D-chain

AUDIT REPORT

Version 1.0.0

Serial No. 20230504000012019

Presented by Fairyproof

May 4, 2023

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the D-chain BlockChain project.

Audit Start Time:

April 24, 2023

Audit End Time:

May 4, 2023

The goal of this audit is to review D-chain's Go implementation for its blockchain functions, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the D-chain team for specified versions. Whenever the code, software, materials, settings, environment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from off-chain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code Review, Including:

- Project Diagnosis

Understanding the size, scope and functionality of your project's source code based on the specifications, sources, and instructions provided to Fairyproof.

- Manual Code Review

Reading your source code line-by-line to identify potential vulnerabilities.

- Specification Comparison

Determining whether your project's code successfully and efficiently accomplishes or executes its functions according to the specifications, sources, and instructions provided to Fairyproof.

2. Testing and Automated Analysis, Including:

- Test Coverage Analysis

Determining whether the test cases cover your code and how much of your code is exercised or executed when test cases are run.

- Symbolic Execution

Analyzing a program to determine the specific input that causes different parts of a program to execute its functions.

3. Best Practices Review

Reviewing the source code to improve maintainability, security, and control based on the latest established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

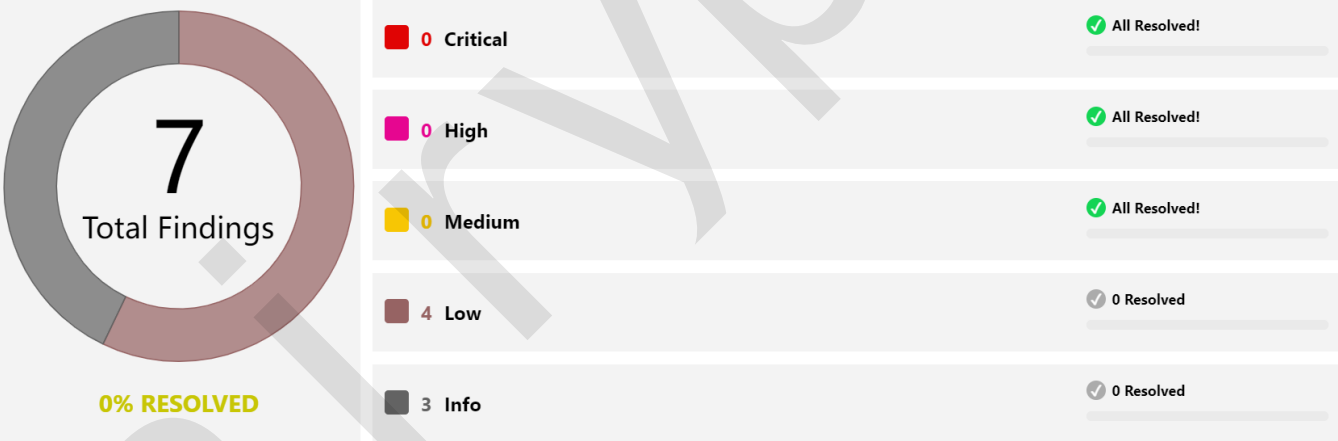
For this audit, we used the following source(s) of truth about how the token issuance function should work:

Source Code: SourceCode

This was considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the D-chain team or reported an issue.

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2023050400012019	Fairyproof Security Team	Apr 24, 2023 - May 4, 2023	Low Risk



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, four issues of low-severity and three issues of info-severity were uncovered. The D-chain team acknowledged all the issues.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Introduction to D-chain

D-chain is an a permissioned consortium blockchain that provides developers with the configurable projects needed to make blockchain applications successful. D-chain implements the QBFT proof of authority (PoA) consensus protocols.PoA consensus protocols work when participants know each other and there is a level of trust between them. A group of nodes in the network act as validators vote to add or remove network nodes.

By default D-chain is a free gas network, however gas can be enabled if required.

At the same time, D-chain provides two types of transactions, the one is public transactions and the other is private transactions.

QBFT is a blockchain consensus protocol based on the Byzantine-fault-tolerant (BFT).
QBFT provides the following features:

- Immediate Finality
- Dynamic Validator Set
- Optimal Byzantine Resilience (the protocol can withstand up to `(n - 1) / 3` validators malfunctioning or behaving maliciously) when operating in a partially synchronous network
- Message complexity of O(n2), where n is the number of validators

The above description is quoted from relevant documents of D-chain.

04. Major functions of audited code

The audited code mainly implements a D-chain's client and it is based on Ethereum's Go implemented client.

05. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Access Control

- Admin Rights
- Arithmetic Precision
- Code Improvement
- Signature Security
- Design Vulnerability
- DoS Attack
- Coding Security
- Transaction Model Security
- Node Communication Security
- Data Storage Security
- RPC Security
- Data History Vulnerability
- Block Processing Security
- Consensus Security
- System Contract Security
- Account Security
- Misc

06. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Informational is not an issue or risk but a suggestion for code improvement.

07. Major areas that need attention

Based on the provided source code the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

- Static Code Analysis

We checked issues that were flagged during static code analysis.
We found some issues, for more details please refer to [FP-1,FP-2,FP-3,FP-4,FP-5,FP-7] in "09. Issue description".

- P2P Security

We checked for code that may affect node authentication, communication encryption security, message format verification, and significantly degrade node performance.

We didn't find issues or risks in these functions or areas at the time of writing.

- Consensus Security

We checked for code that may affect the legitimacy of consensus nodes, consensus security, consensus implementation's security, consensus nodes' fault tolerance, ultimate consistency and finality, and the resistance of the consensus mechanism to attacks.

We didn't find issues or risks in these functions or areas at the time of writing.

- Account Security

We checked for code that may affect account management, permission verification, the security of private key/mnemonic/certificate generation algorithms, and the security of private key/mnemonic/certificate storage and usage.

We didn't find issues or risks in these functions or areas at the time of writing.

- Data Storage Security

We checked for code that may affect the security of data classification storage, sensitive data encryption, data access permissions, and database stability.

We didn't find issues or risks in these functions or areas at the time of writing

- Transaction Model Security

We checked for code that may affect transaction encryption, transaction signature, transaction processing logic, and the resistance of transactions to replay attacks.

We didn't find issues or risks in these functions or areas at the time of writing.

- Miscellaneous

We checked the code for optimization and robustness.
We found one issue, for more details please refer to [FP-6] in "09. Issue description".

08. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Use of Weak Random Number Generator	Unsafe Random Number	Low	Acknowledged
FP-2	TLS MinVersion Too Low	Design Vulnerability	Low	Acknowledged
FP-3	Use of Net/Http Serve Function That Has No Support for Setting Timeouts	DoS Attack	Low	Acknowledged
FP-4	Lack of Setting for ReadHeaderTimeout	DoS Attack	Low	Acknowledged
FP-5	TLS InsecureSkipVerify Set True	Design Vulnerability	Info	Acknowledged
FP-6	Errors Unhandled	Design Vulnerability	Info	Acknowledged
FP-7	Use of Unsafe Calls	Design Vulnerability	Info	Acknowledged

09. Issue descriptions

[FP-1] Use of Weak Random Number Generator

Unsafe Random Number Low Acknowledged

Issue/Risk: Unsafe Random Number

Description:

math/rand is much faster for applications that don't need crypto-level or security-related random data generation. crypto/rand is suited for secure and crypto-ready usage, but it's slower. But in most cases, crypto/rand is likely to be more suitable, unless the performance is critical but the application's security is not (which is rare).

It is highly recommended to use crypto/rand when needing to be secure with random numbers such as generating session IDs in a web application.

Recommendation:

Consider useing `crypto/rand` instead of `math/rand`:

```
package main

import "crypto/rand"

func main() {
    good, _ := rand.Read(nil)
    println(good)
}
```

Update/Status:

The D-chain team has acknowledged the issue.

[FP-2] TLS MinVersion Too Low

Design Vulnerability | Low | Acknowledged

Issue/Risk: Design Vulnerability

Description:

`qlight/config.go` line 93: lack of `TLS` ' Settings for `MinVersion` and `MaxVersion`.

Recommendation:

Consider adding settings as follows:

```
package main

import "crypto/tls"

func saferTLSConfig() {
    config := &tls.Config{}
    config.MinVersion = tls.VersionTLS12
    config.MaxVersion = tls.VersionTLS13
}
```

Update/Status:

The D-chain team has acknowledged the issue.

[FP-3] Use of Net/Http Serve Function That Has No Support for Setting Timeouts

DoS Attack Low Acknowledged

Issue/Risk: DoS Attack

Description:

HTTP timeouts are necessary to expire inactive connections and failing to do so might make the application vulnerable to attacks like slowloris which work by sending data very slow, which in case of no timeout will keep the connection active eventually leading to a denial-of-service (DoS) attack.

Lack of settings in In line 64 of `metrics/exp/exp.go`, line 311 of `cmd/faucet/faucet.go` and line 261 of `internal/debug/flags.go`

Recommendation:

Consider making changes as follows:

```
package main

import (
    "fmt"
    "time"
    "net/http"
)

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintf(w, "Hello, %s!", r.URL.Path[1:])
    })

    server := &http.Server{
        Addr:              ":1234",
        ReadHeaderTimeout: 3 * time.Second,
    }

    err := server.ListenAndServe()
    if err != nil {
        panic(err)
    }
}
```

Update/Status:

The D-chain team has acknowledged the issue.

[FP-4] Lack of Settings for ReadHeaderTimeout

DoS Attack | Low | Acknowledged

Issue/Risk: DoS Attack

Description:

It may be vulnerabile to potential slowloris attacks because ReadHeaderTimeout is not configured in the http.Server. Slowloris is a DoS attack (Denial-of-Service) that tries to overwhelm the target by opening and keeping open multiple HTTP connections. All those connections will sature the server, making it unable to open new ones for legit requests. It lacks settings for `ReadHeaderTimeout` in line 140 of `node/rpcstack.go`

Recommendation:

Consider adding settings for the `ReadHeaderTimeout` field of server.

Update/Status:

The D-chain team has acknowledged the issue.

[FP-5] TLS InsecureSkipVerify Set True

Design Vulnerability | Info | Acknowledged

Issue/Risk: Design Vulnerability

Description:

In line 27 of `qlight/config.go` , the value of `InsecureSkipVerify` is true. It means D-Chain needs to customize verification and be careful.

Recommendation:

Consider adding the setting options for the `ReadHeaderTimeout` field of server.

Update/Status:

The D-chain team has acknowledged the issue.

[FP-6] Errors Unhandled

Design Vulnerability | Info | Acknowledged

Issue/Risk: Design Vulnerability

Description:

In the implementation, many codes need error handlings such as `s.Write(hex)` in line 485 of `common/types.go`.

Recommendation:

Consider using `GoSec` to inspect all the codes and adding necessary error handlings.

Update/Status:

The D-chain team has acknowledged the issue.

[FP-7] Use of Unsafe Calls

Design Vulnerability | Info | Acknowledged

Issue/Risk: Design Vulnerability

Description:

Multiple codes use `unsafe calls` such as the code in line 80 to line 82 in `common\bitutil\bitutil.go` using `unsafe.Pointer(&dst)`. This should be used with caution.

Recommendation:

Consider using `unsafe calls` with great caution

Update/Status:

The D-chain team has acknowledged the issue.

10. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- Consider using a static analysis tool for Go language to check the implementation and checking for each of the flagged issues.

11. Appendices

- N/A

11.2 External Functions Check Points

- N/A

FAIRYPROOF

https://medium.com/@FairyproofT

https://twitter.com/FairyproofT

https://www.linkedin.com/company/fairyproof-tech

https://t.me/Fairyproof_tech

Reddit: https://www.reddit.com/user/FairyproofTech