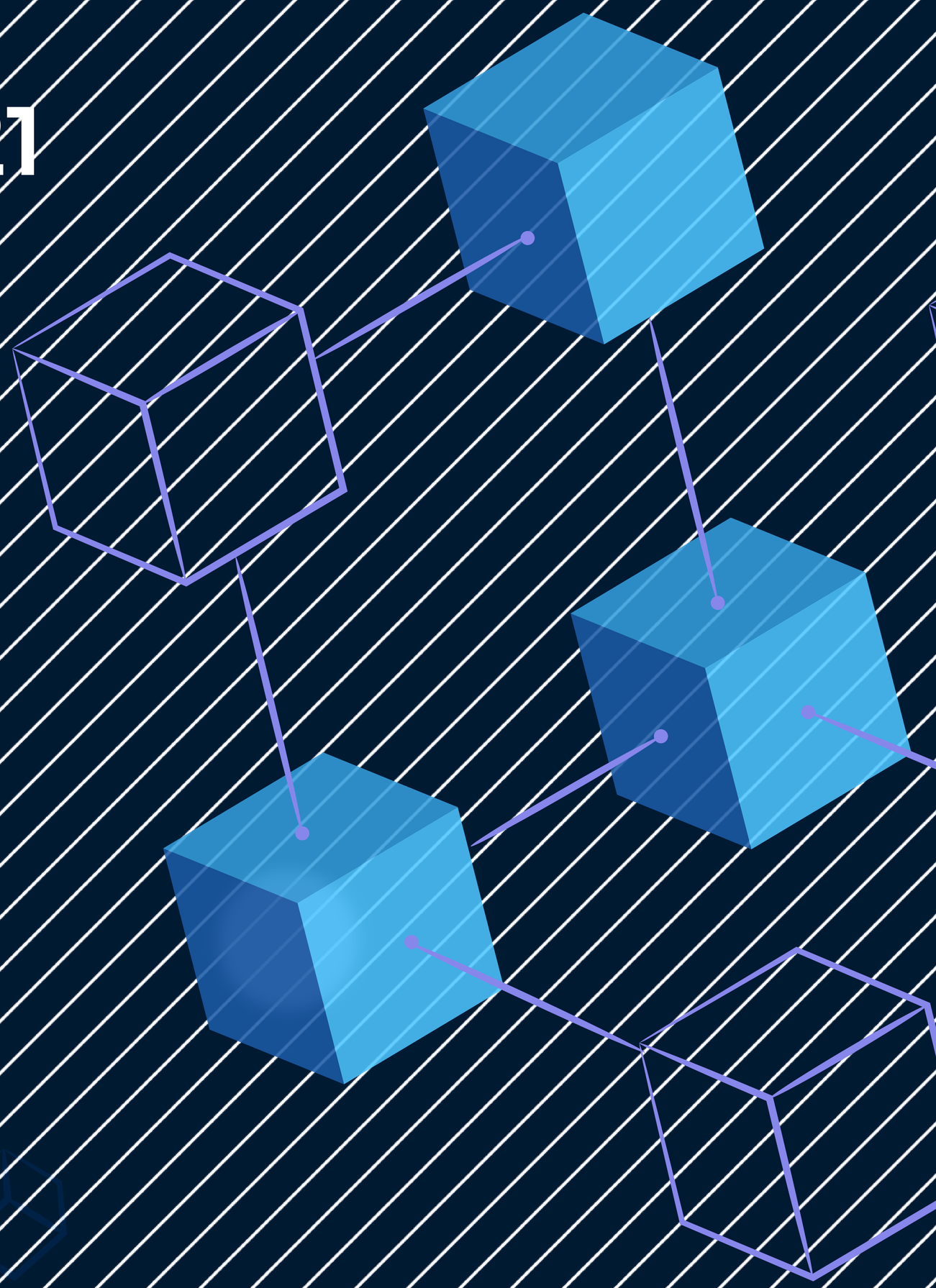
QuillAudits

Audit Report
November, 2021



For

Token Stream

Contents

Scope of Audit	01
Check Vulnerabilities	01
Techniques and Methods	02
Issue Categories	03
Number of security issues per severity.	03
Introduction	04
A. Contract - BSC_Stream.sol	05
High Severity Issues	05
Medium Severity Issues	05
Low Severity Issues	05
Informational Issues	06
Functional Tests	08
Closing Summary	09

Scope of the Audit

The scope of this audit was to analyse TokenStream.App **BSC-stream smart contract**'s codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC20 transfer() does not return
- boolean
- ERC20 approve() race
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis
Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis
Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption
In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Solhint, Mythril, Slither.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
High	A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.
Medium	The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.
Low	Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.
Informational	These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	0	1	5
Closed	0	0	0	0

Introduction

During the period of **October 20, 2021 to October 26, 2021** - QuillAudits Team performed a security audit for the **TokenStream.app** smart contract.

Codebase: https://github.com/Pandora-Finance/BSC-stream-contract
Branch: main
Commit: fe11a461a37eb5cc48d2f97e7ec97312c966b25f
Fixed In: TBD

BSC_Stream.sol	Functionality to create, withdraw and cancel a stream
Types.sol	Contains Stream Struct type

Issues Found

A. Contract – BSC_Stream.sol

High severity issues

No issues were found.

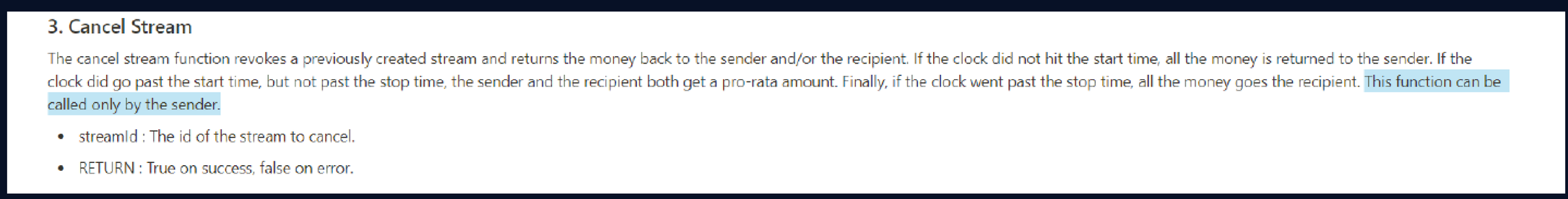
Medium severity issues

No issues were found.

Low severity issues

- **Business Logic Mismatch**

  The documentation provided states, that stream should only be canceled by the sender of the stream


3. Cancel Stream

The cancel stream function revokes a previously created stream and returns the money back to the sender and/or the recipient. If the clock did not hit the start time, all the money is returned to the sender. If the clock did go past the start time, but not past the stop time, the sender and the recipient both get a pro-rata amount. Finally, if the clock went past the stop time, all the money goes the recipient. This function can be called only by the sender.

- streamId : The id of the stream to cancel.
- RETURN : True on success, false on error.

  But the contract allows both the sender or recipient of the stream to cancel it.

```
273         function cancelStream(uint256 streamId)
274             external
275             nonReentrant
276             streamExists(streamId)
277             onlySenderOrRecipient(streamId)
278             returns (bool)
279         {
```

Status: Acknowledged

DEV Comments: “This will not cause any loss of funds from the sender side and receiver will not ever cancel the money sent to them”. Also the documentation will be updated supporting the implemented logic.

Informational Issues

- Old compiler has been used

  Recommendation: Use latest compilers so as to avoid bugs introduced in old compilers

  Status: Acknowledged

- Public functions that are never called by the contract should be declared

  Functions:
  #L179 createStream()

  Status: Acknowledged

- **SafeERC20** library imports the SafeMath library which is not being

  Recommendation: Consider removing the unused code/libraries to save gas.

  Status: Acknowledged

- The contract uses **assert** and **require** as error handling functions.

  assert should be used to test internal errors or for analyzing invariants whereas **require** can be used to test inputs, contract state variables and return values from calls to external contracts.

  It should be noted that **assert(false)** reverts the changes **by consuming all the remaining gas**, hence should be used according to the business logic of the contract.

  References:
  1. https://docs.soliditylang.org/en/v0.5.17/control-structures.html#error-handling-assert-require-revert-and-exceptions
  2. https://swcregistry.io/docs/SWC-110
  3. https://swcregistry.io/docs/SWC-123
  4. https://ethereum.stackexchange.com/questions/15166/difference-between-require-and-assert-and-the-difference-between-revert-and-thro

  Status: Acknowledged

- **block.timestamp** has been used for comparisons

  Recommendation: Avoid using block.timestamp, as it can be manipulated by miners

  Status: Acknowledged

Functional test

Function Names	Testing results
getStream	PASS
deltaOf	PASS
balanceOf	PASS
createStream	PASS
cancelStream	PASS
withdrawFromStream	PASS

Results
No major issues were found. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of TokenStream.App. No major issues have been reported during the audit. Suggestions are also provided in order to improve the code quality. Overall The smart contract is well written and documented.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **TokenStream.App**. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the **TokenStream.App** Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
November, 2021

For

Token Stream

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com