Software Engineering and Applications 软件工程与应用, 2021, 10(6), 809-822
Published Online December 2021 in Hans. http://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2021.106085

Hans汉斯

开源社区中区块链项目的代码漏洞分析

程　伟[1]，陈克豪[1]，丁　荪[2]，刘　杨[1,3*]

[1]浙江理工大学，信息学院，浙江　杭州
[2]Scantist，新加坡
[3]新加坡南洋理工大学，新加坡

收稿日期：2021年11月26日；录用日期：2021年12月21日；发布日期：2021年12月29日

摘　要

随着近年来关于区块链领域的研究探索不断加深，例如数字货币，商品溯源等基于区块链技术的各种应用已经深入我们的生活，与此同时，区块链网络中的漏洞也在不断被人挖掘，即使区块链技术理论本身是安全的，但是大部分应用都是研发人员基于区块链技术之上进行的软件系统设计。我们很难保证研发人员在设计软件时是否会引入第三包组件，由第三方组件产生的漏洞，是导致软件出现问题的重要原因之一。本文将关注开源社区中与交易场景相关的区块链项目的漏洞情况，基于Scantist SCA对项目的漏洞情况进行扫描并分析。首先，为了能够获取到符合主题的项目，本文通过网络爬虫的方式在GitHub社区中进行项目检索区块链项目，并通过五个维度建立筛选模型，最终确定待扫描项目。然后，本文针对Scantist SCA进行指标分析，来了解SCA工具在漏洞检测中有哪些能力，分析SCA工具通过哪些指标来帮助分析漏洞。当掌握了SCA在漏洞分析中的帮助后，对交易场景下的区块链项目中，第三方组件漏洞现状进行了一个评估，主要对该主题下社区中项目平均CVE安全漏洞数，安全等级分布，公共漏洞情况三个方面进行分析，最终得出该主题下的普遍情况。其次，在分析了漏洞情况后，为了解决这些漏洞带来的影响，本文根据Scantist SCA给予的漏洞修复方法进行尝试修复。在修复后，对项目的安全性，修复兼容性问题做出探讨。最后，本文经过上述的分析，对现有的漏洞问题进行讨论，对已经开发完成的代码给出检测方案，对开发初期或中期的项目，提出安全防护左移策略方案。

关键词

区块链，交易场景，SCA工具，漏洞修复

Analysis of Code Vulnerabilities of Blockchain Projects in the Open Source Communities

Wei Cheng[1], Kehao Chen[1], Sun Ding[2], Yang Liu[1,3*]

[*]通讯作者。

文章引用: 程伟, 陈克豪, 丁荪, 刘杨. 开源社区中区块链项目的代码漏洞分析[J]. 软件工程与应用, 2021, 10(6): 809-822.
DOI: 10.12677/sea.2021.106085

[1]School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou Zhejiang
[2]Scantist, Singapore
[3]Nanyang Technological University, Singapore

Received: Nov. 26th, 2021; accepted: Dec. 21st, 2021; published: Dec. 29th, 2021

Abstract

With the deepening of research and exploration in the field of blockchain in recent years, various applications based on the blockchain technology such as digital currency and commodity traceability have penetrated into our lives. At the same time, the loopholes in the blockchain network are also constantly mining by people, even if the blockchain technology theory itself is safe, most of the applications are based on the blockchain technology to carry out the software system design by developers. It is difficult for us to guarantee whether developers will introduce the third package of components when designing software. Vulnerabilities generated by third-party components are one of the important reasons for software problems. This article will focus on the vulnerability of blockchain projects related to transaction scenarios in the open source communities, and scanning as well as analyzing the vulnerabilities situations of the project based on Scantist SCA. First of all, to be able to obtain projects that meet the theme, this article uses a web crawler to search for blockchain projects in the GitHub community, and establish a screening model through five dimensions to finally determine the project to be scanned. Then, this article analyzes the indicators of Scantist SCA to understand the capabilities of the SCA tools in vulnerability detection, and analyzes which indicators the SCA tool uses to help analyze the vulnerabilities. After mastering the help of SCA in vulnerability analysis, an assessment was made on the current situation of third-party component vulnerabilities in blockchain projects in transaction scenarios, mainly the average number of CVE security vulnerabilities in the community projects under this topic, and the distribution of security levels, and the public vulnerability situation. Analyze the three aspects, and finally get the general situation under the theme. Secondly, after analyzing the vulnerability situations, in order to solve the impact of these vulnerabilities, this article attempts to repair according to the vulnerability fix method given by Scantist SCA. After repairing, discuss the safety of the project and repair compatibility issues. Finally, after the above analysis, this article discusses existing vulnerabilities, provides a detection plan for the developed code, and proposes a left-shift strategy for security protection for projects in the early or mid-term development.

Keywords

Blockchain, Transaction Scenarios, SCA Tools, Vulnerability Fix

1. 引言

当谈到区块链的时候，往往会涉及几个关键词：“不可伪造”、“全程留痕”、“可以追溯”、“公开透明”、“集体维护”等，给人的第一印象就是安全。区块链的各个区块之间通过随机的哈希算法实现连接，后一个区块包含前一个区块的哈希值，随着信息交流扩大，一个区块与一个区块相连，形成的结果就是区块链。根据区块链的51%算力攻击法则，只要不能掌握全部区块数据节点的51%，就无法随

意操控和修改网络数据，这使得区块链本身变得相对安全，避免了主观、人为的数据变更。这也就是为什么我们说区块链相对安全的原因。我们从理论上解决了区块链的安全问题。那与区块链相关或者基于区块链技术[1]进行开发的项目就真的安全了吗？解决了区块链底层技术的安全性问题后，区块链项目安全问题就回到了软件系统设计自身的层面了。

生活中大家时刻与软件系统打交道，无论是普通用户，技术研发人员，学术研究人员，都在使用着软件系统。计算机软件技术发展给我们带来便利的同时，也带来了许多安全隐患。大多数时候，安全隐患问题更多的是交由学术研究者和技术研发人员来控制。这些安全隐患对于大多数普通用户来说，往往并没有那么容易感知，但是如果在软件设计过程中研发人员忽略了安全隐患带来的风险，引入了漏洞，那么软件漏洞带来的危害(如个人隐私信息、财产安全等)将由用户承担。各种网络安全事件频发，严重危害了网络的进一步发展和用户对其的信任度。软件系统作为互联网时代的重要组成部分，其本身在编码时期由于开发和设计不良引起的问题所产生的漏洞是导致不法分子攻击的根本原因。

本文针对开源社区(GitHub 社区)中交易场景下的区块链相关的项目的漏洞情况展开分析研究。基于Scantist SCA (Software Composition Analysis)对项目的漏洞情况、漏洞修复问题进行检测。

2. 研究问题

根据上述引言可知，本研究的目的是通过针对交易场景下的区块链项目[2]进行分析。基于 SCA 工具，将开源社区中漏洞情况进行分析，帮助开发人员在进行相关开发时，提高安全编码的意识，更好的对项目进行漏洞分析及修复。

我们提出以下研究问题：

1) 区块链项目在交易场景下的第三方组件漏洞现状如何？

2) 第三方组件漏洞修复可达性如何？

3. 数据收集与筛选

3.1. 数据收集

数据收集是利用 Python 中的 request 库，编写 GitHub 通用爬虫程序。其主要工作原理是：请求访问 GitHub 的 API。通过 API 获取到每个项目的 URL 链接，再进行内容解析，通过解析结果将数据存入 CSV 中，待后续进行筛选。爬虫算法如图 1 所示。

通用爬虫程序编码完成后，对 GitHub 进行关键词检索，由于 GitHub 中没有对交易场景下的区块链项目进行专门分类，所以数据集的获取需要自行采集。通过尝试阶段对区块链相关关键词进行检索。将检索后的结果，进行人工筛选出有价值的，符合研究目标的关键词进行重新检索，如 Bitcoin Payment，Ethereum Transaction 等。在此基础上我们一共收集了 3290 个项目，等待二轮的数据筛选。

3.2. 数据筛选

3.2.1. 需要筛选的原因

在数据收集阶段，我们爬取了与区块链相关的项目 3290 个。由于区块链本身的项目数量十分庞大，并且语言繁杂，这给代码安全分析带来了很大的挑战，目前大部分 SCA 工具支持的都是主流语言，所以一些小众的语言并不适合代码安全检测。我们在筛选的时候，选取主流且具有代表性的语言 Java 和 JavaScript 作为项目扫描的主要对象。除了主流语言的筛选之外。主题的选择十分重要，区块链的应用场景十分广泛，如贸易，物流，文娱，金融交易，社交，知识产权，工业等场景，其应用成熟度也是不一样的。在金融交易场景下的应用成熟度是目前来说最高的。所以我们选择对金融交易相关场景进行开源

项目的代码安全分析。筛选过程需要对一些主题不适合的区块链项目进行剔除，我们选择通过手工筛选的方式对 3290 个项目进行二轮筛选。最后保留下来了 153 个项目。

3.2.2. 手工筛选的意义

交易场景下的项目的定义比较宽泛，而且 GitHub 社区并没有单独为交易场景建立明确的主题，所以需要我们制定一些标准，对较为符合我们主题的项目进行保留。手工筛选的过程能够让项目的匹配度更加契合主题，使最终代码安全的分析更加符合分析的领域。有些项目由于各种原因，在 GitHub 上已经弃用、停止维护，显然这些项目是不符合我们的需求的。

在筛选过程中通过项目主题契合度、是否适合扫描、维护程度、流行度、活跃程度五个维度建立筛选模型，如图 2 所示。

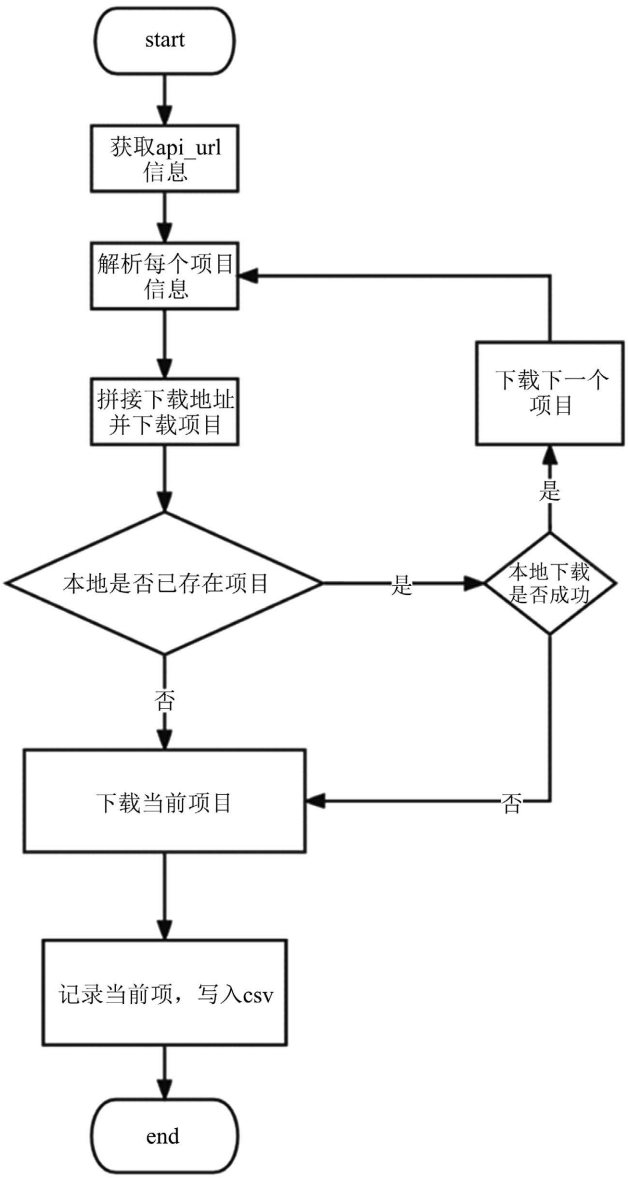

Figure 1. The crawler algorithm implementation process
图 1. 爬虫算法实现流程

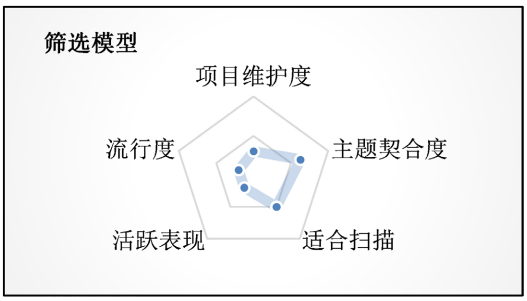

Figure 2. Screening model
图 2. 筛选模型

在通过建立检索模型后进行手工筛选，我们发现一些类型与交易场景息息相关。例如在 Bitcoin Payment 中：实时加密货币价格跟踪器、比特币电子钱包、比特币支付网关、支付监听器、比特币支付处理系统、比特币现金和莱特币支付机器人、密钥管理服务等项目。又例如 Ethereum Transaction 中：以太坊交易管理器、以太坊支付二维码生成器、以太坊交易跟踪库等项目[3]，与交易场景密切相关的项目是符合作为候选条件的。

4. 研究问题与结果分析

在研究问题之前，我们对 SCA 工具做了一个分析，来了解 SCA 工具在漏洞检测中有哪些能力，在 4.1 中我们分析了以 Scantist SCA 为例，分析了 SCA 工具通过哪些指标来帮助分析漏洞的情况。4.2~4.4 中我们对研究问题进行了探究。

4.1. SCA 工具指标分析

不同的 SCA 工具对漏洞扫描结果有着不同的呈现方式，只是侧重点会有所不同，并且增加除扫描以外的辅助功能，以帮助漏洞修复。

Scantist SCA [4]提供了多个指标来帮助对依赖项中的漏洞进行风险评估，我们根据扫描报告，将这些指标分为四类，以帮助进行评估。

1) 基于代码分析的指标：Scantist SCA 工具可以分析源代码或二进制文件及代码片段，以推断依赖项的使用情况，可应用于编译与未编译的项目。

a) 对于未编译的项目会进行 Air Gap 扫描(预测扫描)。通过直接依赖或项目环境，推测间接依赖。间接依赖由依赖图谱预测，获得的是项目依赖的近似值，这种方式是通过预测手段获取，在某些情况下可能无法达到 0%误报。

b) 对于编译过后的项目，会生成完整依赖树，扫描编译过程中的环境依赖树内容。相比于未编译扫描，其优点是依赖树信息完整且正确，并且可达致 0%误报结果。只是项目编译过程可能比较耗时。

我们对扫描方式做了结果对比，以确保项目扫描的准确性，如图 3 所示，编译过后的项目可以将整个依赖树更加完整的显示出来，不会出现漏报或者误报情况。

2) 基于组件的指标：作为依赖项使用的第三方组件，本身的特征可以代表与之相关的风险。每个组件可以找到它的历史版本信息，当前项目中使用的版本以及每个组件对应的漏洞详情。

3) 基于依赖特征的指标：依赖关系的范围和深度与它在应用程序中包含的漏洞的风险有关。依赖关系的深度越深，意味着间接依赖(可传递依赖项)越多。依赖追踪就越清晰。依赖树信息就越完整且正确。

4) 基于安全漏洞的指标：漏洞本身的特征可用于评估风险，从漏洞本身可以分析出该漏洞的详细信息。每个漏洞，都可以找到其完整的漏洞依赖关系，并根据漏洞对应组件，找到其当前版本和补丁版本，进一步计算出修复兼容性。除此以外 CVE 信息可以根据 CVSS 进行评分，即通用漏洞评分系统。它是一

个行业公开标准，其被设计用来评测漏洞的严重程度，并帮助确定所需反应的紧急度和重要度。得分在0.1~3.9之间的通常被认为是低风险漏洞，得分在4.0~6.9之间的是中等风险漏洞，得分在7.0~8.9之间的漏洞是高风险漏洞，得分在9.0~10.0之间的则是超高风险漏洞。

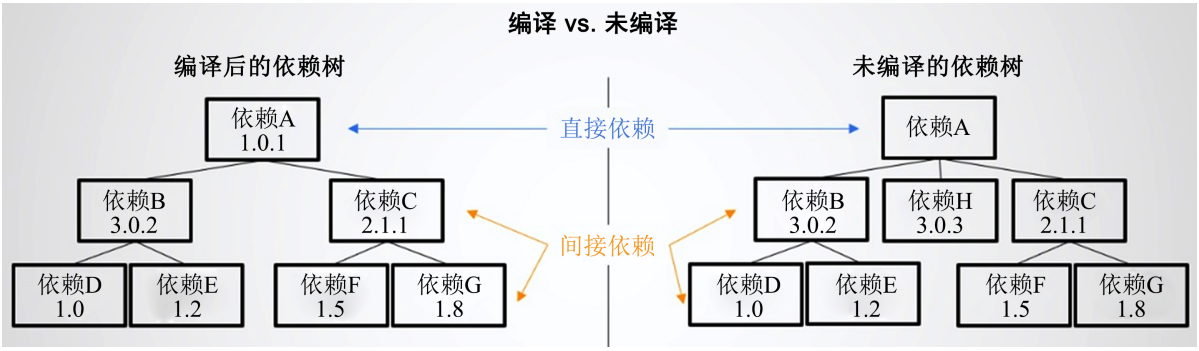

Figure 3. Comparison of compiled and uncompiled dependencies
图 3. 编译与未编译依赖对比

4.2. 交易场景下的区块链项目中第三方组件漏洞现状分析

目的：交易场景相关的项目是对安全要求比较高的领域，其处于整个交易的核心位置，如果无法保证交易的安全性，那么可以遇见的灾难是交易金额错误，交易系统崩溃等一系列问题[5]。扫描结果的量化分析，能够对当前领域的情况进行一个全局性的概述，帮助相关开发人员提高安全的认知。本节通过平均漏洞数、高频组件、漏洞评级等角度来揭示社区中当前漏洞的现状。

方法：对筛选后的项目进行预编译，Java项目通过Maven，Gradle包管理工具对pom.xml，build.gradle文件进行编译，JavaScript项目通过npm工具对package.json文件进行编译，以此构建其依赖树。预编译的依赖树结果见图4，未编译的依赖树结果见图5。最终将获得的依赖树提供给Scantist SCA进行扫描编译过程中的环境依赖树内容。

```
[root@localhost cryptocurrency-api-master]# npm fund
cryptocurrency-api@0.0.0
├─┬ https://github.com/cheeriojs/cheerio?sponsor=1
│ └── cheerio@1.0.0-rc.10
├─┬ https://github.com/sponsors/fb55
│ └── cheerio-select@1.5.0, htmlparser2@6.1.0, css-select@4.1.3, css-what@5.0.1, domelementtype@2.2.0
├─┬ https://github.com/cheeriojs/dom-serializer?sponsor=1
│ └── dom-serializer@1.3.2
├─┬ https://github.com/fb55/domhandler?sponsor=1
│ └── domhandler@4.2.2
├─┬ https://github.com/fb55/htmlparser2?sponsor=1
│ └── htmlparser2@6.1.0
├─┬ https://github.com/fb55/domutils?sponsor=1
│ └── domutils@2.8.0
├─┬ https://github.com/fb55/nth-check?sponsor=1
│ └── nth-check@2.0.1
├─┬ https://github.com/fb55/entities?sponsor=1
│ └── entities@2.2.0
├─┬ https://opencollective.com/nodemon
│ └── nodemon@2.0.13
├─┬ https://github.com/sponsors/ljharb
│ └── resolve@1.20.0, is-core-module@2.7.0, object.assign@4.1.2, call-bind@1.0.2, has-symbols@1.0.2, get-intrinsic@1.1.1
```

Figure 4. Pre-compiled dependency tree
图 4. 预编译的依赖树

```
[root@localhost lapp-demo-master]# cd ../cryptocurrency-api-master/
[root@localhost cryptocurrency-api-master]# npm fund
cryptocurrency-api@0.0.0

[root@localhost cryptocurrency-api-master]# npm install
```

Figure 5. Uncompiled dependency tree
图 5. 未编译的依赖树

并非所有的开源项目都能够构建成功，经过图 4、图 5 对比发现，进行预编译后的项目可以在本地获取完整的依赖树。未编译项目无法获取完整的依赖树，只能通过 Scantist SCA 提供的依赖图谱(即 Air Gap 模式)，来进行预测。为了消除这类因未构建完成，只是通过预测产生的结果带来的影响。对构建失败的项目，进行手工修复了失败项目。预编译的扫描结果见图 6。

```
[root@localhost fas-frontend-master]# java -jar /opt/project/scantist-bom-detect.jar  -working_dir /opt/project/batch/bitcoin-payment/javascript/fas-frontend-master/
[INFO ] 2021-10-07 10:22:02.357 main scantist.ci.Application:41 - You can run with `--debug` and check scantist_report.log for more detail.
[INFO ] 2021-10-07 10:22:02.364 main scantist.ci.PropertyManager:36 - init PropertyManager
[INFO ] 2021-10-07 10:22:02.716 main scantist.ci.ProjectManager:73 - init ProjectManager
[INFO ] 2021-10-07 10:22:02.718 main scantist.ci.ProjectManager:97 - sbd version: 2021/8/25
[INFO ] 2021-10-07 10:22:02.731 main scantist.ci.ProjectManager:116 - Detect package managers...
[INFO ] 2021-10-07 10:22:04.202 main scantist.ci.ProjectManager:126 - detectBcmTool results: [NPM]
[INFO ] 2021-10-07 10:22:04.202 main scantist.ci.ProjectManager:128 - Detect package managers done.
[INFO ] 2021-10-07 10:22:04.203 main scantist.ci.ProjectManager:162 - Scan type: source_code
[INFO ] 2021-10-07 10:22:04.203 main scantist.ci.ProjectManager:174 - start to extract dependency information...
[INFO ] 2021-10-07 10:22:04.218 main utils.Executable.ExecutableUtil:37 - Resolved executable npm with file: /usr/local/bin/npm
[INFO ] 2021-10-07 10:22:09.623 main utils.Executable.ExecutableUtil:113 - finish execution with exit code: 0
[INFO ] 2021-10-07 10:22:09.685 main scantist.ci.ProjectManager:196 - Gather files of interest...
[INFO ] 2021-10-07 10:22:11.137 main scantist.ci.ProjectManager:211 - Gathering files of interest - targetedFiles: 6
[INFO ] 2021-10-07 10:22:11.145 main scantist.ci.ProjectManager:213 - Gathering files of interest done
[INFO ] 2021-10-07 10:22:11.145 main scantist.ci.ProjectManager:217 - generate report
[INFO ] 2021-10-07 10:22:11.156 main scantist.ci.ProjectManager:225 - Triggering scantist scan

[INFO ] 2021-10-07 10:22:11.156 main scantist.ci.ProjectManager:280 - Post timeout sets to 60 seconds.
[INFO ] 2021-10-07 10:22:11.516 main scantist.ci.ProjectManager:454 - Upload to http://119.8.181.73:8237/ci-scan/
[INFO ] 2021-10-07 10:22:13.288 main scantist.ci.ProjectManager:472 - Upload success.
[INFO ] 2021-10-07 10:22:13.288 main scantist.ci.ProjectManager:473 - scan_id:321
[root@localhost fas-frontend-master]#
```

Figure 6. Pre-compiled scan results
图 6. 预编译的扫描结果

4.2.1. CVE 安全漏洞数

当获得了扫描结果后，对 CVE 安全漏洞进行统计学加权平均数方法，以此来展示扫描的 153 个项目中，包含的平均 CVE 安全漏洞数。每个项目所包含的组件数不同，可以统计出单个项目包含最小组件数为 1 个，最大组件数为 2069 个。全部扫描的 153 个项目中包含总组件数为 13,912 个，根据这一结果可以计算其“权”。

平均 CVE 安全漏洞数通过以下公式计算：

$$AVG.Vul = \frac{\sum_{i=1}^{n} x_i f_i}{\sum_{i=1}^{n} f_i}$$

其中，x_i 是每个项目的权值，f_i 是每个项目漏洞数。

根据计算并统计结果得出，平均每个软件项目存在 21.17 个已知开源软件漏洞。这一结果可以让我们清晰的看到，在区块链项目中，漏洞是普遍存在的。尽管交易场景下区块链项目的漏洞数比奇安信《2021 中国软件供应链安全分析报告》中指出的“国内企业平均软件项目存在 66 个漏洞”的总数低，但就交易场景下的区块链项目漏洞数是无法接受的。

4.2.2. 安全等级

除了开源社区平均 CVE 安全漏洞数结果外，本节更加细化的将漏洞进行安全等级划分，划分依据来源于 CVSS 评分系统，如表 1 所示。

Table 1. CVSS score sheet
表 1. CVSS 评分表

Rating	CVSS Score
None	0.0
Low	0.1~3.9
Medium	4.0~6.9
High	7.0~8.9
Critical	9.0~10.0

将 154 个项目中扫描出的 3240 个 CVE 安全漏洞和 2929 个安全隐患，按照 CVSS 评分表，进行四个安全等级分类。安全隐患分析如图 7 所示，CVE 安全漏洞分析如图 8 所示。

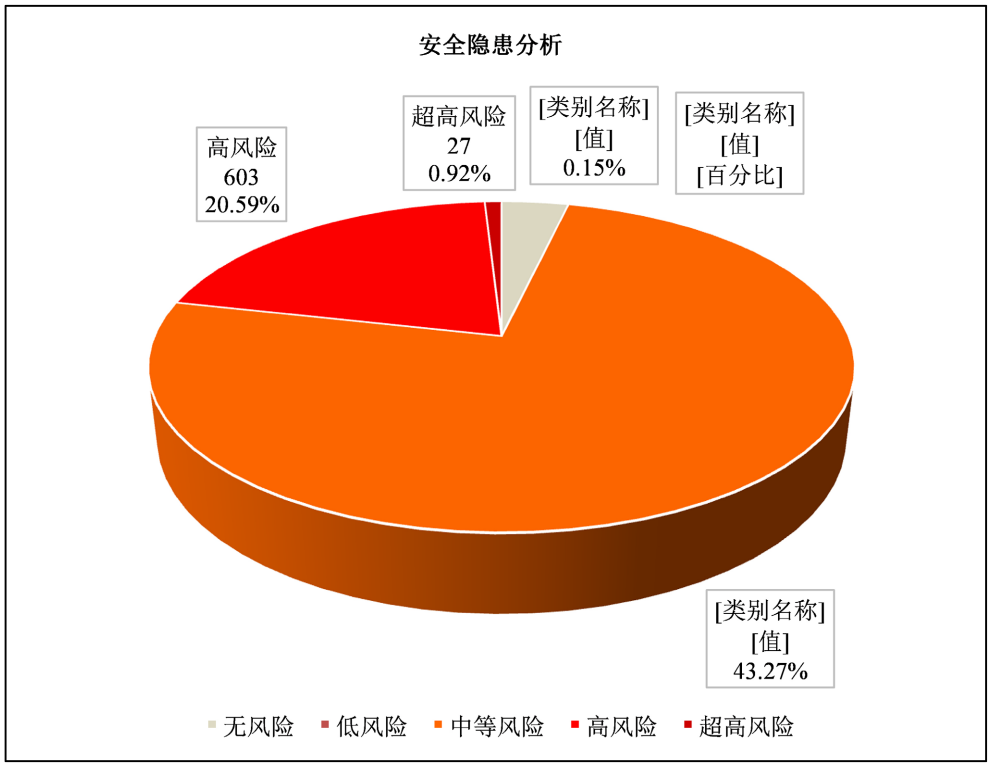

Figure 7. Safety hazard analysis
图 7. 安全隐患分析

其中安全隐患的漏洞数据来源于源代码的修改日志，漏洞追踪和安全报告。

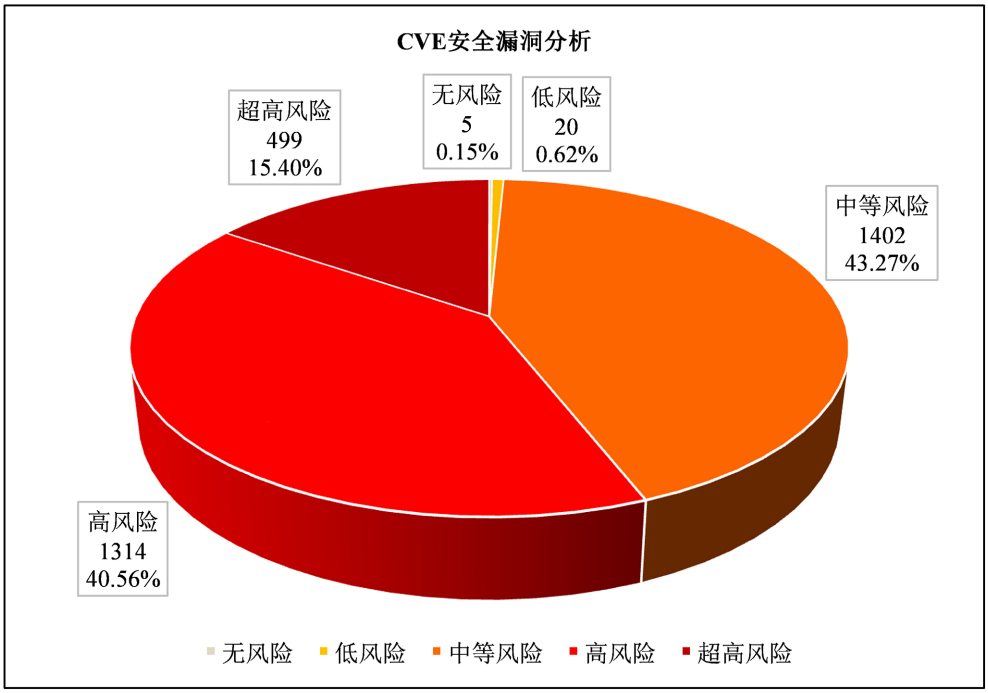

Figure 8. CVE security vulnerability analysis
图 8. CVE 安全漏洞分析

CVE 安全漏洞数据来源于 NVD，指的是已发现的 CVE 安全漏洞。

从安全隐患的角度和 CVE 安全漏洞的角度对比可以发现低风险和超高风险的比例相对来说低一些，但中等风险和高风险仍然占据了区块链项目的绝大部分比例。这也反映出与交易相关的区块链项目的安全情况有待提升。

4.2.3. 公共的漏洞

本小节通过将项目扫描结果按照组件进行归类，目的是找到各个项目中公共使用的组件，并通过公共的组件找到这些公共的 CVE 安全漏洞。一方面可以说明这些 CVE 安全漏洞在区块链项目[6]中影响较广，另一方面也为后续漏洞的修复提供了可能性。

我们归类了所有项目中共同使用的前 20 个组件，组件归类见表 2。

Table 2. Common components
表 2. 公共的组件

组件名	组件描述
Lodash	JavaScript 实用程序库
Minimist	一款命令行参数解析工具
Elliptic	提供快速加密的代码库
Hosted-git-info	识别和转换协议之间各种 git 主机 URL
Glob-parent	从全局字符串中提取非魔术父路径

Continued

Yargs-parser	一款选项解析
Lni	解析和序列化 Ini 格式文件的代码库
Yargs-yn	一个类似 I18n 的代码库
Path-parse	提供路径解析功能的插件
Underscore	函数式编程库
Node-fetch	将 window.fetch 带到 node.js 的轻量级模块
Tar	用于文件压缩/解压缩的软件包
Debug	用于调试程序的模块
Githubws	一个 WebSocket 应用软件
Kind-of	一款类型检查软件包
Brances	Bash 的括号扩展应用程序
Hoek	一个 hapi 系统的实现
Fstream	一个用于读取和写入文件的库
Set-value	一款能够使用点表示法在对象上设置嵌套值的模块
Deep-extend	开源的递归对象扩展模块

当通过扫描结果获得公共组件后，便可以依据公共组件来找到对应的 CVE 漏洞。需要注意的是，有时候一个组件会对应多个 CVE 漏洞。具体对应关系如表 3 所示。

Table 3. Correspondence table between components and vulnerabilities
表 3. 组件与漏洞对应关系表

组件名	漏洞编号	危险程度	CVSS 打分	影响项目数
Lodash	CVE-2020-28500	中等	5.3	84
	CVE-2019-10744	超高风险	9.1	
	CVE-2018-16487	中等	5.6	
	CVE-2019-1010266	中等	6.5	
	CVE-2018-3721	中等	6.5	
	CVE-2020-8203	高	7.4	
Minimist	CVE-2020-7598	中等	6.8	80
Elliptic	CVE-2020-28498	中等	5.4	76
	CVE-2020-13822	中等	6.8	
Hosted-git-info	CVE-2021-23362	中等	5.3	61

Continued

Glob-parent	CVE-2020-28469	高	7.5	58
Yargs-parser	CVE-2020-7608	中等	4.6	54
Lni	CVE-2020-7788	高	7.3	52
Yargs-yn	CVE-2020-7774	高	7.3	50
path-parse	CVE-2021-23343	高	7.5	46
Underscore	CVE-2021-23358	高	7.2	41
Node-fetch	CVE-2020-15168	中等	5.3	34
Tar	CVE-2018-20834	高	7.5	33
Debug	CVE-2017-16137	中等	5.3	33
Githubws	CVE-2021-32640	中等	5.3	30
Kind-of	CVE-2019-20149	中等	5	30
Brances	CVE-2018-1109	中等	5.3	29
Hoek	CVE-2018-3728	高	8.8	27
Fstream	CVE-2019-13173	高	7.5	25
Set-value	CVE-2019-10747	超高	9.8	22
Deep-extend	CVE-2018-3750	超高	9.8	21

通过表 2 容易发现，项目共同使用的组件的功能往往趋向于基础工具类模块，如路径解析模块、文件解压缩模块、程序调试模块、读写文件模块等。这种基础性的组件，不仅仅在区块链领域频繁使用，可以预见的是，在开源社区中其他项目中也是会被广泛使用。

通过表 3 不难发现，这类组件对应的 CVE 安全漏洞，其危险程度往往在中等风险或高风险。影响数目广，亟需给出解决方案，修复漏洞。

4.3. 第三方组件漏洞修复分析

目的：在 4.2.3 节，我们对公共组件漏洞情况进行了分析，找到了项目中公共使用的组件，并发现这些组件往往影响的项目数都在几十个不等，这意味着一旦修复了这些高频使用的组件，那对整体的安全性的帮助将是巨大的。本节通过案例研究使我们深入了解组件以及漏洞修复情况。

4.3.1. Scantist SCA 进行漏洞推荐修复

Scantist SCA 具有扫描后对漏洞进行推荐修复的能力，即给含有漏洞的组件提供修复建议，并根据需求进行安全替换[7]。其工作原理是：根据扫描找到项目中包含的组件，一个组件往往包含多个 CVE 漏洞。Scantist SCA 会去漏洞数据库对组件进行匹配，数据库中包含该组件的所有版本信息。从中可以找到不含漏洞的组件版本，我们根据当前使用的组件版本与无漏洞版本进行对比，最终对组件进行修复并计算出修复后的兼容度。组件信息如表 4 所示，推荐修复情况如表 5 所示。

Table 4. Take the component Lodash as an example
表 4. 以组件 Lodash 为例

版本	发行日期	CVE
4.17.21	20/02/2021	0
4.17.20	14/08/2020	2
4.17.19	09/07/2020	3
4.17.11	13/09/2018	5
4.17.10	25/04/2018	8
4.17.4	01/01/2017	10
4.16.4	07/10/2016	11

Table 5. Recommended repair
表 5. 推荐修复

Component Name	Current Version	Fix Version	Severity	Fix Compatibility	Vulnerabilities Fixed
Commons-File Upload	1.2.1	1.3.3	Critical	Medium	CVE-2013-0248, CVE-2014-0050, CVE-2016-3092, SEC-2019-10014, CVE-2016-1000031
Commons-Collections	1.0	3.2.2	Critical	Medium	CVE-2017-15708, CVE-2015-6420, SEC-2019-11294
Commons-BeanUtils	1.9.0	1.9.4	Critical	High	CVE-2014-0114, CVE-2019-10086, CVE-2017-15708, SEC-2019-11294, CVE-2015-6420
Poi	3.6	4.1.1	Critical	Low	CVE-2017-12626, SEC-2019-10024, CVE-2019-12415, CVE-2014-3574, CVE-2014-9527, CVE-2014-3529, CVE-2017-5644, CVE-2012-0213, CVE-2016-5000, CVE-2019-17571
Commons-IO	2.0	2.7	MEDIUM	Medium	CVE-2021-29425
HttpClient	4.3.5	4.5.13	MEDIUM	Medium	CVE-2015-5262, CVE-2020-13956, SEC-2019-10019

从表 4 中很容易看出，Lodash 组件在早期版本中，持续被爆出 CVE 漏洞，但是到了 4.17.21 版本就不含漏洞了，所以对于该组件，可以将其版本更新为 4.17.21。但是需要注意的是，组件的版本修改后，会出现不匹配的问题。在表 5 中的 Fix Compatibility 中会提示修复兼容性。对于修复兼容性高的组件，修复后往往不会出现问题。而对于修复兼容性低的组件，可能会出现修复后程序无法编译现象，所以对于此类问题，需要我们在修复漏洞后做一个兼容性问题检查。

4.3.2. 修复兼容性分析

我们从扫描的项目中选取了 10 个具有代表性的项目进行漏洞修复后的代码兼容性验证，这些项

目平均包含 50+个漏洞。在筛选完项目后，通过 Scantist SCA 进行推荐修复。

对修复后的项目进行了重新编译，观察修复后的项目是否可以继续使用。结果表明，9 个项目修复后的项目中正常使用，不受组件修复的影响，即高兼容度。但仍然有 1 个项目修复后会出现无法编译通过的情况，即组件修复不兼容的情况。

基于此，对剩余的无法编译的项目进行分析，最终发现当含有漏洞组件版本号与无漏洞版本号差距很大时，比较容易出现不兼容现象，因为无漏洞版本的 API 发生了改变，导致使用无漏洞版本的组件后，出现无法识别现象。这种情况下，对程序的代码进行重构，使新编写的代码能够与新的组件兼容[8]。

随后，再次扫描修复后的项目，发现所有项目中的组件都不包含漏洞问题，并且可以正常使用。经分析，可以得出结论，通过 Scantist SCA 对项目进行组件成分分析后，可以发现一个项目中包含的漏洞，并且根据组件情况，进行漏洞修复，最终保证区块链项目的安全。

5. 讨论

经过上述的分析，我们发现开源社区中的区块链项目或多或少都存在安全问题[9]。面对如此庞大的漏洞情况，用 SCA 工具检测是十分有必要的，在经过 SCA 工具的检测以及修复后，可以将有安全漏洞的项目变成不含漏洞的项目。虽然我们也可以等开发过程结束后，进行漏洞测试，再进行修复，但这样的方式是一种亡羊补牢的方式，它会引入向后兼容性的风险，因为不同版本的第三方组件作用会有细微差异，这就引入了兼容性的问题，我们往往还需要将修复后的代码进行简单的重构。

面对这个问题，我们可以在项目开发初期，就将安全防护嵌入开发，为实现这一点，最好的方式是实施安全防护左移策略。安全防护“左移”就是将安全防护尽可能地移到开发流程的早期。比如在开发流程的早期，还在确定选择哪些第三方组件作为开发需要的组件的时候，就可以使用 SCA 进行安全扫描，将有问题的组件替换成相同或者无安全问题的版本。这样就确保开发过程中使用的组件始终是安全的。在开发中期如果需要引入新的组件，可以再次进行安全评估，以保证整个开发过程的安全性。安全防护左移不仅有利于降低网络风险，还可以降低成本。IBM 研究院发现，在设计过程中解决安全问题，比在实施过程中解决的成本要低六倍。

6. 总结和展望

6.1. 本文总结

本文主要分析和研究了开源社区中区块链项目的漏洞情况。本文提出了一个基于 Scantist SCA 的漏洞扫描方法。该方法基于一个基本事实：目前区块链技术发展迅速，大家关注点在区块链理论的安全性。但是很少对基于区块链进行开发的项目的安全性做一个系统性的现状分析，尤其是交易场景下的区块链项目。我们对这些项目的安全性情况不了解，但是却离不开这些项目，如比特币电子钱包，比特币支付处理系统等。由此，我们主要做了两个方面的研究。

第一个方面是 GitHub 社区中区块链项目的整体情况分析。我们通过统计方法计算了平均每个项目的漏洞情况，并对项目中漏洞的风险等级做了评估，最后也找到了该领域的项目公共组件的漏洞情况。

第二个方面是根据推荐的修复统计了修复率，并尝试使用 SCA 工具对漏洞进行修复。对不能通过推荐修复的漏洞，提出了修复办法。

6.2. 未来工作展望

虽然我们对区块链项目的代码漏洞情况进行了分析，但是仍然存在不足之处。未来将针对不足之处

展开分析：

1) SCA 工具是否可以除了建议修复选项外，同时对向后兼容的风险进行解释。先前的工作发现，开发人员害怕破坏原有的组件结构是他们不想更新易受攻击的依赖项的主要原因之一。我们可以进行更深入地分析，以了解依赖项中的某个版本更改会导致哪些代码更改，以及它如何影响依赖项应用程序。

2) 开发人员如何响应来自 SCA 工具的警报也不在本次研究的范围之内。类似的，是什么导致依赖性漏洞误报是一个开放的研究问题[10]，超出了本研究的范围。当前我们进行案例研究，以调查开源社区中区块链项目的漏洞情况和漏洞修复后给社区安全性带来的提升为目的。未来开发人员对 SCA 工具的响应情况还可以做进一步研究。

基金项目

本研究得到国家自然科学基金面上项目资助，项目编号为 61972359；得到浙江省自然科学基金一般项目资助，项目编号为 LY19F020052。

参考文献

[1] 李婷. 面向区块链智能合约的实时漏洞检测技术研究[D]: [硕士学位论文]. 成都: 电子科技大学, 2021.

[2] 涂良琼, 孙小兵, 张佳乐, 蔡杰, 李斌. 智能合约漏洞检测工具研究综述[J]. 计算机科学, 2021(11): 79-88.

[3] 黄凯峰. 以太坊平台智能合约漏洞检测工具研究[D]: [硕士学位论文]. 深圳: 深圳大学, 2019.

[4] Scantist (2021) Poly Network Hack: Managing Open Source Vulnerabilities. https://scantist.com/resources/blog/poly-network-hack

[5] Roman, B., Christian, B., Juho, L. and Matti, R. (2017) Opportunities and Risks of Blockchain Technologies (Dagstuhl Seminar 17132). *Dagstuhl Reports*, **7**, 99-142. https://doi.org/10.4230/DagRep.7.3.99

[6] Parizi, R.M., Dehghantanha, A., Choo, K.-K.R. and Singh, A. (2018) Empirical Vulnerability Analysis of Automated Smart Contracts Security Testing on Blockchains. *Proceedings of the* 28*th Annual International Conference on Computer Science and Software Engineering* (*CASCON* 18), 103-113. https://arxiv.org/abs/1809.02702

[7] Ombredanne, P. (2020) Free and Open Source Software License Compliance Tools for Software Composition Analysis. *Computer*, **53**, 105-109. https://doi.org/10.1109/MC.2020.3011082

[8] Yang, E. (2018) Fuzz Testing & Software Composition Analysis in Software Engineering. 2018 *International Symposium on VLSI Design, Automation and Test* (*VLSI-DAT*), **2018**, 1-3. https://doi.org/10.1109/VLSI-DAT.2018.8373240

[9] Tschannen, P. and Ahmed, A. (2020) Bitcoin's APIs in Open-Source Projects: Security Usability Evaluation. *Electronics*, 9, Article No. 1077. https://www.mdpi.com/2079-9292/9/7/1077

[10] Imtiaz, N., Thorn, S. and Williams, L. (2021) A Comparative Study of Vulnerability Reporting by Software Composition Analysis Tools. *IEEE International Symposium on Empirical Software Engineering and Measurement* (*ESEM*), Article No. 5. https://doi.org/10.1145/3475716.3475769