SALUS SECURITY

JUNE 2023

CODE SECURITY ASSESSMENT

TEAHOUSE FINANCE

Overview

Project Summary

- Name: Teahouse Finance - TeaVaultV3Pair
- Version: commit 9873e75
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
  - https://github.com/TeahouseFinance/TeaVaultV3Pair
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Teahouse Finance - TeaVaultV3Pair
Version	v2
Type	Solidity
Dates	June 14 2023
Logs	June 7 2023; June 14 2023

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	3
Total Low-Severity issues	2
Total informational issues	5
Total	10

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Uniswap V3 pool may not be initialized for the tokens	Medium	Data Validation	Resolved
2	feeConfig is not set during initialization	Medium	Business Logic	Resolved
3	Centralization risk	Medium	Centralization	Mitigated
4	Implementation contract could be initialized by everyone	Low	Business Logic	Resolved
5	The multicall function can be used to steal funds in the contract	Low	Business Logic	Resolved
6	Users can inflate the share price by transferring tokens into the contract	Informational	Business Logic	Acknowledged
7	Use of floating pragma	Informational	Configuration	Partially Resolved
8	Inconsistent comments	Informational	Code Quality	Resolved
9	Cache array length outside of loop	Informational	Gas Optimization	Resolved
10	Initialize variables with default value	Informational	Gas Optimization	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Uniswap V3 pool may not be initialized for the tokens	
Severity: Medium	Category: Data Validation
Target: - contracts/TeaVaultV3Pair.sol	

Description

The variable pool is initialized by calling factory.getPool() for tokens. However, there may be a possibility that no pool exists in Uniswap v3 for tokens. In that case, the pool variable will be assigned zero address and it won't be possible to modify it later.

contracts/TeaVaultV3Pair.sol:L84

```
pool = IUniswapV3Pool(factory.getPool(_token0, _token1, _feeTier));
```

Recommendation

Make sure that pool is not zero address and put a check to ensure that.

Status

This issue has been resolved by the team with commit f7797f9.

2. feeConfig is not set during initialization

Severity: Medium	Category: Business Logic
Target: - contracts/TeaVaultV3Pair.sol	

Description

The state variable feeConfig is not set during initialization and it can only be set via setFeeConfig function. If it is not set, users can deposit without paying entryFee or withdraw without paying exitFee.

If the owner calls setFeeConfig function via a separate transaction, users can front-run that transaction to avoid paying the fee.

Recommendation

Consider setting the feeConfig variable to an initial value within the initialize function.

Status

This issue has been resolved by the team with commit f7797f9.

3. Centralization risk	
Severity: Medium	Category: Centralization
Target: - contracts/TeaVaultV3Pair.sol	

Description

There are some privileged roles in the TeaVaultV3Pair contract.

The owner of the TeaVaultV3Pair contract can update the contract logic, change the fee configuration and assign the manager. The manager controls funds within the TeaVaultV3Pair contract through privileged functions.

Since there is no cap on fee configuration, if the owner's private key is compromised, an attacker could set feeConfig.vault to an address he controls, set all fee configurations to the maximum, and assign the manager to himself. If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

Consider transferring the privileged roles to multi-sig accounts and setting a hard cap on fee configuration.

Status

This issue has been mitigated by the team with commit f7797f9. The team has added a comment to recommend using a multi-sig account for the owner and set the FEE_CAP during initialization.

4. Implementation contract could be initialized by everyone

Severity: Low	Category: Business Logic

Target:
- contracts/TeaVaultV3Pair.sol

Description

According to OpenZeppelin, the implementation contract should not be left uninitialized.

An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy. There is nothing preventing the attacker from calling the initialize() function in TeaVaultV3Pair's implementation contract.

Recommendation

To prevent the implementation contract from being used, consider invoking the _disableInitializers function in the constructor of the TeaVaultV3Pair contract to automatically lock it when it is deployed.

Status

This issue has been resolved by the team with commit f7797f9.

5. The multicall function can be used to steal funds in the contract

Severity: Low	Category: Business Logic
Target: - contracts/TeaVaultV3PairHelper.sol	

Description

There is a refund of ETH and vault-related tokens at the end of the multicall function. If TeaVaultV3PairHelper has surplus tokens, not only the owner can withdraw them using rescueFund(), users can also withdraw them using multicall().

Recommendation

Consider adding a note about this issue to the comments and documentation.

Status

This issue has been resolved by the team with commit f7797f9. The team has added a notice to remind users not to transfer tokens to the contract.

2.3 Informational Findings

6. Users can inflate the share price by transferring tokens into the contract	
Severity: Informational	Category: Business Logic
Target: - contracts/TeaVaultV3Pair.sol	

Description

The amount of tokens that users need to deposit is based on the balance of the contract and how many shares they want to mint. Thus, if someone directly transfers some tokens to the contract, the share price may be inflated.

Take the first depositor as an example. The first depositor can deposit a very minimal amount of tokens, such as 1 wei, followed by sending a large amount of funds directly to the contract. This inflates the share price, so subsequent depositors must put in larger deposits.

Recommendation

Consider adding a note to the comments and documentation.

Status

This issue has been acknowledged by the team.

7. Use of floating pragma

Severity: Informational	Category: Configuration
Target: - All	

Description

```
pragma solidity ^0.8.0;
```

The TeaVaultV3Pair contracts use a floating compiler version ^0.8.0.

Using a floating pragma is discouraged, as code may compile to different bytecodes with different compiler versions. Use a locked pragma statement to get a deterministic bytecode. Also use the latest Solidity version to get all the compiler features, bug fixes and optimizations.

Recommendation

It is recommended to use a locked Solidity version throughout the project. It is also recommended to use the most stable and up-to-date version.

Status

This issue has been partially resolved by the team with commit f7797f9. The team does not lock the pragma version in ITeaVaultV3Pair.sol, ITeaVaultV3PairHelper.sol and VaultUtils.sol.

8. Inconsistent comments

Severity: Informational	Category: Code Quality
Target: - contracts/interface/ITeaVaultV3Pair.sol	

Description

This comment is copied from the above getToken0Balance function but forgot to change token0 to token1.

contracts/interface/ITeaVaultV3Pair.sol:L72-L74

```
/// @notice get vault balance of token0
/// @return amount vault balance of token0
function getToken1Balance() external view returns (uint256 amount);
```

Recommendation

Consider fixing the mismatch between comments and implementations.

Status

This issue has been resolved by the team with commit f7797f9.

9. Cache array length outside of loop

Severity: Informational	Category: Gas Optimization
Target: - contracts/TeaVaultPair.sol	

Description

contracts/TeaVaultV3Pair.sol: L607

```
for (uint256 i = 0; i < positions.length; i++)
```

contracts/TeaVaultV3Pair.sol: L621

```
for (uint256 i = 0; i < positions.length; i++)
```

contracts/TeaVaultV3Pair.sol: L632

```
for (uint256 i = 0; i < positions.length; i++)
```

The solidity compiler will always read the length of the array from storage during each iteration. The length of the array can be cached in a local variable to save gas.

Recommendation

Consider caching the length of the array in a local variable stored on the stack.

Status

This issue has been resolved by the team with commit f7797f9.

10. Initialize variables with default value

Severity: Informational	Category: Gas Optimization

Target:
- contracts/TeaVaultPair.sol

Description

contracts/TeaVaultPair.sol:L200

```
for (uint256 i = 0; i < positionLength; i++)
```

contracts/TeaVaultPair.sol:L225-226

```
uint256 entryFeeAmount0 = 0;
uint256 entryFeeAmount1 = 0;
```

contracts/TeaVaultPair.sol:L263

```
uint256 exitFeeAmount = 0;
```

contracts/TeaVaultPair.sol:L367

```
for (uint256 i = 0; i < positionLength; i++)
```

contracts/TeaVaultPair.sol:L399

```
for (uint256 i = 0; i < positionLength; i++)
```

contracts/TeaVaultPair.sol:L426

```
for (uint256 i = 0; i < positionLength; i++)
```

contracts/TeaVaultPair.sol:L595

```
for (uint256 i = 0; i < data.length; i++)
```

contracts/TeaVaultPair.sol:L607

```
for (uint256 i = 0; i < positions.length; i++)
```

contracts/TeaVaultPair.sol:L632

```
for (uint256 i = 0; i < positions.length; i++)
```

There are some variables that are initialized with a default value of 0, which consumes extra gas.

Recommendation

Consider optimizing gas consumption by avoiding unnecessary initialization of variables.

Status

This issue has been resolved by the team with commit f7797f9.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 9873e75:

File	SHA-1 hash
contracts/TeaVaultV3Pair.sol	1880bd0aaf5488dd9584f2f06cadfd3751f99844
contracts/library/GenericRouter1Inch.sol	e3809f66efa8380906f8b0213fdd148f5887794a
contracts/library/VaultUtils.sol	1354ca7f5091b046fb0bbddabad15f9093fa980d
contracts/TeaVaultV3PairHelper.sol	1ec4a2e015ba1351d13403903460373c83f3375e
contracts/interface/IGenericRouter1Inch.sol	b694e28df1714db5e00de250a559336a94947d73
contracts/interface/IWETH9.sol	9a023de828c44e27c52a9506d8f850fdabde9d97
contracts/interface/ITeaVaultV3Pair.sol	6839fab6d3f369505a12c4328b4a54cf42216cb9
contracts/interface/ITeaVaultV3PairHelper.sol	309b5878ce2ee74eac2e0815d1b91656a88cb1e3