CryptoBillions Security Audit Report

November 21, 2024

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

BugBlow utilized a widely adopted approach to performing a security audit. Below is a breakout of how our team was able to identify and exploit vulnerabilities.

1.2.1 Code and architecture review:

- Review the source code.
- Read the project's documentation.
- Review the project’s architecture.

Stage goals

- Build a good understanding of how the application works.

1.2.2 Check code against vulnerability checklist:

- Understand the project’s critical assets, resources, and security requirements.
- Manual check for vulnerabilities based on the auditor's checklist.
- Run static analyzers.

Stage goals

- Identify and eliminate typical vulnerabilities (gas limit, replay, flash-loans attack, etc.)

1.2.3 Threat & Attack Simulation:

- Analyze the project's critical assets, resources and security requirements.
- Exploit the found weaknesses in a safe local environment.
- Document the performed work.

Stage goals

- Identify vulnerabilities that are not listed in static analyzers that would likely be exploited by hackers.
- Develop Proof of Concept (PoC).

1.2.4 Report found vulnerabilities:

- Discuss the found issues with developers
- Show remediations.
- Write an initial audit report.

Stage goals

- Verify that all found issues are relevant.

1.2.5 Fix bugs & re-audit code:

- Help developers fix bugs.
- Re-audit the updated code again.

Stage goals

- Double-check that the found vulnerabilities or bugs were fixed and were fixed correctly.

1.2.4. Deliver final report:

- The Client deploys the re-audited version of the code
- The final report is issued for the Client.

Stage goals

- Provide the Client with the final report that serves as security proof.

Severity classification

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Vulnerabilities leading to assets theft, fund access locking, or any other loss of funds.
High	Vulnerabilities that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Vulnerabilities that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Vulnerabilities that do not have a significant immediate impact and could be easily fixed.

1.3 Project Overview

This project implements a smart contract for distributing revenue to investors based on their predefined share percentages. It receives funds (in the form of Jettons) and calculates each investor's allocation according to their ownership stake. The contract ensures transparent transfers by validating the sender and managing distributions directly to investors' Jetton wallets.

1.4 Project Dashboard

Project Summary

Title	Description
Client name	CryptoBillions
Project name	Shareholder
Timeline	12.11.2024 - 20.11.2024
Number of Auditors	1

Project Last Log

Date	Commit Hash	Note
10.11.2022	104a2e3f80beee1f15143a4979384616e4ced8ec	Fixes

Project Scope

The audit covered the following files:

File name	Link
/contract/constants.fc	https://github.com/crypto-billions/contract-shareholders/blob/main/contracts/constants.fc
/contract/jetton-utils.fc	https://github.com/crypto-billions/contract-shareholders/blob/main/contracts/jetton-utils.fc
/contract/op-codes.fc	https://github.com/crypto-billions/contract-shareholders/blob/main/contracts/op-codes.fc
/contract/params.fc	https://github.com/crypto-billions/contract-shareholders/blob/main/contracts/params.fc
/contract/shareholder.f	https://github.com/crypto-billions/contract-

c	shareholders/blob/main/contracts/shareholder.fc

1.5 Summary of findings

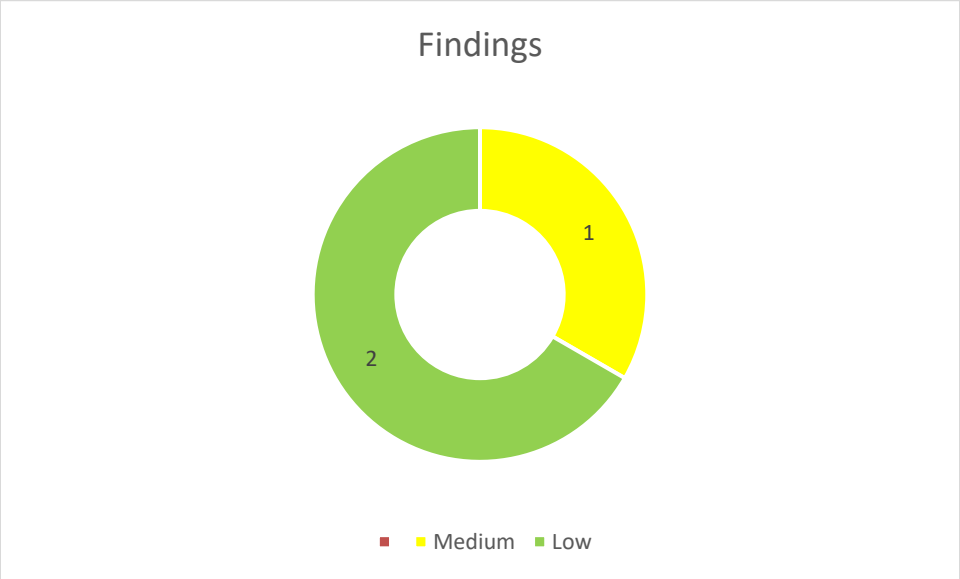

Figure 1. Findings chart

Severity	# of Findings
CRITICAL	0
HIGH	0
MEDIUM	1
LOW	2

2. FINDINGS REPORT

2.1 Critical

Not found

2.2 High

Not found

2.3 Medium

2.3.1 Unfair distribution of shares

Status: acknowledged

Description

The following formula is used to calculate the amount of shares for each investor to receive.

```
int op::transfer() asm "0xf8a7ea5 PUSHINT";
```

```
int shareholder_amount = (jetton_amount * percent) / 10000;
```

The *jetton_amount* is a user-controlled field and if sent less than

$$jetton\_amount < \frac{10000}{percent},$$

the *shareholder_amount* will result in 0. The sent jetton_amount must be checked.

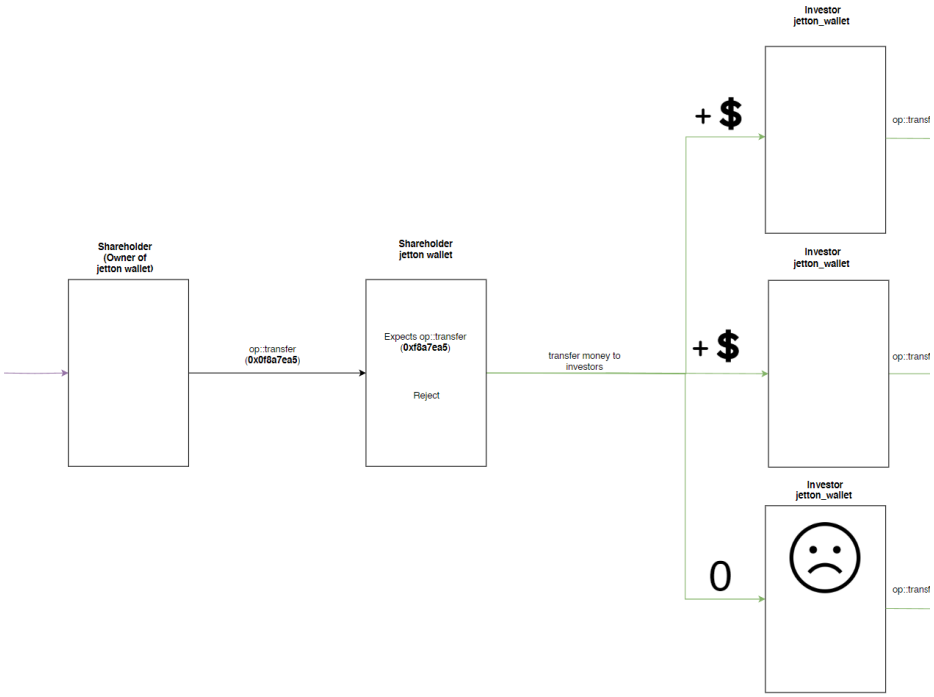

Figure 3. Unfair distribution of shares in edge scenarios.

Consider the following example:

Jetton amount is 9. The investors percentages go as follows: 6000 (*60%*), 3000 (*30%*), 1000 (*10%*).

Because of integer division one of the investors will receive 0.

```
int shareholder_amount = (9 * 6000) / 10000 = 54 000 / 10 000 = 5
int shareholder_amount = (9 * 3000) / 10000 = 27 000 / 10 000 = 2
int shareholder_amount = (9 * 1000) / 10000 = 9000 / 10 000 = 0
```

Remediation

1. Check sent jetton_amount. It must be more than min(percent) / 10000.

$$jetton\_amount \geq \frac{10000}{\min(percent)},$$

2.4 Low

2.4.1 Insufficient amount of gas to cover distribution.

Status: acknowledged

Description

The shareholder contract checks whether sent Ton is enough to cover the complex transaction.

```
throw_unless(error::insufficient_transfer_fee, msg_value > (forward_ton_amount *
shareholder_count) + (2 * const::gas_consumption + const::min_tons_for_storage));
```

Indeed, the forward_ton_amount must cover the expenses for the investors jetton wallet contracts to receive the money. However, the transfer transaction from the shareholder.fc contract itself represents n transactions, where n = shareholder_count. Thus, the const::gas_consumption must be multiplied by n. The correct formula should be

```
throw_unless(error::insufficient_transfer_fee, msg_value > (forward_ton_amount *
shareholder_count) + (shareholder_count * const::gas_consumption +
const::min_tons_for_storage));
```

Remediation

1. Fix the fee requirements with the formula above.

2.2.2 Operation code mismatch

Status: acknowledged

Description

There is an operation code mismatch between the shareholder contract and its jetton wallet.

The shares distribution contract (shareholder.fc) uses op::jetton::transfer code that is defined as 0x0f8a7ea5.

constants/constants.fc:

```
const int op::jetton::transfer = 0x0f8a7ea5;
```

However, its jetton wallet contract that receives the message for transfer, expects a slightly different code 0xf8a7ea5.

```
int op::transfer() asm "0xf8a7ea5 PUSHINT";
```

Figure 2. Failed transfer

The extra “0” in the first code has probably been added by accident, and this slight mistake doesn’t affect anything, because in hexadecimal an extra 0 added at the beginning results in the same number. However, we recommend using the same constant for situations like this to avoid potential problems in the future.

Remediation

1. Change the operation code from 0x0f8a7ea5 to 0xf8a7ea5
2. Use the same constants library when used in different contracts

CONSCLUSION

In this audit, we identified several problems that could impact the security and operability of the contract:

- Medium: Unfair Distribution of Shares: Integer division in the share allocation formula can result in some investors receiving no shares in edge scenarios where the jetton amount is insufficient to meet the percentage allocations. This introduces unfairness, especially for investors with smaller allocations. Remediation includes adding validation to ensure that the jetton_amount exceeds the minimum required for distribution.
- Low: Insufficient Gas for Distribution: The gas calculation formula for covering distribution transactions fails to account for the cumulative gas consumption of executing transfer transactions to all recipient wallets. This oversight can cause distribution failures due to inadequate gas. The formula must be updated to reflect the correct gas requirements, ensuring all transactions are processed successfully.
- Low: Operation code mismatch: An operation code mismatch between the shareholder contract and its jetton wallet prevents outgoing transfers from functioning correctly. This mismatch (an extra "0" in the operation code) arises from the problem that different constants are used for the same operation.

Each issue has been documented with appropriate remediation steps. Addressing these issues will enhance the security and robustness of the Aqua protocol.

The Client is advised to implement the recommended changes and conduct a follow-up audit to verify the effectiveness of this remediation.