TOKEN SALE AUDIT

INTRO

The Ethic Hub team is working on a decentralised platform that will allow small farmers to get affordable loans for their business. The team asked me to review the Etix Token that will power the platform and the crowdsale contracts that will distribute tokens among investors.

The code is located in the open-source github repository EthicHub/presale. The version of the code that is being reviewed was published as the commit `9797c6aecfc24211893a692320fe69e1086bb4c6`.

All of the issues are classified using the popular OWASP risk rating model. It estimates the severity taking into account both the likelihood of occurrence and the impact of consequences.

SUMMARY

The code is well organized and documented. The repository contains extensive tests suits both on unit and integration level. The EthicHub team went an extra step with implementing the composition design pattern that adds flexibility and reduces coupling among contract modules.

There were no critical errors found.. Only one issue with a high severity was identified. Most of the comments regard edge case scenarios and contain advice that will allow easier maintenance of the crowdsale process in all of the circumstances.

FOUND ISSUES

It's possible to distribute more tokens than a pool size

Severity: **HIGH** (Impact: HIGH, Likelihood: MEDIUM)

The `FixedPoolWithDiscountsTokenDistributionStrategy` allocates tokens in two phases implemented separately in the distributeTokens and compensate functions. In the distributeTokens function there is no validation if the amount of tokens held in the contract are sufficient to compensate participants in return for the contribution. The potential tokens shortage will be discovered in the compensate function which can be called after the crowdsale only. At this moment, all of the contributions are collected and cannot be refunded to the buyers.

The current mechanism protecting against the described scenario is based on estimating the token price and multiplying it by the token pool size to calculate the hard cap. In order to achieve the full safety, the worst case scenario for the price needs to be assumed, so all participants are guaranteed to receive tokens. On the other hand, this approach may lower the number of distributed tokens, if not all of the participants use the maximum discounts.

We recommend using an explicit check if the amount of tokens that are held in the distribution contract is sufficient to satisfy the next contribution. Consider adding in a `line 81` an expression `require(totalContributed <= token.balanceOf(this));`.

Fixed in the commit [9730b2745a9dabb4620de25b705c24c0446aaf19]

Distribution Strategy may be hijacked

Severity: **MEDIUM** (Impact: HIGH, Likelihood: LOW)

The `TokenDistributionStrategy` can be linked to a `_crowdsale` by anyone, who manages to call the initializeDistribution function first. This will create a narrow vulnerability window allowing a malicious agent to gain control over the contract if he manages to submit a transaction before the Ethix team. This may turn out to be profitable, as the `TokenDistributionStrategy` is supposed to hold and allocate all the tokens using the `distributeTokens` function. Therefore miners may be incentivized to reorder transactions to increase the attack vector giving the higher priority to a malicious invocation.

We suggest adding a security check to validate the caller of the initializeDistribution function. It could be done by validating the address of the `owner` of the `_crowdsale` contract.

Fixed in the commit 48da3667193945a97c73d0954003357f7a27748d

Unbound loop

Severity: **MEDIUM** (Impact: HIGH, Likelihood: LOW)

The `vestedAmount` function iterates over the total number of periods, defined in the `numPeriods` variable, to calculate the number of already `vestedPeriods`. Due to the lack of limit for `numPeriods` value, there is a risk of running out of gas if the number of periods is large enough. This will freeze the contract blocking the usage of both the `release` and `revoke` function.

I recommend validating the `_numPeriods` argument in the `IntervalTokenVesting` constructor, in line 56, to assure that it is within the safe bounds. The calculation of the `vestedPeriods` variable can also be optimised by starting the next iteration from the previously calculated `vestedPeriods` value as the time is increased monotonically. An alternative optimization would be dividing the time elapsed from the `start` by the `periodDuration`.

Fixed in the commit 0e74ada43053350f4f62f2a1fae4f9e0d5418ecc

Whitelisted registeredAmount may not be allowed as a bid

Severity: **MEDIUM** (Impact: HIGH, Likelihood: LOW)

The `changeRegistrationStatus` function sets a minimum value of a contribution, called `registeredAmount`, which is required from an investor to earn a preferential rate. However, there is no mechanism to validate, that the minimum value is less than the upper bound for a valid purchase that is defined as `maximumBidAllowed` in the `EthicHubPresale` contract. In case these bounds overlap, an investor won't be able to make a contribution at all.

Consider validating the `registeredAmount` for every whitelisted investor to ensure it is within the bounds defined by the `EthicHubPresale` contract.

Fixed in the commit [550d608eeeaaaee4da6a92175960fbed36179ce4]

Emit Transfer events for the implicit token generation

Severity: **MEDIUM** (Impact: LOW, Likelihood: HIGH)

Tokens are created in the EthixToken constructor but no events are emitted. All minting operations should transfer events from the 0x0 address according to the ERC20 standard. This behaviour is often required by explorers to correctly track minting events and analyse the token flow.

We recommend emitting the event Transfer(0x0, owner, INITIAL_SUPPLY) in line 21.

Fixed in the commit [6e86643428ffafd0fb712c51af0cec46b6f169a3]

Tokens added to revoked vesting are automatically released

Severity: **LOW** (Impact: MEDIUM, Likelihood: LOW)

The `IntervalTokenVesting` may be revoked at any moment by the contract `owner`.Revoking will transfer all of the unvested funds to the `owner` account. However, any additional tokens transferred to the already revoked contract will be automatically vested and could be claimed by the beneficiary. This behaviour is unexpected and will prevent refunding the tokens that are sent to a revoked contract by a mistake.

Consider freezing the balance after vesting has been revoked and adding a mechanism to reclaim any outstanding tokens that are deposited by mistake.

Fixed in the commit [182af42127dd404281bad52751a50fe908f453b2]

Unchecked math operation

Severity: **LOW** (Impact: MEDIUM, Likelihood: LOW)

There is a math operation in the calculation of the `vestedAmount` that isn't properly checked and can produce corrupted results in case of an overflow, This kind of errors are extremely hard to find, so it's always better to be safe and perform checked operations.

I recommend using the SafeMath library for all of the calculations and replacing low-level addition in `line 115` with the safe addition wrapper function `start.add(periodDuration.mul(i))` .

Fixed in the commit [8a3b4c77692b600998983fad9d50803f38afcfdd]

Unvalidated start of token vesting

Severity: **LOW** (Impact: MEDIUM, Likelihood: LOW)

The `IntervalTokenVesting` doesn't validate the input argument `_start`. Therefore, it's possible to start vesting in a date in the past when all of the funds may be immediately released by the `beneficiary`.

I recommend validating all of the input parameters and adding a check require(_start >= now) in line 50.

Fixed in the commit [779fc3c40f6c9ded548c1513b713abbf6c542f33]

Discount term is misleading

Severity: **LOW** (Impact: MEDIUM, Likelihood: LOW)

The `FixedPoolWithDiscountsTokenDistributionStrategy` uses the term "*discount*" to add bonus tokens in the `calculateTokenAmount` function. The official dictionary definition of a discount is "*reduction of the price*" which is how the potential buyers will expect it to be implemented. For example, if there is 20% discount and the original price was 100 units per item, the price after discount will be 80 and a person who holds 100 units will be able to buy 1.25 item (80/80 + 20/80). Unfortunately, with the current implementation, a person will be able to buy 1.2 item.

I recommend fixing the calculation formula and change it to:
`tokens.mul(100).div(HUNDRED.sub(discountIntervals[i].discount))`
to implement the expected behaviour. An alternative solution is to rephrase the name and comments to describe the calculations as counting a bonus.

Fixed in the commit [4787705b994ea1f686bc2f2fd57c68c3dd972801]

Unbounded discount configuration

Severity: **LOW** (Impact: MEDIUM, Likelihood: LOW)

The `FixedPoolWithDiscountsTokenDistributionStrategy` defines validation rules for discounts that are granted if a user purchase tokens in one of the time intervals. However, there is no checking for the upper bound of a discount, so any potential errors in discount configurations may have very severe consequences of distributing an unexpected amount of tokens.

Consider validating the upper bound for any discount by adding `require(discountIntervals[i].discount <= MAX_DISCOUNT)`. In case, the discount is implemented as suggested in the issue above, the `MAX_DISCOUNT` should be equal to 100.

Fixed in the commit [1464f3db6eed96f920f1ac9b1510ebec71b73a73]

Vesting policy should be configured before the crowdsale

Severity: **LOW** (Impact: MEDIUM, Likelihood: LOW)

The `VestedTokenDistributionStrategy` doesn't put any restriction when the configuration of the vesting scheme should be done. Therefore, the rules for the vesting may not be known for the investors at the moment of making a contribution. The configuration of vesting defined in the `configureVesting` function is allowed to be performed only once, which brings a risk of corrupting a crowdsale once it's in progress, if there is any error in the configuration. Moreover, by not configuring the vesting at all, the contract owner has a power to block the distribution of the tokens after collecting the funds.

I recommend configuring the vesting before the Ethix crowdsale has started by defining the rules in the `VestedTokenDistributionStrategy` constructor.

No clear policy about handling the unsold tokens

Severity: **LOW** (Impact: MEDIUM, Likelihood: LOW)

The `EthicHubTokenDistributionStrategy` contract holds all the tokens that are going to be distributed during the the crowdsale. However, there is no clear policy what will happen with any outstanding tokens that haven't been sold. Moreover, there is no function that will allow reclaiming unsold tokens when the crowdsale is finalized.

I recommend defining a clear policy about handling the unsold tokens such as burning or allocating for a future sale. This will require implementing a method that will allow reclaiming the unsold tokens after the crowdsale is over.

Fixed in the commit [894f83e85845cb34a7a30817534527e6bfd7763b]

NOTES AND SUGGESTIONS

- Consider adding additional parameters to the `Revoked` event such as `refundValue` or `tokenAddress` to facilitate tracking and filtering by client applications.
- We recommend using the `view` keyword instead of the legacy `constant` keyword to mark functions with the read-only access, such as `releasableAmount` and `vestedAmount`.
- Consider removing the artefacts from the revision control system found in lines 13 - 16: "<<<<<<< *HEAD …"* to clean up the code.
- The no-argument `PausableToken` constructor call in the `EthicToken` is redundant.
- There is a wrong title in the contract description. Consider replacing *FixedRateTokenDistributionStrategy* with *FixedPoolWithDiscountsTokenDistributionStrategy*.
- Declaration of the `SafeMath` library in all the contracts that derive from the `TokenDistributionStrategy` is redundant as it's already declared in the mentioned base contract. Similar redundancy occurs in the `CompositeCrowdsale` sub-contracts.
- The variable name `contributions` may be misleading, as it is usually used to describe the amount of ether contributed to the crowdsale contract, not the amount of tokens that are going to be granted.
- There is a typo in the word "*deppending*". It should be spelt "depending".
- The `VestedTokenDistributionStrategy` contract description has the same content as the `FixedPoolWithDiscountsTokenDistributionStrategy`. Consider adding a proper content describing how the vesting process works.
- In the description of the `vestedAmount` return value, it should say "*amount of tokens*" instead of just "*token*".

- Consider adding a validation to the `changeRegistrationStatuses` function to check if `targets` and `amounts` have the same length.
- There is a typo in the commentary, where it says "*whilelist*" it should say "*whitelist*".
- Consider removing this part of the `CompositeCrowdsale` description: "*CompositeCrowdsale is at the WIP stage and is meant to illustrate composition approach for managing crowdsale logic. It shouldn't be used in production code before necessary upgrades and testing.*".
- I recommend adding a sanity check require(_tokenDistribution != 0x0) to the `CompositeCrowdsale` constructor in the line 50.
- Consider updating the `CappedCompositeCrowdsale` description. Where it says "*Extension of Crowdsale*", it should say "*Extension of CompositeCrowdsale*". Similar issue occurs in the `RefundableCompositeCrowdsale` contract.
- Consider moving the preconditions for a valid purchase defined in lines 50-51 of `EthicHubPresale` to the `validPurchase` function, in order to maintain consistency across all of the crowdsale contracts.
- Consider restricting access to the `compensate` function only to parties involved (crowdsale owner and contributor) to keep the full control and protect against unforeseen problems like temporary problems with the access to the token wallet.

Fixed in the commits add162e699cda89e63fd92519c7e0a13a17e8cfc and 2ab2d3f14e566b6504ccbfc3b22b60dfc396a52a

Jakub Wojciechowski