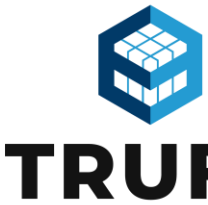

Smart Contract Audit Report

for

TCV Platform

Preliminary Comments

September, 2024

Contents

1 Introduction

Trufy has been engaged by what to perform a security audit of the TCV Platform smart contracts. The purpose of this audit is to achieve the followings:

- Ensure that smart contract functions work as intended.
- Identify possible vulnerabilities, which could be exploited by an attacker.
- Identify smart contract bugs, which might lead to unexpected behavior.
- Make recommendations to improve code safety and readability.

Our results show that the given version of smart contracts is well-designed.

1.1 About TCV

TCV is web3's trustless market making infrastructure protocol that enables running sophisticated algorithmic strategies for liquidity provision on DEX V3 & Lending pool on multi-chains. Liquidity providers can utilize TCV Vaults to have their liquidity be managed in an automated, capital-efficient, non-custodial and transparent manner. Particularly, TCV enables the execution of sophisticated algorithmic strategies for liquidity provision on DEX V3 and lending pools across multiple blockchain networks. This advanced protocol leverages automation to streamline and optimize the liquidity management process, ensuring that LPs can maximize their returns with minimal effort and risk.

1.1.1 Overview of Contracts

The core contracts allow users to:

- Manage and transfer a vault's token pair holdings to/from Uniswap V3 liquidity positions through manager accounts.
- Set and control vault parameters (e.g., manager, Uniswap V3 pools, tick range) via the vault owner role.

1.1.1.1 TCV Vault The ERC20 TCV Vault is the central contract for liquidity management. It collects assets for a token pair and delegates a manager to deploy capital across various Uniswap V3 LP positions.

1.1.1.2 TCV Factory This factory contract deploys TCV vaults. It allows users to create vaults for any token pair and configure the owner, manager, and initial parameters when calling `deployVault`.

1.1.1.3 TCV Router The TCV Router contract handles user interactions, including: - Adding liquidity to a TCV vault. - Removing liquidity from a TCV vault.

1.2 Project Summary

- Project Name: TCV
- Language: Solidity
- Codebase:
- Commit: 10598e8dfa74746fb001eb0b20ddb164e81d7e0b
- Audit method: Static Analysis, Manual Review
- Scope:
  - contracts/core/TCV.sol
  - contracts/core/abstract/TCVStorage.sol
  - contracts/periphery/TCVRouter.sol

1.3 Vulnerability Summary

Severity	# of Findings
Critical	0
Medium	2
Low	1
Informational	2

2 Findings

ID	Title	Type	Severity
ID-01	Redundant for loop	Coding Style	**Informational**
ID-02	Redundant action	Coding Style	**Informational**
ID-03	Lack of Slippage Control	Logical Issue	**Medium**
ID-04	Approving more fund than necessary	Logical Issue	**Low**
ID-05	Lack of initialization for state variable	Logical Issue	**Medium**

3 Detailed Results

3.1 ID-01: Redundant for loop

Type	Severity	Location
Coding Style	**Informational**	TCV.rebalance(Rebalance calldata rebalanceParams_) (TCV.sol#218)

3.1.1 Description

Detect redundant for loop because rebalanceParams_.burns.**length** is equal to 1 and the ele- ment at index 0 rangeToTokenIds[0] is always used.

```
218     for (uint256 i; i < rebalanceParams_.burns.length; i++) {
219         uint256 tokenId = rangeToTokenIds[0];
```

3.1.2 Recommendation

Remove the for loop.

3.2 ID-02: Redundant action

Type	Severity	Location
Coding Style	**Informational**	TCV.rebalance(Rebalance calldata rebalanceParams_) (TCV.sol#242)

3.2.1 Description

Detect redundant action burning the `tokenId` when `liquidity == 0` at TCV.sol#242 because in the code below, after withdrawing all `liquidity`, it is always equal to 0 and the `tokenId` is burned again.

```
242         if (liquidity == 0)
243             INonfungiblePositionManager(nftManager).burn(tokenId);
```

3.2.2 Recommendation

Remove the extra burn operation when `liquidity == 0` to avoid redundant actions.

3.3 ID-03: Lack of Slippage Control

Type	Severity	Location
Logical Issue	**Medium**	TCV.rebalance(Rebalance calldata rebalanceParams_) (TCV.sol#262)

3.3.1 Description

When calling _withdraw function, the input parameters amount0Min_ and amount1Min_, which are the minimum amount of token0 and token1 that should be accounted for the burned liquidity, are both set to 0. This means there is no control over the slippage of burning liquidity. This can lead to a loss of funds if the slippage is higher than expected.

3.3.2 Recommendation

Set amount0Min_ and amount1Min_ to reasonable values. These values can be passed as inputs in rebalanceParams_.swap to ensure better slippage control.

3.4 ID-04: Approving more fund than necessary

Type	Severity	Location
Logical Issue	Low	TCV.rebalance(Rebalance calldata rebalanceParams_) (TCV.sol#298)

3.4.1 Description

Detect that the approve function is called with the whole balance of TCV contract for token0 and token1. This can lead to unexpected behavior if the approved address, which is the swap router, is compromised.

```
298     for (uint256 i; i < rebalanceParams_.burns.length; i++) {
299         uint256 tokenId = rangeToTokenIds[0];
```

3.4.2 Recommendation

Only approve the specific amount required for the transaction to mitigate risks.

3.5 ID-05: Lack of initialization for state variable

Type	Severity	Location
Logical Issue	Medium	TCVStorage.sol#46

3.5.1 Description

The state variable _ranges stored in the TCVStorage contract does not have a setter function as well as not initialized inside any constructor. The only way to update it without error is inside TCV.rebalance() function, when calling the function without burning any liquidity. In that case, the range is created with rebalanceParams_.mints parameters and pushed into _ranges. It should be noted that both function TCV.mint() and TCV.burn() will lead to unexpected behavior if the _rangesis not initialized before calling them.

3.5.2 Recommendation

Add a setter function for _ranges.

4 Appendix

4.1 Severity Definitions

Critical

This level vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.

Medium

This level vulnerabilities are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to critical-risk severity.

Low

This level vulnerabilities should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution.

Informational

This level vulnerabilities can be ignored. They are code style violations and informational statements in the code. They may not affect the smart contract execution.

4.2 Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.