quantstamp / **Lendroid_Review** Private

Branch: master ▾ Lendroid_Review / **simple_lst_distribution.md** Find file Copy path

kbak typo fix 9f4a91c 23 hours ago

2 contributors

```
195 lines (138 sloc)   11 KB
```

Overview

This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

This security audit report follows a generic template. Future Quantstamp reports will follow a similar template and they will be fully generated by automated tools.

Specification

Our understanding of the specification was based on the following documentation:

- TGE and TRS specification document, and
- the below specification provided via email by Lendroid:
  - LST tokens are minted seperately and the total amount required for the TGE withdrawals is transferred into this contract.
  - All data resides in two contracts: 1) `SimpleTGE` (deployed) and 2) `SimplePreTGE` (to be deployed). Data in both the contracts is held in the following data structures:

    ```
    mapping (address => Contribution) public contributions;
    struct Contribution {
      bool hasVested;
      uint256 weiContributed;
    }
    ```

  - The contract `SimpleLSTDistribution.sol` tracks users' allocations in the following data structures:

    ```
    struct allocation {
      bool hasVested;
      uint256 weiContributed;
      uint256 LSTAllocated;
      // indicates whether the user has already withdrawn their funds
      bool hasWithdrawn;
    }
    mapping (address => bool) public allocations
    ```

  - The contract `SimpleLSTDistribution.sol` is a withdrawal contract and the function `withdraw()` can be run successfully only once by each user: `require(!allocations[msg.sender].hasWithdrawn)`.
  - If the user has not vested, his tokens are transferred to him immediately.
  - If the user has vested:

        - 10% of his allocation are transferred to him.
        - 90% is put into a ERC20 Zeppelin `TokenVesting` smart contract.
    - If the user decided to vest in either the contract `SimplePreTGE` or the contract `SimpleTGE` , then their contributions in both phases will be combined and will follow vesting rules.
    - There is a hard cap of 12 billion tokens.
    - Tokens should be paused on contract creation and not available for transfers until manual action is taken by the Lendroid team.

Methodology

The review was conducted during 2017-Feb-19 through 2017-Feb-26 by Richard Artoul and the Quantstamp team, which included senior engineers Kacper Bak and Steven Stewart.

Their procedure can be summarized as follows:

0. Code refactoring
1. Code review
    - Review of the specification
    - Manual review of the code
    - Comparison to the specification
2. Testing and automated analysis
    - Test coverage analysis
    - Symbolic execution (automated code path evaluation)
3. Best-practices review
4. Itemize recommendations

Source code

The following source code was reviewed during the audit.

Repository	Commit
tge-contracts	49a38a9

Security Audit

Quantstamp's objective was to evaluate the Lendroid LST Distribution contract (and supporting contracts) for security-related issues, code quality, and adherence to best-practices.

Possible issues include (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrance and cross-function vulnerabilities

- Denial of service / logical oversights

Test coverage

We evaluated the test coverage using `truffle` and `solidity-coverage`. The below notes outline the setup and steps that we performed.

Setup

Testing setup:

- Truffle v4.0.6
- TestRPC 6.0.7
- solidity-coverage 0.4.9
- Oyente 0.2.7
- Mythril 0.11.1

Steps

Steps taken to run the full test suite:

- Commented out the tests for the existing contract `SimpleTGE`.
- Ran the coverage tool `solidity-coverage`.

Evaluation

The coverage results of the `simpleLSTDistribution.sol` file:

```
100% Statements
91.67% Branches
100% Functions
100% lines
```

The coverage results of the `SimplePreTGE.sol` file:

```
100% Statements
50% Branches
100% Functions
100% Lines
```

In both cases, the missing branch coverage appears to be due to missing coverage for the `else` paths of the `require()` statements.

The coverage results of the `LendroidSupportToken.sol` file:

```
100% Statements
100% Branches
100% Functions
100% Lines
```

Symbolic execution (the Oyente tool) did not detect any vulnerabilities of types Parity Multisig Bug 2, Callstack Depth Attack, Re-Entrancy Vulnerability, Transfer Concurrency, or Time Dependency.

Oyente reported 99.8% EVM code coverage for the `LendroidSupportToken.sol` contract.

Oyente reported 99.4% EMV code coverage for the `SimplePreTGE.sol` contract.

Oyente reported 31.9% EVM code coverage for the `SimpleLSTDistribution.sol` contract.

Mythril tool reported that the function `release()` is called on the contract `TokenVesting.sol` (which is found in the map `vesting` for a beneficiary whose address is provided as a function parameter). The concern is that the function `release()` may feature an arbitrary behavior. We believe this to be a benign issue because the contract `TokenVesting.sol` and the contract `LendroidSupportToken.sol` are both controlled by the Lendroid team.

Recommendations

Deviation from Specification

1. The function `mintTokens()` of the contract `simpleLSTDistribution.sol` was never mentioned in the specification, and it allows the Lendroid team to mint an arbitrary number of tokens to any user regardless of contribution. Furthermore, it emits the same event `LogLSTsWithdrawn()` as the function `withdraw()` although these two events are conceptually different. We consider these issues serious enough to either update the specification (and, perhaps, inform the community), or remove the function `mintTokens()` from the contract.

2. The specification stated that *LST tokens are minted seperately and the total amount required for the TGE withdrawals is transferred into this contract*. However, in Lendroid's implementation, the token is created and minted by the contract `SimpleLSTDistribution`. We recommend updating either the code or the specification to make both artifacts consistent with each other.

Code Documentation

We noted that the majority of the functions were self-explanatory. Although standard documentation tags (such as `@dev`, `@param`, and `@returns`) were missing inline comments provided sufficient information to clarify most of the code. We recommend, however, adding documentation to the function `withdraw()` to clarify its logic.

Code Restructuring

In `SimplePreTGE.sol`, the function `disableAllocationModificationsForEver()` shall return `true` on successful completion or not return anything at all. Currently it returns `false` on success, which is not a common practice.

Naming Conventions

We detected a few casing inconsistencies. For example, the structure `allocation` (in the file `simpleLSTDistribution.sol`) should be capitalized. The method `disableAllocationModificationsForEver` (in the file `SimplePreTGE.col`) should be named `disableAllocationModificationsForever()`. None of these issues affect the security or effectiveness of the contract, but adhering to naming conventions is considered best practice.

Appendix

File Signatures

Below are SHA256 file signatures of the relevant files reviewed in the audit.

```
$ shasum -a 256 ./contracts/* ./contracts/*/* ./contracts/*/*/*
add108e970664f85d69c5dbd8d13af075e3550c3f75c10767eb172984b290f69  contracts/LendroidSupportToken.sol
98a2f9cf3f6d74d71d23374f6b118f9a4ecc12e0b2b701f3b7ba1fb7d5bc8026  contracts/Migrations.sol
089789cf707d0b4c85d64c3dc9f0690620e056eb3984359fd1cbcc4a9b7f9c41  contracts/SimplePreTGE.sol
84508755afb802b570b1c797cb67b66f8763a3b6b8a27e3a85987300e1608387  contracts/simpleLSTDistribution.sol

$ shasum -a 256 ./test/* ./test/*/*
810ef08fab9ed70223ee8ffec1c90fee1c143022cf5c31e7d71389b2bf489885  test/simple_LST_distribution.js

$ shasum -a 256 ./migrations/*
42c21b4229b39fd1cad164ed6d4c24168620e2f04a66521b2b2f2945e23b867d  migrations/1_initial_migration.js
```

Disclosure

Purpose of report

The scope of our review is limited to a review of Solidity code and only the source code we note as being within the scope of our review within this report. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks.

The report is not an endorsement or indictment of any particular project or team, and the report does not guarantee the security of any particular project. This report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset.

No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp Technologies Inc. (QTI). Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that QTI are not responsible for the content or operation of such web sites, and that QTI shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that QTI endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. QTI assumes no responsibility for the use of third-party software on the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by QTI; however, QTI does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.