CoinFabrik

Etherparty Token Audit

April 2018

By Coinfabrik

Introduction

Coinfabrik team has been hired to audit Etherparty smart contracts. Firstly, we will provide a summary of our discoveries and secondly, we will show the details of our findings.

Summary

The contracts audited have been taken from the eth-smart-contracts repository at https://github.com/etherparty/eth-smart-contracts/tree/develop. The audit is based on the commit *1587f3a962413c93ebde3c4268572b1bb24c76d3*, and updated to reflect the changes made in commit *8ab2c9dae2b01d662c264eb335729b99df85e1fa* at branch f*eat/coinfabrik_audit_changes*

The audited contracts and their checksums are the following:

```
65275a20a6355647ea18dbd4b391ddde4391ea1bfb8807fe159299045fae837e  Crowdsale.sol
4a119381225068b41f0e5fa4590541f86b31d36ae98263f9fb37c5236e936d6c  Token.sol
```

The Etherparty contracts are templates for the creation of token contracts. They are configured with other scripts that leave some values uninitialized.

No real security problems were found in these contracts, but some minor adjustments are recommended.

The following analyses were performed:

- Misuse of the different call methods: call.value(), send() and transfer().
- Integer rounding errors, overflow, underflow and related usage of SafeMath functions.
- Old compiler version pragmas.
- Race conditions such as reentrancy attacks or front running.
- Misuse of block timestamps, assuming anything other than them being strictly increasing.
- Contract softlocking attacks (DoS).
- Potential gas cost of functions being over the gas limit.
- Function qualifiers missing or misused.
- Fallback functions with higher gas cost than what a transfer or send call allows.
- Fraudulent or erroneous code.
- Code and contract interaction complexity.
- Wrong or missing error handling.
- Overuse of transfers in a single transaction instead of using withdrawal patterns.
- Insufficient analysis of function input requirements.

Detailed findings

Enhancements

Use of emit keyword in events

As of 0.4.21, it is encouraged to use emit when using events, since it improves code legibility and helps differentiate between an event emission and a function call. In the buyTokens() function this is not done. Neither in token.sol nor in moveAllocation().

This was fixed in the latest commit sent to us.

Emission of transfer event from 0x0 at moveAllocation()

It is possible that the emission of the event coming from 0x0 is unintended. This behavior is generally reserved for minting. Given that the allocation is moved from the Crowdfund contract, it is possible that the intended behavior is:

```
emit Transfer(msg.sender, _to, _amount);
```

It was decided that this behavior will not be modified in the future by the development team.

Observations

We will write some of the verifications that we made to the contracts even though no severe problems were found during the audit:

- Misuse of the different call methods: *call.value()*, *send()* and *transfer()*.
  We found no incorrect use of *call()* or *send()*, and *transfer()*.
- Integer rounding errors, overflow, underflow and related usage of SafeMath functions.
  The mathematical operations performed are protected against overflow using requires for input parameters.
- Race conditions such as reentrancy attacks or front running.
  There are 4 possible calls to another contract:
  - When the Crowdfund contract calls the Token to schedule the starting time, or reschedule it.
  - To buy tokens, at the buyTokens function.
  - At the closing of the crowdfund.

  They all call to the known Token contract, so we see no possible usage of it for malicious purposes.
- Misuse of block timestamps, assuming anything other than them being strictly increasing.
  Timestamps are used to start the sale. The start function is only accessible by the owner, and to close the sale. They are also used for validation when buying tokens,

and for getting the rate at which tokens are bought, but the assumption is still that they are strictly increasing.

- Contract softlocking attacks (DoS) / unbounded gas usage.
  No function in the contract has a loop that can be abused to cause a soft lock or an unbounded gas usage.

Conclusion

To sum up, the audited smart contracts have been well documented and easy to follow. Enhancement proposals do not need to be implemented and do not affect the contract security. Nevertheless, we encourage the team to evaluate them and consider if the contracts are working as intended for future contract versions.
No real security problems were found in the contracts.
We also found that automated testing is being used in the contracts by the Etherparty development team, which is a good practice.