Miguel's Checklist

Summary

Author: Miguel

Source: Solidity Audit Checklist

Info

Checklist

- ☐ All functions are `internal` except where explictly required to be `public`/`external`. [?]
- ☐ There are no arithmetic overflows/underflows in math operations.
- ☐ Using the OpenZeppelin safe math library [?].
- ☐ Ether or tokens cannot be accidentally sent to the address `0x0`.
- ☐ Conditions are checked using `require` before operations and state changes.
- ☐ State is being set before and performing actions.
- ☐ Protected from reentry attacks (A calling B calling A). [?]
- ☐ Properly implements the ERC20 interface [?].
- ☐ Only using modifier if necessary in more than one place.
- ☐ All types are being explicitly set (e.g. using `uint256` instead of `uint`).
- ☐ All methods and loops are within the maximum allowed gas limt.
- ☐ There are no unnecessary initalizations in the constructor (remember, default values are set).
- ☐ There is complete test coverage; every smart contract method and every possible type of input is being tested.
- ☐ Performed fuzz testing by using random inputs.
- ☐ Tested all the possible different states that the contract can be in.
- ☐ Ether and token amounts are dealt in wei units.
- ☐ The crowdsale end block/timestamp comes after start block/timestamp.
- ☐ The crowdsale token exchange/conversion rate is properly set.
- ☐ The crowdsale soft/hard cap is set.
- ☐ The crowdsale min/max contribution allowed is set and tested.
- ☐ The crowdsale whitelisting functionality is tested.
- ☐ The crowdsale refund logic is tested.
- ☐ Crowdsale participants are given their proportional token amounts or are allowed to claim their contribution.
- ☐ The length of each stage of the crowdsale is properly configured (e.g. presale, public sale).
- ☐ Specified which functions are intented to be controlled by the owner only (e.g. pausing crowdsale, progressing crowdsale stage, enabling distribution of tokens, etc..).
- ☐ The crowdsale vesting logic is tested.
- ☐ The crowdsale has a fail-safe mode that when enabled by owner, restricts calls to function and enables refund functionality.
- ☐ The crowdsale has a fallback function in place if it makes reasonable sense.
- ☐ The fallback function does not accept call data or only accepts prefixed data to avoid function signature collisions.

- [ ] Imported libraries have been previously audited and don't contain dyanmic parts that can be swapped out in future versions which can be be used maliciously. [?]
- [ ] Token transfer statements are wrapped in a `require`.
- [ ] Using `require` and `assert` properly. Only use `assert` for things that should never happen, typically used to validate state after making changes.
- [ ] Using `keccak256` instead of the alias `sha3`.
- [ ] Protected from ERC20 short address attack. [?].
- [ ] Protected from recursive call attacks.
- [ ] Arbitrary string inputs have length limits.
- [ ] No secret data is exposed (all data on the blockchain is public).
- [ ] Avoided using array where possible and using mappings instead.
- [ ] Does not rely on block hashes for randomness (miners have influence on this).
- [ ] Does not use `tx.origin` anywhere. [?]
- [ ] Array items are shifted down when an item is deleted to avoid leaving a gap.
- [ ] Use `revert` instead of `throw`.
- [ ] Functions exit immediately when conditions aren't meant.
- [ ] Using the latest stable version of Solidity.
- [ ] Prefer pattern where receipient withdrawals funds instead of contract sending funds, however not always applicable.
- [ ] Resolved warnings from compiler.