Tend ICO Audit

by MonteLabs

November 21, 2017

1 Introduction

The Tend ICO consists of two parts, (i) the Crowdsale and (ii) the Tend token. This audit checks for security and correctness issues in both parts, as well as in their interfaces. The code was written by Validity Labs, and is hosted at `github.com/validitylabs/ico-tend`. The verified version is commit d7da1138252580671893b9cdbc1f79d14fa0bc59 from November 14, 2017.

This document is a report containing our findings while analysing and testing the code. Our suggestions were taken into account by the developers and the corrected code is commit e3f0b6a7550e19b617c0b33bf3fa42f572581099 from November 18, 2017.

We have analysed the three new smart contracts that were created:

- IcoCrowdsale.sol;
- DividendToken.sol;
- IcoToken.sol.

These contracts use OpenZeppelin smart contract libraries which are considered standard and have been heavily audited and used by the community in general. Figure 1 shows the relation between the used smart contracts, where the red nodes are the OpenZeppelin libraries and the green nodes are the newly implemented smart contracts, object of this audit.

The code has a very high quality, employs good practices and contains many important tests. Section 2 lists the issues found during this audit for the Crowdsale and the Tend token, separated by severity level (High, Medium or Low) or as minor suggestions to improve code readability or remove redundant code. For each issue we suggest a solution. The new tests, scripts and code fixes can be found at `github.com/montelabs/ico-tend`. Section 3 presents our concluding thoughts on the audit.

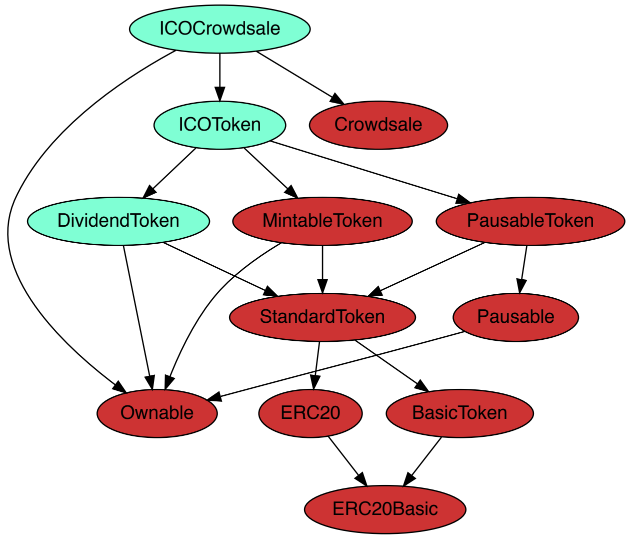

Figure 1: Relationship between the smart contracts used in the Tend ICO.

2 Issues

2.1 High severity

Tend token owner can mint arbitrary amounts of token after the crowdsale.

In the end of function `finalize()` at `IcoCrowdsale.sol:265`, the ownership of the Tend token contract is transferred from the Crowdsale contract to its owner (`Ownable(token).transferOwnership(owner)` at `IcoCrowdsale.sol:278`). After that, the owner can call any function from `MintableToken.sol`, including `mint(address _to, uint256 _amount)` at `MintableToken.sol:34`, and generate arbitrary amounts of token to anyone, including themselves. That potentially violates the minted cap requirements for the token.

The new test “`should not mint more tokens after finalize()`” was created in `IcoCrowdsale.js` to catch this issue.

The suggested fix is adding the line `MintableToken(token).finishMinting();` right before the ownership transfer.

2.2 Medium severity

Exchange rate fluctuation.

Since the exchange rate for the token is fixed in CHF, and the rate is not mutable (`IcoCrowdsale.sol:87`), the crowdsale may be affected by severe variations in the conversion between CHF and Ether.

A potential solution would be to use an oracle that can be invoked in fixed intervals to update the rate value from the `IcoCrowdsale`. The oracle can be implemented with Oraclize, querying already known exchanges, or use the average rate among them.

After contacting Tend and Validity Labs, they have informed us that this decision was made consciously to avoid adding more complexity to the smart contracts.

2.3 Low severity

OpenZeppelin library files are not checked.

The `zeppelin-solidity` contracts are hard-coded in the beginning of the contracts using a dynamic folder, downloaded when installing modules. There is an unlikely possibility that the OpenZeppelin's code base is compromised. In that case, the files that are downloaded are compromised, thus the ICO is compromised.

A possible solution would be to write scripts able to attest that the OpenZeppelin files are correct. We provide a script that runs after the modules installation, to check the used OpenZeppelin's files integrity.

Users can launder money if owner fails to request unclaimed dividends before a new Payin.

If the owner fails to request the unclaimed dividends and a treasurer sends a new Payin, the new claims from the holders will give them more than they deserve according to the amount of tokens they hold. This also happens if an arbitrary smart contract destroys itself and sends its remaining balance to the Tend token. This could lead to uneven accounting, and malicious users could try to use this for money laundering.

The new test "`should increase the owner's balance, because token balance is not 0 while doing a Payin. Token balance should be the same as the Payin afterwards`" was added to `IcoToken.js` to catch this issue.

The suggested fix is to enforce that users cannot get more dividends than the fair amount, by adding the line

```
if (this.balance > msg.value) owner.transfer(this.balance - msg.value);
```

in the beginning of the fallback function (Payin) in `DividendToken.sol`. This would guarantee that the contract's remaining balance (before the Payin) is transfered to the owner, as calling the `requestUnclaimed` function.

Timestamp dependency.

Both the crowdsale and the token contracts rely on timestamp (`now`) for their internal logic. The timestamp of a block can be manipulated by a malicious miner by a few hours. We do not consider that an offensive action can be taken against any of the two contracts using this technique, since the smallest time unit used in the contracts is `days`.

2.4 Minor suggestions

- Function `claimDividend` from `DividendToken.sol` can save gas by returning if `payment` is 0;
- In `IcoCrowdsale.sol:167,176,190`, the code `now <= endTime.add(confirmationPeriod)` is unnecessary because it is already checked in the above `require`.

3 Conclusion

The Tend ICO code provided by Validity Labs is of very good quality, easy to read, and their thorough test suite leaves barely any space for problems.

We have found only one severe issue which can be quickly fixed with one line of code. The other lower risk issues can also be fixed without much effort.

The developers followed our suggestions and fixed the main issues.

This document is stored on IPFS as a security evidence of the Tend ICO audit. This audit also implies that MonteLabs issues an on-chain audit verification seal that can be accessed directly via the smart contract or at montelabs.com/audits as soon as the Tend contracts are deployed, where all the security evidences can be seen.