0x Guard

Smart contracts security assessment

Final report

Tariff: Standard

Tulip.money

April 2022

0xguard.com

hello@0xguard.com

Contents

Introduction

This report has been prepared for the Tulip.money team upon their request.

The audited project is a fork of the Tomb Finance Project. The code is available in the Github repository. The code was checked in 3632044 commit.

The purpose of this audit was to ensure that no issues were introduced with the changes to the original code and that known vulnerabilities (e.g. circumventing the protocol's fee system) are fixed prior to deployment.

Further details about Tulip.money are available at the official website: http://tulip.money.

Name	Tulip.money
Audit date	2022-04-08 - 2022-04-16
Language	Solidity
Platform	Oasis

Contracts checked

Name	Address
Tulip	https://github.com/moneytulip/tulip-v2-contracts/blob/363204487b9ec46f76c91e4826dd103324254554/contracts/Tulip.sol
Petal	https://github.com/moneytulip/tulip-v2-contracts/blob/363204487b9ec46f76c91e4826dd103324254554/contracts/Petal.sol

Bud	https://github.com/moneytulip/tulip-v2-contracts/blob/363204487b9ec46f76c91e4826dd103324254554/contracts/Bud.sol
PetalRewardPool	https://github.com/moneytulip/tulip-v2-contracts/blob/363204487b9ec46f76c91e4826dd103324254554/contracts/PetalRewardPool.sol
TulipGenesisRewardPool	https://github.com/moneytulip/tulip-v2-contracts/blob/363204487b9ec46f76c91e4826dd103324254554/contracts/TulipGenesisRewardPool.sol
TulipRewardPool	https://github.com/moneytulip/tulip-v2-contracts/blob/363204487b9ec46f76c91e4826dd103324254554/contracts/TulipRewardPool.sol
Oracle	https://github.com/moneytulip/tulip-v2-contracts/blob/363204487b9ec46f76c91e4826dd103324254554/contracts/Oracle.sol
Garden	https://github.com/moneytulip/tulip-v2-contracts/blob/363204487b9ec46f76c91e4826dd103324254554/contracts/Garden.sol
Treasury	https://github.com/moneytulip/tulip-v2-contracts/blob/363204487b9ec46f76c91e4826dd103324254554/contracts/Treasury.sol
TokenSale	https://github.com/moneytulip/tulip-v2-contracts/blob/363204487b9ec46f76c91e4826dd103324254554/contracts/TokenSale.sol

RebateTreasury	https://github.com/moneytulip/tulip-v2-contracts/blob/363204487b9ec46f76c91e4826dd103324254554/contracts/RebateTreasury.sol
Multiple contracts	

Procedure

We perform our audit according to the following procedure:

Automated analysis

- Scanning the project's smart contracts with several publicly available automated Solidity analysis tools
- Manual verification (reject or confirm) all the issues found by the tools

Manual audit

- Comparing the project to the Tomb Finance implementation

Classification of issue severity

High severity	High severity issues can cause a significant or full loss of funds, change of contract ownership, major interference with contract logic. Such issues require immediate attention.
Medium severity	Medium severity issues do not pose an immediate risk, but can be detrimental to the client's reputation if exploited. Medium severity issues may lead to a contract failure and can be fixed by modifying the contract state or redeployment. Such issues require attention.
Low severity	Low severity issues do not cause significant destruction to the contract's functionality. Such issues are recommended to be taken into consideration.

Issues

High severity issues

No issues were found

Medium severity issues

1. Commission tokens (TulipGenesisRewardPool)

In 214-219L when transferring a commission token, the same commission is charged. Token `wRose` set may have different commissions or change them, so the calculation of `user.amount` can be violated.

Recommendation: It is recommended to compare the balance of the token before and after the execution of the `pool.token.safeTransferFrom()` function, thus you will find out how much is spent on the commission.

2. Unlimited fee (TulipGenesisRewardPool)

There is no check for the `_fee` input value in the `add()` function. Thus, the operator can put a 100% commission on the withdrawal of funds.

Recommendation: It is recommended to add a check for the `_fee` input value so that it falls within a certain range of values.For example: `require(_fee > 0 && _fee <= 500, "_fee in range")`.

3. Contract ownership (Multiple contracts)

The `governanceRecoverUnsupported()` function (found in the `Tulip`, `Petal`, `PetalRewardPool`, `TulipRewardPool` and `TulipGenesisRewardPool` contracts) can remove all tokens from the contract balance if the operator role is compromised.

Recommendation: There is a large number of functions with the `onlyOperator()` modifier, there is a possibility that the operator can be compromised. It is recommended to create multiple roles for different kinds of functions to reduce the operator's influence. It is also recommended to add a time delay to the especially important set functions using the TimelockController. We also recommend that you look through the entire codebase to find functions that are dangerous for you as the owner of the project (mainly set functions), if there are any, add a call to them via a multisig wallet. This helps to avoid the issue of owner compromise.

Low severity issues

1. Unused memory variable (Tulip)

The variable `currentTaxRate` on 153L is not used.

Recommendation: It is recommended to remove this variable to optimize gas usage.

2. Unused state variables (Tulip)

State variables `burnThreshold`, `autoCalculateTax`, `excludedAddresses`, `taxRate` and `taxCollectorAddress` are not used in this contract.

Recommendation: It is recommended to remove all references to these variables. You can also remove the associated set functions - `setTaxCollectorAddress()`, `setBurnThreshold()`, `excludeAddress()` and `includeAddress()`.

3. Same functions (Petal)

There are two identical `setDevFund()` functions on 57L and 63L.

Recommendation: Remove all duplicate functions.

4. Reentrancy attack (PetalRewardPool)

When withdrawing, some pool tokens may be subject to a reentrancy attack. The variable `user.rewardDebt` on 230L is updated after calling `pool.token.safeTransfer()`.

Recommendation: It is recommended to update the value of the `user.rewardDebt` variable before calling `pool.token.safeTransfer()`.

5. Reentrancy attack (TulipGenesisRewardPool)

When withdrawing, some pool tokens may be subject to a reentrancy attack. The variable `user.rewardDebt` on 245L is updated after calling `pool.token.safeTransfer()`.

Recommendation: It is recommended to update the value of the `user.rewardDebt` variable before calling `pool.token.safeTransfer()`.

6. Reentrancy attack (TulipRewardPool)

When withdrawing, some pool tokens may be subject to a reentrancy attack. The variable `user.rewardDebt` on 233L is updated after calling `pool.token.safeTransfer()`.

Recommendation: It is recommended to update the value of the `user.rewardDebt` variable before calling `pool.token.safeTransfer()`.

7. Variable should be immutable (TokenSale)

The state variable of `tokenContract` has to be immutable, since it doesn't change anywhere.

Recommendation: Make the `tokenContract` variable immutable.

8. Deprecated assert (TokenSale)

Assert is a deprecated expression in Solidity. Require fully replaces while spending less gas.

Recommendation: Replace the assert statement on 35L with the require statement.

9. Few events (Multiple contracts)

Many set functions from contracts are missing events when changing important values in the contract.

Recommendation: Create events for these set functions.

Conclusion

0 high, 3 medium, 10 low severity issues were found.

The Tulip.money Project was compared to the Tomb Project. Tulip.money has changed the implementation of the `TulipGenesisRewardPool` contract. New contracts have been added: TokenSale, RebateTreasury.

In the `TulipGenesisRewardPool` contract, the `fee` field has been added to the pool structure, which is used to calculate the commission when withdrawing funds.

The changed contract is not affected by the vulnerability that was discovered in the Tomb before because it doesn't contain the implementation of transfer with taxes.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability)set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without 0xGuard prior written consent.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xGuard to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

x Guard