SOLIDIFIED

Sample Audit Report for [Company X]. March 13, 2018.

Summary

Audit Report prepared by Solidified for Company X covering the token and crowdsale contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the below token sale. The debrief took place on March 13, 2018 and the final results are presented here.

Audited Files

The following files were covered during the audit:

- XToken.sol
- MainCrowdSale.sol
- Whitelist.sol
- CappedCrowdsale.sol
- FinalizableCrowdsale.sol

Notes

The audit was conducted on commit `562c763345e2e2d795f9ee54da526cf3e2992ff2`
The audit was based on the solidity compiler `0.4.20+commit.3155dd80`

Intended Behavior

The purpose of these contracts is to create the XToken and distribute it to the public, in a crowdsale process. Intended behavior spec example can be found here.

Issues Found

1. Validation of external call return values

MainCrowdSale.sol Line 104

If there's not enough gas for FundDistibutor's fallback function to run (which could happen deliberately or accidentally-- network congestion has a lot of people playing with gas price/limit in metamask without realizing what they're doing for example), it will fail silently. In other words,

an out of gas exception in FundDistributor's fallback will not propagate up the call stack. This will cause `wallet.call.value(x)()` to just fail and return false, not throw.
Specifically, if `wallet.call.value(amount)()` can fail silently, thereby: in `close()`, closing the refund vault without actually sending along the funds, i.e. locking all the funds in the vault in `forwardFundsToWallet()`, thus effectively locking the funds sent for that purchase.

More info can be found here: https://consensys.github.io/smart-contract-best-practices/recommendations/#handle-errors-in-external-calls

Recommendation

```
require(wallet.call.value(amount)())
```

2. Possibility of making the auction unfinalizable

If the first buy transaction lands into the window between '`cap`' and '`cap - minimumInvesment`' the contract enters a locked state in which vault can't be closed and no further buy transaction can be processed. `vault.close()` in finalization function doesn't get called, because `goalReached` returns 'false' and `vault.close()` in `forwardFundsToWallet` doesn't get called, because the vault is empty.

3. Reaching the soft cap will stop the sale prematurely

If the sale reaches the goal before the `initialEndTime`, no further purchase is possible until the extra sale period starts. This happens because the function `hasEnded` will return true when both the `goalReach` and `initialEndTimeNotReached` returns true, making any further transaction fail.

Recommendation

Make a function to check whether the `initialEndTime` was reached, return false if not.

4. Token burning mechanism is not present

As per the spec document shared, unsold tokens should be burned as soon as the token sale is finished. The current implementation has no such method to burn the remaining tokens during the finalization process and therefore, there's no guarantee that such action will be taken.

Recommendation

Consider implementing this in the finalization function rather than burning it manually.

5. There is no reliable way to stop the main sale

The intended behaviour states that there must be a way to stop the main sale if a bug is found, but such mechanism is not present in the contracts. The only way to achieve that is for the owner to burn all of his tokens, which may not be possible (if owner is another contract, for example).

Recommendation

Implement a direct trigger that blocks purchases when activated.

6. Owner can still transfer tokens during the sale

During the sale, or before, the owner is still allowed to transfer tokens as he wishes. That might be contrary to the statement in the intended behaviour that says *"No tokens should be handed out “for free” to anyone"*. Also, the sale contract relies on the owner’s balance for some calculation and therefore, transferring tokens outside can alter some sale parameters.

Recommendation

Consider reviewing the power of owner transferring tokens at any time.

7. Consider changing the refund calculation

For different combination of rate, hardcap and other parameters, value of `amountToBeRefunded` might get overwritten due to multiple assignments. This will not occur with the current values given in the document, but may occur with other values.

Recommendation

Consider rethinking the calculation method for refund to resolve this issue.

8. Company X keeps track of non-holders

After transferring all the tokens, a user might still be present in the holders array and in `isHolder` mapping in the main contract. Therefore, those arrays wouldn't really reflect the true holders.

Recommendation

Review the need to keep this list on the contract, as its rather expensive and don't necessarily reflects the reality. If this information is needed, consider implementing an off-chain solution.

9. Consider removing duplicate validations

Some validations such as checking for the cap during the token buying process are performed multiple times. It is not a bad practice to have multiple checks, but avoiding it can save a bit of gas amount during the transaction.

10. Consider removing empty functions

In the contracts there are a few empty functions, which serve no purpose at all, so consider removing them. An example is the whitelisted constructor and the finalization function, but there are others.

11. Consider following the solidity style guide

Consider following the Solidity guidelines on formatting the code and commenting. It can improve the overall code quality and readability.

12. Update Compiler Version

The compiler version is outdated. It's recommended to use latest compiler version (0.4.20) as of this writing.

Closing Summary

Several major and minor issues were found during the audit which can severely break the intended behaviour. It is strongly advised that these issues are correct before proceeding with the crowdsale. It is furthermore recommended to post the contracts on Solidified or other public bounty afterwards.

XToken.sol has been verified as fully ERC20 compliant, and has taken recommended measures to mitigate the known EIP20 API Approve / TransferFrom multiple withdrawal attack. Beyond the issues mentioned, the contracts were also checked for overflow/underflow issues, DoS, and re-entrancy vulnerabilities. None were discovered.

OpenZeppelin contracts such as Ownable/SafeMath/etc. have been widely audited and secured and as such, were not prioritized for auditing.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of the Company X platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.