<<codecontext.io

Streamr Token Sale Code Review

Version 1.0

1. Introduction

This document summarizes our best effort to review of the provided source code of the Streamr Token & Token Sale solidity smart contracts. We were given the following deployed contracts:

https://etherscan.io/address/0x0Cf0Ee63788A0849fE5297F3407f701E122cC023#code
https://etherscan.io/address/0xfD2eB46FF56D785B64f1567e53083819c7F0168B#code
https://etherscan.io/address/0x1bb7804d12fa4f70ab63d0bbe8cb0b1992694338#code
https://etherscan.io/address/0x6Cd81d35B3B8dc27A54ae20bAeC956586383D34B#code
https://etherscan.io/address/0x37662fbdc707f7b23bef62a807923f12c8d70dca#code
https://etherscan.io/address/0xd2ee82eaf419e71dfca38b524aab589a0800acf8#code

2. About this review

This is a source code security audit: only security considerations have been annotated in this report. We did not review:

- Contracts functional design
- Design, deployment, operation and contingency plan of the ICO
- EVM code generated
- Credibility or Profitability of the team, idea, or any investment in general.

Critically analyse the provided information: code reviews are hard, and some of the analysis provided in this code review could be erroneous or coming from a misinterpretation of the operations done in these smart contracts.

This review is an opportunity to re-read the code: Instead of checking only the points proposed, consider re-reading all the code to find additional issues.

Bug bounties and multiple security reviews are a must: Code reviews are not enough to ensure the safety of a smart contract, a public bug bounty is essential, as it gives the community at large a chance to participate in reviewing the overall security of the code. A great example is the Aragon Bug Bounty Program.

3. Review results

3.1. General considerations

In general, the code follows all the security measures and best practices expected in when writing solidity smart contracts that will hold user funds:

- Skilled developers with strong knowledge of the Ethereum security domain
- Well crafted state transitions
- Safe coding patterns
- Easy to read, well commented source code
- Usage/wrapping of well known and well reviewed smart contracts from trusted companies without significant modifications[1]

3.2. Critical problems

No critical problems has been found

3.3. Potential problems

- The `MultiSigWallet` used allows non-owners to execute previously failed transactions via `executeTransaction`. This is solved in latest version of the contract.

- In `Crowdsale.preallocate` if for whatever reason you need to change the multisig (using `Crowdsale.setMultisig`, you have a small slot of 'test investments' to check if everything is ok, defined by the constant `Crowdsale.MAX_INVESTMENTS_BEFORE_MULTISIG_CHANGE`. But with the current code, each preallocation counts as an investment, so, it seems that you are not going to be able to change the multisig if you have already called `preallocate` `MAX_INVESTMENTS_BEFORE_MULTISIG_CHANGE` (in code =5) times before

- if you require a signature, but with zero address signer, like `Crowdsale.setRequireSignedAddress(true,0x0)` then users will be able to do a Crowdsale.investWithSignedAddress` using a malformed signature, since `ecrecover` returns zero if there's an error validating the signature.

3.4. Minor issues / comments

- In `StandardToken.transferFrom` you can send to yourself more tokens than your balance.

[1] Even in these cases some bugs could be found later (like the recent parity multisig vulnerability), so a few quick checks through commits have been done to check if any catastrophic bugs have been introduced.

- Since `preallocate` does not check about current state, the Crowdsale owner is able to preallocate tokens even the sale is finished, before calling `finalize()`. We don't know if this is done by design, but is not the expected behaviour if the function is called "preallocate"

- There's a weird thing that is if the Crowdfunding fails and the idea is to return all the funds from the multisig, those funds will never be in the exclusive possession of the original investors, since if always possible to fake an account, event in failed state, using `preallocate`. So to ensure that does not happen, ownership needs to be removed after `loadRefund()` with something like `transferOwnership(0xdead)`

4. Conclusion

The reviewed smart contracts are well crafted, following the expected common security practices. No critical problems have been found. The potential problems require a detailed analysis and could be properly mitigated. We recommend proceeding with a public bug bounty program, as usual in public token offerings.