哈希算法：

```
export function check(key: string) {
  if (key == undefined) return false;
  const decrypt = hashCode(key);
  if (decrypt != CONST_KEY_HASH) {
    return false;
  }
  return true;
}

export const hashCode = (s) => {
  return s.split('').reduce(function (a, b) {
    a = (a << 5) - a + b.charCodeAt(0);
    return a & a;
  }, 0);
};
```

`hashCode` 函数并不是一种加密哈希函数，且极易发生冲突。它仅执行非常简单的按位和加法运算。此外，该函数的输入长度和输出长度遵循明确且可预测的模式，使其很容易被反向解析和破解。由于这种弱哈希机制，攻击者可以轻易生成导致相同 `CONST_KEY_HASH` 值的密钥，从而危及 API 授权过程的安全性。
以下是概念验证 (PoC) 代码，展示了如何利用这种弱哈希的漏洞

```
const generateRandomString = (length: number): string => {
  const characters =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
  let result = '9';
  const charactersLength = characters.length;
  for (let i = 1; i < length; i++) {
    result += characters.charAt(Math.floor(Math.random() *
charactersLength));
  }
  return result;
};

const poc = () => {
  const attempts = 10000000000;
  for (let i = 0; i < attempts; i++) {
    const randomString = generateRandomString(5);
    if (check(randomString)) {
      console.log(`Found valid key: ${randomString}`);
    }
  }
};

poc();
```

## 不安全的使用导致的哈希碰撞

一种常见的情况是，HASH(A+B+C)=HASH(D+E)，出现这种情况的原因是A+B+C=D+E，由于‘A+B+C’和‘D+E'完全一样，但是ABCDE 各自不一样。

## 使用了不安全的数字签名算法

数字签名算法用于验证数据的真实性和来源，防止数据被篡改。早期的签名算法如 DSA 和 RSA 可能在量子计算威胁下失效。使用 ECDSA 或 EdDSA 等现代签名算法可以提供更强的安全性，保护数据的合法性和防伪能力。特别是分布式系统和智能合约中，确保签名算法的可靠性至关重要。

## 使用了不安全的加密算法

加密算法的强度直接影响数据的机密性，弱加密算法（如DES）可能被攻击者轻易破解。建议使用 AES-256 等更高位数的对称加密算法，并在通信过程中实施端到端加密（如 TLS）保障数据的传输安全，防止被窃听或篡改。
此外，确保密钥管理到位，防止密钥泄露。

## 使用了不安全的随机数生成算法

随机数生成器是许多密码学操作的基础，特别是在生成密钥、IV（初始化向量）和非对称加密中的重要参数时，确保其不可预测性至关重要。以下是常见的安全问题：

- 不安全的随机数生成算法导致随机数被预测甚至操控；
- 在公链上的随机数算法基本都存在随机数预测的风险，因为信息全都公开透明所以随机数都可以被预测。
- 不安全的随机数生成算法导致随机数的随机性很差从而提高某些漏洞甚至问题（比如矿工出块）发生的概率。

## 使用了不安全的随机数种子

随机数种子泄露或者可以被暴力破解导致随机数种子泄露进而导致随机数被预测；

## 密码学侧信道攻击

侧信道攻击通过监测系统的物理特征（如功耗、执行时间、缓存等）来获取敏感信息。这种攻击绕过了算法本身的安全性，尤其在硬件设备和嵌入式系统中更为常见。防御手段包括对算法实现进行优化，使执行时间和功耗保持恒定，减少可泄露的特征。此外，通过掩蔽和混淆等技术降低侧信道信息泄露的可能性。

## 签名延展性

签名延展性是指，在不改变签名内容的情况下，能够通过已知的有效签名推导计算出另一个不同的有效签名。这种特性带来的一个显著风险是交易延展性，交易延展性使得恶意用户能够利用不同签名变体进行交易重放，重放的交易因为签名不同所以拥有不同的 hash 值，在交易确认过程中混淆用户对交易的状态判断，从而实现双重支付。

# 3.8 账本安全漏洞

## 交易内存池漏洞

1、交易可以重放
2、失败的交易没有扣除手续费

## 区块哈希碰撞漏洞

区块的构造方式如果有问题，则会产生碰撞。

## 孤儿区块处理逻辑漏洞

对孤块可选择直接丢弃，但若选择缓存，则必须添加如高度，时间等的限制条件。

## 默克尔树哈希碰撞漏洞

默克尔树叶子节点如果构造方式有问题，则会产生碰撞。

## 交易金额处理问题

由于交易金额处理时发生上下界溢出、类型不统一、精度误差、出现负数及因外部条件改变造成的非预期值。

## 交易手续费处理问题

由于交易手续费处理时发生上下界溢出、类型不统一、精度误差、出现负数及因外部条件改变造成的非预期值。

## 区块与交易验证的时间过于敏感

由于不同节点的时间有误差，所以时间验证不能过于敏感，不如容易导致区块分叉。

## 交易的签权逻辑有问题

主要包括以下2个方面：

- 伪造身份绕过
- 权限检查错误

## 3.9 经济学模型漏洞

经济学模型在区块链和分布式系统中起着至关重要的作用，影响着网络的激励机制、治理结构和整体可持续性。
以下是主要的关注点：

### Cosmos 应用链的经济模型（以UniChain为例）

Cosmos 的应用链采用了区块链互操作性和可扩展性为核心的经济模型。以 UniChain 为例，其经济模型设计不仅考虑了链上代币的流通和使用，还考虑了不同应用场景的需求。通过利用 Cosmos SDK，UniChain 能够创建特定于应用的区块链，并实现跨链通信，促进不同链之间的资源共享和价值流动。经济模型需关注代币发行量、通货膨胀率、交易费用结构以及链上治理机制，以确保生态系统的稳定与繁荣。

### 激励机制是否合理

激励机制是经济模型的核心部分，直接影响用户和节点的参与度。合理的激励机制应确保各参与者（如矿工、验证者、开发者和用户）都能获得公平的回报。需要评估激励结构的可持续性，确保其能够有效防止恶意行为和中心化趋势。此外，激励机制还应适应市场变化，随着网络的成熟和用户需求的演变进行调整。例如，是否设有合适的奖励和惩罚机制，如何平衡短期和长期激励等，都是需要深入分析的问题。

### 网络经济学的可持续性

经济模型还需关注网络的长期可持续性，包括对生态系统内各方的经济激励、价值创造及其分配的管理。需要评估是否存在经济不平衡或资源浪费的情况，确保所有参与者都能在网络中获得合理的价值。此外，需分析可能影响网络稳定性的外部因素，如市场波动和用户行为变化。

### 市场反馈机制

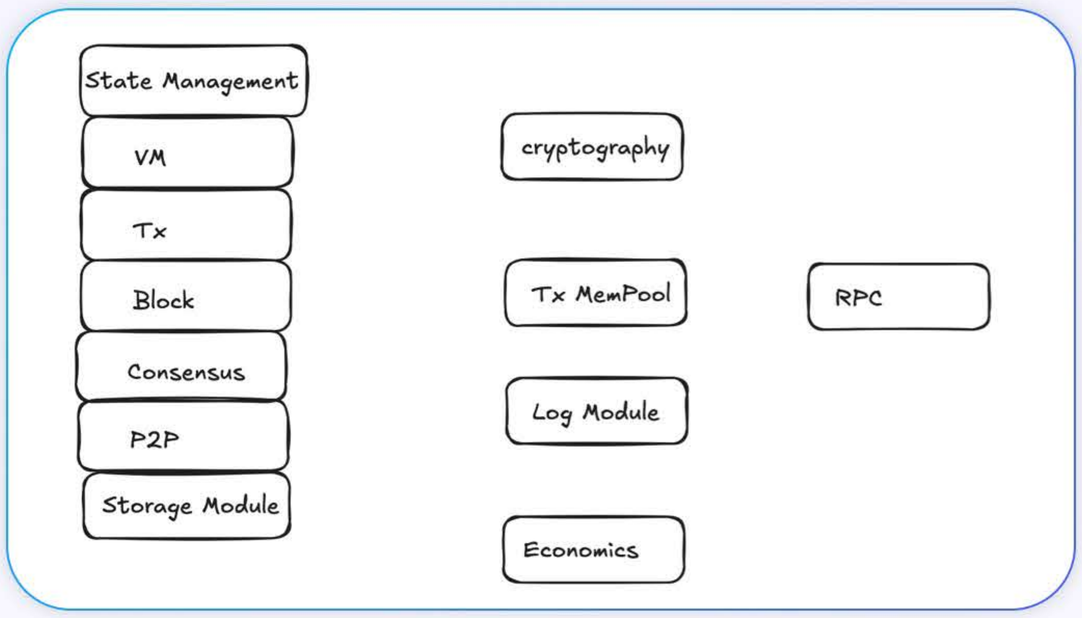
建立有效的市场反馈机制，使经济模型能够根据用户需求和市场动态进行调整。通过定期的数据分析和用户反馈，能够及时识别并修正潜在问题，确保经济模型的灵活性和适应性。

### 治理结构的影响

经济模型的设计应与治理结构相结合，确保用户在治理过程中能够参与经济决策，从而增强社区的参与感和归属感。合理的治理机制能够促进经济模型的自我修复和持续优化，提高网络的整体健康度。

# 4 攻击面列表

以下是常见的攻击面列表：




## 4.1 虚拟机

攻击面：虚拟机负责执行智能合约和处理字节码，通常承载大量复杂逻辑，存在重入攻击、整数溢出和内存溢出等潜在风险。
此外，高计算消耗的智能合约可能引发 DoS 攻击，导致资源耗尽。此外，智能合约的字节码若包含未审计的漏洞，容易引发任意代码执行和权限提升等风险。

## 4.2 P2P 节点发现与数据同步模块

攻击面：P2P 节点的发现和同步功能若设计不当，容易遭遇 女巫攻击（Sybil Attack），通过伪造大量假节点来控制网络，导致网络性能下降甚至失效。节点数据同步过程中的 路由表污染 会影响节点间的连接质量，使得部分节点不可用。此外，数据包中可能存在伪造或恶意数据，导致节点接收错误信息，进而影响同步和共识。

## 4.3 区块解析模块

攻击面：区块解析涉及大量数据处理，若解析代码中存在溢出或错误处理，可能被恶意区块攻击，导致服务崩溃或拒绝服务（DoS）。此外，不正确的区块格式校验可能导致网络传输的区块被篡改而无法识别，从而影响全网一致性。

## 4.4 交易解析模块

攻击面：交易解析涉及验证交易结构和签名，若对伪造的交易格式、恶意数据或异常签名处理不当，可能导致虚假交易通过，消耗系统资源。此外，交易解析中若存在边界溢出问题，也可能被利用执行内存注入攻击。

## 4.5 交易内存池

攻击面：内存池是交易入链前的临时存储区，可能被滥用插入大量无效交易或恶意交易，导致 内存占用攻击，使节点无法响应正常请求。此外，恶意攻击者可以利用交易内存池插入重复或高频交易，进一步造成 资源枯竭 和 DoS 风险。

## 4.6 共识协议模块

攻击面：共识机制设计不完善或被操控时，可能遭遇 双花攻击、自私挖矿、51%攻击 等问题。攻击者可以通过控制超过一半的计算能力执行恶意分叉，影响交易记录的合法性。
此外，某些共识机制在面对高延迟或分区网络时，可能会因缺乏明确的容错机制而失效。

## 4.7 RPC 接口

攻击面：RPC 接口是外部与区块链节点交互的途径，如果访问权限配置不当，可能导致未经授权的访问和数据泄露。特权 RPC 接口若未妥善保护，攻击者可以伪造请求执行高权限操作，进一步操控链上数据。
此外，RPC 接口易遭遇请求洪水攻击，导致节点响应过载。

## 4.8 日志处理模块

攻击面：日志模块负责记录系统运行的详细信息，若攻击者能通过日志注入写入伪造日志内容，可能导致敏感信息泄露。过度记录的日志也可能被恶意利用造成 日志膨胀，导致存储资源消耗甚至系统不可用。

## 4.9 网络中间件

攻击面：区块链网络通信中间件如果没有加密传输和身份认证机制，可能遭遇 中间人攻击（MITM），导致数据包截获和篡改。
此外，中间件易受流量攻击（如 DoS）和协议滥用攻击，影响整个网络的正常通信。

## 4.10 加密算法

攻击面：加密算法的设计和实现直接关系到数据的安全性。若存在哈希碰撞漏洞，可能导致交易内容被伪造；加密算法若未遵循强随机性原则，可能使得密钥泄露。其他常见攻击还包括 侧信道攻击，如通过观察加密执行过程中的能耗或电磁泄露获取敏感信息。

## 4.11 经济学模型

攻击面：区块链的经济模型设计需平衡各方激励，否则可能导致攻击者通过操控代币流通、降低矿工奖励等方式影响系统稳定性。
经济模型的不合理激励设计可能导致矿工（或验证节点）不按预期行为执行，带来 经济性攻击 风险。

## 4.12 数据存储模块

攻击面：链上数据和链下数据存储存在未经授权访问、篡改和数据持久性风险。攻击者可以尝试利用数据库权限不足或不安全的存储机制，直接修改账本数据或智能合约状态。此外，不合理的数据存储策略可能导致数据膨胀，影响系统性能。

## 4.13 状态管理模块

攻击面：区块链的状态管理用于记录账户余额、合约存储等关键数据。如果状态管理模块设计不当，可能被攻击者利用，造成账户余额错误或状态信息篡改。恶意攻击者也可通过构造特殊交易导致状态劫持，造成资源锁死。

# 5 历史安全事件

## 5.1 Move 生态中的安全事件

2023年6月，Move 虚拟机被发现严重拒绝服务漏洞，可以导致Sui、Aptos 等公链全网崩溃，甚至可能硬分叉。
安全研究员 poetyellow 在发现该漏洞后，公布了相关细节。不过Move 虚拟机开发团队内部之前也独立发现了该漏洞，并花费1个多月时间来修复此漏洞。

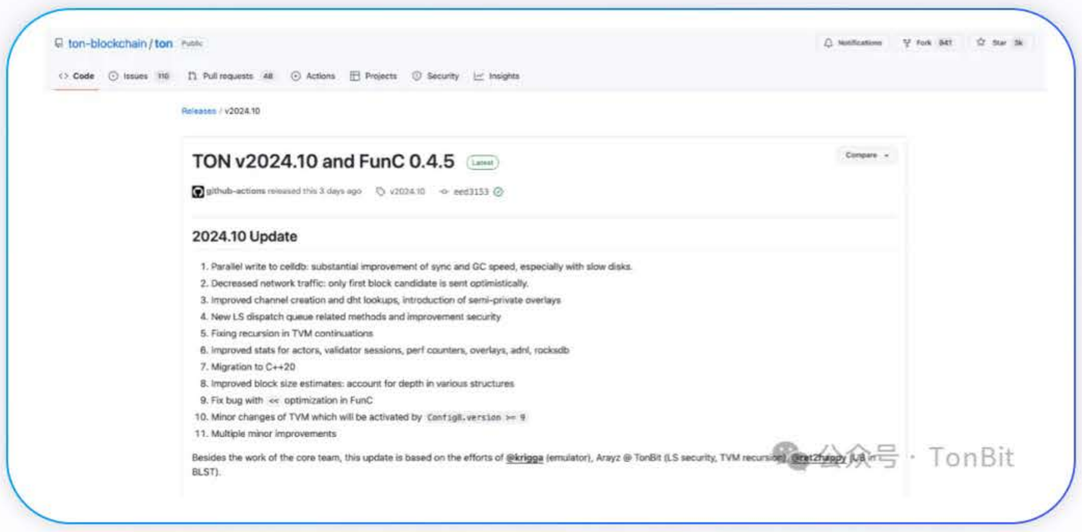
图片来源:https://www.bankless.com/sui-vs-aptos

这个漏洞的类型是无限递归漏洞。在编程语言里面，函数无限递归调用导致的栈溢出，是一个通用DoS漏洞类型，即使安全的Rust语言也逃不掉。

2024年9月，MoveBit 成功发现并协助修复了 Aptos 网络中的一个内存池DoS漏洞，定级为High。
该漏洞因内存池交易驱逐机制不完善，可能导致多达90%的正常交易被节点拒绝。Aptos 团队已在 v1.19.1 版本中修复了该漏洞，并在官方发布说明中感谢 MoveBit 的贡献。

## 5.2 TON 生态中的安全事件

近日，TON 官方团队在其最新版本更新说明中，正式致谢 BitsLab 旗下的安全团队 TonBit 对于 TON 虚拟机中关键漏洞的发现工作。该漏洞若被恶意利用，可能导致虚拟机资源耗尽、系统崩溃，进而影响整个 TON 网络的稳定性。
TonBit 团队凭借深厚的技术实力，快速定位问题并提出有效解决方案，为 TON 虚拟机构建了更为安全的运行环境，进一步增强了TON 生态系统的整体稳定性。




该漏洞的根本原因在于 TON 虚拟机在处理合约延续（continuations）时的嵌套操作设计存在风险。恶意合约可以通过创建深度嵌套的延续结构，引发递归评估过程，从而耗尽虚拟机的宿主栈空间。这种资源耗尽攻击可能导致 TON 虚拟机异常崩溃，简单来说就是不用一个 TON，就可以导致所有的 Validator 宕机，直接影响系统的可用性。

TonBit 团队经过深入分析，和 Ton Core 协作提出了创新的解决方案，该方案可以调整虚拟机的内部跳转机制，以迭代方式替代递归调用，可以有效防止此类攻击的发生。
该解决方案已在 TON 最新版本中得到应用，为 TON 用户提供了更安全、稳定的操作体验。

## 5.3 比特币拓展生态的安全事件

2023年10月比特币的扩容技术——闪电网络被发现存在一个潜在的安全漏洞。开发者 Antoine Riard 在发现该漏洞后，公布了相关细节。


这一漏洞被称为「替代循环攻击（replacement cycling attacks）」，可能会危及通过闪电网络流动的资金安全，导致交易延迟或无法按预期进行处理，从而可能引发比特币闪电网络通道内的资金损失风险。

## 5.4 应用链生态安全

10 月 16 日，Cosmos 生态软件开发公司 All in Bits 披露了 Cosmos Hub 流动性质押模块（LSM）存在严重安全问题的根源。调查发现，大部分 LSM 代码由与朝鲜有关联的开发者编写。

据 All in Bits 报告，LSM 的核心问题包括：1）允许规避罚没的设计缺陷仍然存在；2）LSM 并非独立模块，而是对现有质押、分配和罚没模块的一系列修改，可能影响所有质押的 ATOM；3）超过 19 个月的代码更改未经审计；4）项目负责人 Zaki Manian 和 Iqlusion 公司存在重大信息误导；5）Interchain Foundation（ICF）、Stride Labs 以及 Informal Systems 在项目推进过程中缺乏透明度。

调查显示，LSM 的开发始于 2021 年 8 月，由 Zaki Manian 和 Iqlusion 主导。然而，大部分代码实际由后来被确认与朝鲜有关联的开发者 Jun Kai 和 Sarawut Sanit 编写。尽管 2022 年 7 月的 Oak Security 审计报告指出了关键漏洞，特别是与罚没规避相关的问题，但这些漏洞并未得到充分解决。此外，Zaki Manian 在 2023 年 3 月得知开发者与朝鲜的关联后，未向 Cosmos 社区披露这一重要信息。相反，他在 2023 年 4 月继续推动 LSM 的信号提案，声称模块已“完成”，这一行为被 All in Bits 认定为重大失实陈述和严重疏忽。

All in Bits 建议采取以下紧急措施：1）立即修复 LSM 的主要质押漏洞；2）对 LSM 进行全面、即时的安全审计；3）全面披露朝鲜特工参与的调查时间表；4）将相关 ICF 方列入黑名单；5）针对 ICF 资助项目制定新的审计和监督协议。

# 6 链开发的最佳安全实践

区块链系统的开发要求在多个方面保证安全，涵盖数据处理、共识机制、加密、资源管理等模块，旨在保护链上数据完整性、交易有效性及系统的稳定性和抗攻击能力。

## 6.1 区块

### 区块同步过程

确保节点间的区块同步过程在网络层面和数据层面的一致性，避免因同步问题导致区块数据不一致，确保在不同网络条件下节点仍能有效同步。

### 区块格式解析过程

完善区块格式解析逻辑，确保能对格式异常或畸形数据有效检测，避免因解析漏洞导致崩溃或数据泄漏。
此过程包括对字段长度、数据类型及有效值的验证。

### 区块生成过程

包括合理的 Merkle tree root 构建方式，确保哈希树结构符合数据完整性要求。
同时要求对生成过程中的随机数和签名有严格控制，避免重放攻击或恶意数据注入。

### 区块校验过程

对区块校验包括对区块签名、交易内容、区块头信息等进行综合验证。校验逻辑需足够健壮，以防止伪造签名或恶意篡改数据。

### 区块确认逻辑

确保区块确认机制可以及时检测和处理链上区块的异常情况，保证确认过程足够严谨，防止双花攻击及分叉带来的影响。

### 区块哈希碰撞

对区块哈希算法设计及其碰撞检测的处理机制进行评估，避免恶意制造哈希碰撞带来的安全风险，同时确保哈希函数的不可预测性和随机性。

### 区块处理资源限制

评估区块处理资源，包括孤儿区块池大小、计算量、存储消耗等，以避免资源被恶意占用。限制计算频率、硬盘读写等，以应对可能的 DoS 攻击。

## 6.2 交易

### 交易同步过程

确保交易同步过程中数据完整性与一致性，避免交易丢失或伪造。交易同步需适应不同网络环境，保证数据可靠传输。

### 交易哈希碰撞

检查交易哈希算法的碰撞抵御性，确保不会因恶意构造交易而使系统误判为重复交易。此过程需涵盖对交易哈希生成机制的验证。

### 交易格式解析

完整的格式解析包括字段长度、类型验证，避免因格式错误导致异常或崩溃。解析逻辑需对输入数据进行过滤，确保安全。

### 交易合法性校验

校验各类交易签名、内容结构，防止恶意或畸形交易进入系统。包括对签名、输入输出的合法性验证，以及逻辑正确性校验。

### 交易处理资源限制

通过设置交易池大小、处理频率等限制，避免系统因高负载交易而被过度占用。确保计算、内存和硬盘寻址等资源消耗在合理范围内。

### 交易延展性攻击

确保交易结构不可轻易被更改，以防攻击者通过修改交易内部字段（如ScriptSig）来重新生成交易hash，但不影响交易的实际内容。

### 交易重放攻击

实现防重放机制，通过 nonce 或时间戳等方式防止恶意交易的多次广播和验证。为每个交易引入唯一性标识，以避免重复处理。

## 6.3 合约虚拟机

### 合约字节码校验

对字节码进行严格的格式、语法校验，确保没有潜在的代码执行漏洞。此过程应当检测合约输入的安全性，防止异常代码执行。

### 合约字节码执行

确保合约的执行过程稳定，尤其在异常情况或资源消耗方面的管控。需设置安全沙箱环境避免代码逃逸。

### Gas 模型

确保各项原子操作的 gas 成本与其实际资源消耗相匹配，避免因资源消耗不平衡而导致 DoS 攻击。
通过测试确定每种操作的资源代价，以避免被恶意利用。

## 6.4 日志系统

### 日志记录的完整性和安全性

对字节码进行严格的格式、语法校验，确保没有潜在的代码执行漏洞。此过程应当检测合约输入的安全性，防止异常代码执行。

### 日志上传过程中的敏感信息泄露

确保上传过程中敏感数据脱敏处理，以防止重要信息泄露至外部。

### 日志服务的安全性

避免日志服务在不安全的环境中暴露，确保日志记录仅供内部使用，防止未经授权的访问。

### 日志存储资源管理

控制日志存储占用的内存和硬盘资源，通过定期清理、压缩存储来优化资源使用，避免日志积累导致系统崩溃。

## 6.5 RPC 接口

### 传统安全

参照传统 Web2 安全实践，确保接口访问受限于授权用户，避免恶意访问。
包括访问控制、身份验证、重放保护、防止 SQL 注入等措施。

## 6.6 P2P 协议

### 节点发现和连接建立

确保节点发现和连接的可靠性和安全性，防止节点伪造、数据泄露或中间人攻击。支持端到端加密，并确保节点身份验证机制安全。

### 数据传输的完整性和抗干扰性

通过加密和数据校验来确保传输数据不被篡改，防止恶意节点的攻击或数据注入，保障数据传输的完整性。

## 6.7 密码学

### 加密算法选择

使用经过验证的、抗量子计算的加密算法，避免过时或易破解算法。
特别在随机数生成、签名验证等方面确保采用安全的密码学方法。

### 随机数生成的安全性

采用高质量的随机数生成器，避免因随机数预测导致的安全问题。可以采用硬件生成器或真随机数方案，确保安全性。

## 6.8 加密传输

### 传输协议加密

使用 TLS 或其他安全协议保证传输过程不被窃听或中间人攻击。验证各节点证书的有效性，防止伪造节点参与网络。

## 6.9 模糊测试

### Fuzzing 覆盖范围

针对关键模块如区块解析、交易处理、智能合约执行等实施 Fuzzing 测试，检测潜在的崩溃、边界值错误或异常输入情况。

### 输入处理的稳健性

通过 Fuzzing 检测系统对恶意或异常输入的反应，确保能够处理或忽略这些输入而不影响系统的正常运行。

## 6.10 静态代码分析

### 开源安全工具的使用

使用静态代码分析工具（如SonarQube）进行代码扫描，检测潜在漏洞。特别关注合约代码、输入验证等关键代码路径。

### 代码审查流程

定期进行手动代码审查，通过自动和人工结合的方式，确保代码质量和安全性。
可建立代码审查制度，设定不同层级的代码审查机制。

## 6.11 第三方安全审计

### 独立审计流程

委托具备资质的第三方安全审计机构进行全面安全评估，涵盖代码、架构和逻辑设计等方面，确保系统符合行业安全标准。

### 持续安全评估

定期开展审计，尤其在系统更新或升级后再次进行安全审查，确保新特性引入没有带来新的安全隐患。

# 7 新兴生态公链安全展望

区块链系统的开发要求在多个方面保证安全，涵盖数据处理、共识机制、加密、资源管理等模块，旨在保护链上数据完整性、交易有效性及系统的稳定性和抗攻击能力。

## 7.1 Move 生态的安全观察

Move 语言的出现为区块链生态系统提供了全新的智能合约编程方式，主要应用在 Aptos 和 Sui 等公链上。Move 语言的设计初衷便是以安全性为核心，通过其资源管理、静态类型系统和内存管理来预防常见漏洞。然而，随着生态的不断扩展，Move 仍需关注特定的安全领域：

资源管理和状态一致性：Move 独特的资源类型允许开发者在合约中明确管理资产的所有权，这虽然减少了资产丢失或重入攻击的风险，但复杂的资源转移和管理逻辑可能带来新的错误。确保资源生命周期管理的有效性、避免资源转移漏洞，是关键。

权限控制与访问管理：Move 生态的模块化开发便于组件复用，但模块的访问权限控制至关重要。开发者应严格限制敏感操作的权限，确保模块功能和访问级别的合理性，避免攻击者利用高权限合约模块进行操作。

安全审计和代码验证：Move 代码的复杂性增加了审计的难度，需要持续进行安全审计和形式化验证，确保代码不包含溢出、逻辑错误等常见风险。标准化的审计流程和定期的代码回溯有助于确保 Move 生态的长期安全。

## 7.2 TON 生态的安全展望

TON 生态系统在扩展去中心化应用（dApps）和基础设施方面迅速发展，但由于其独特的架构和功能，TON 面临一些独特的安全挑战。以下是对 TON 生态开发者的安全建议和最佳实践：

节点分布和防护：TON 使用了分片和分布式哈希表（DHT）技术以提高网络扩展性，但若节点分布不均衡或缺乏足够保护，可能导致恶意节点在网络中占据主导地位，进行路由表污染或网络分区攻击。开发者应增强节点验证机制，并通过增加节点监控和黑名单机制提升网络防御能力。

智能合约的安全性：TON 的智能合约编程与其他公链不同，合约逻辑较为复杂。开发者应严格遵循安全开发最佳实践，注重代码的资源管理和边界检查，避免常见的合约漏洞。对合约进行代码审计和定期回顾，使用合约测试工具能提升代码可靠性。

数据完整性和防篡改：TON 的分布式存储增加了数据共享和访问的便捷性，但也带来了篡改风险。开发者可引入多层次的数据加密和认证机制，并在节点之间加入数据一致性验证，确保数据传输的完整性。

## 7.3 比特币拓展生态安全展望

比特币拓展生态旨在解决主链的交易吞吐问题，同时保证安全性和去中心化。

离链交易的信任模型：比特币的拓展生态通过离链技术来提升交易速度，开发者需确保离链交易的信任机制足够可靠。例如闪电网络中的双向支付通道需要使用多重签名技术，并确保通道的关闭过程安全，以防资金冻结或丢失。

隐私性与透明度：闪电网络的通道交易可在不公开的情况下完成，虽然提升了隐私性，但也增加了监管难度，潜在引发恶意行为。Layer 2 网络需在隐私性和透明度之间取得平衡，通过可选择性地公开部分交易记录，增强合规性。

用户体验与安全性：拓展生态的复杂性带来了用户操作难度，如闪电网络的通道管理可能对普通用户不友好，增加了操作失误风险。比特币拓展生态可通过设计更友好的界面和操作简化工具，提升用户体验，减少安全隐患。

# 8 About BitsLab

BitsLab 是一家致力于守护和构建新兴 Web3 生态系统的安全组织，愿景是成为备受行业和用户尊敬的 Web3 安全机构。旗下拥有三个子品牌：MoveBit、ScaleBit 和 TonBit。BitsLab 专注于新兴生态系统的基础设施开发与安全审计，覆盖但不限于 Sui、Aptos、TON、Linea、BNB Chain、Soneium、Starknet、Movement、Monad、Internet Computer 和 Solana 等生态。同时，BitsLab 在审计多种编程语言方面展现了深厚的专业能力，包括 Circom、Halo2、Move、Cairo、Tact、FunC、Vyper 和 Solidity。

作为区块链安全领域的领先者，BitsLab 已为 Movement、Aptos Framework、Catizen、Synthetix、Tether、Cetus、UniSat、Nervos CKB、iZUMI Finance 和 Pontem 等多个旗舰项目提供安全审计服务。迄今为止，BitsLab 已交付超过 400 项安全解决方案，审计了 40 万行代码，保护了价值 80 亿美元以上的资产，为全球超过 200 万用户提供了安全保障。这些成就充分体现了 BitsLab 对高质量审计服务的承诺，并树立了区块链行业的安全标准。

此外，BitsLab 团队汇聚了多位顶级漏洞研究专家，他们曾多次荣获国际 CTF 奖项，并在 TON、Aptos、Sui、Nervos、OKX 和 Cosmos 等知名项目中发现了关键漏洞。BitsLab 将继续深耕 Web3 安全领域，助力新兴生态系统的健康发展。

bits lab

# Contact US

## BitsLab 官方渠道信息

https://bitslab.xyz/

https://x.com/0xbitslab

https://t.me/BitsLabHQ

## BitsLab子品牌官方网站

MoveBit：https://www.movebit.xyz/

TonBit：https://www.tonbit.xyz/

ScaleBit： https://www.scalebit.xyz/

## BitsLab品牌资源

https://somber-throne-617.notion.site/BitsLab-Brand-Assets-12e7c2e0096880e58c9fd49f0852f49b

## Telegram 审计需求联系

@starchou

免责声明： 本报告仅为信息提供之用，不构成任何形式的投资建议、推荐或担保。读者在做出投资决策前应自行进行独立调查和分析。

数据准确性： 本报告中的数据已尽可能准确和最新，但由于市场的快速变化，某些数据可能在报告发布后发生变化。

报告局限性： 由于信息获取的限制，本报告可能无法涵盖所有安全事件和市场变化。读者应结合其他来源进行综合判断。



安全提示： 用户应提高自身安全意识，定期更新安全软件和防范措施，避免成为网络攻击和诈骗的受害者。

参考文献： Aptos、Sui等项目方官方网站和技术白皮书。

Move编程语言的相关技术

https://github.com/allinbits/announcements/blob/main/2024_10_15_lsmnk.md

https://cryptorubic.medium.com/bitcoin-layer-2-what-is-it-must-know-projects-in-2024-7f775acebcc7