Simple Token Sale Audit

OPENZEPPELIN SECURITY | NOVEMBER 8, 2017

Security Audits

The Simple Token team asked us to review and audit their Simple Token Sale contracts. We looked at the code and now publish our results.

The audited code is located in theOpenSTFoundation/SimpleTokenSalerepository. The version used for this report is commit `66025fede245a382ef6ed9f32dd0ecd1da34301f`. The `ProcessableAllocations` contract is an internal tool and outside of the scope of this audit.

Here's our assessment and recommendations, in order of importance.

Update: *The Simple Token team has followed most of our recommendations and updated the contracts. The new version is at commit* `ac1b56cb9bb3b8f822a55fe6cf7d0c6d79651e29`.

Critical Severity

No issues of critical severity.

High Severity

No issues of high severity.

Medium Severity

Sale can end while being paused

[illegible] is done only after calling the [illegible] function and the state during the pause period may be misreported.

A consequence of this could be losing the ability to update whitelist in `updateWhitelist` function during the pause period.

We recommend updating `hasSaleEnded` function to consider the pause period.

Update: *Fixed in these two commits.*

Unchecked assumption that trustee and token have the same owner

In the `reclaimTokens` function, there is a token transfer to the `trustee` contract owner, but according to the comment above it should be transferred to the `tokenContract` owner: *Note that the trustee should be able to move tokens even before the token is finalised because SimpleToken allows sending back to owner specifically.* This statement is only true if the token contract owner is the same as trustee owner, but this precondition is not enforced anywhere.

We recommend replacing `owner` with `tokenContract.owner` in line 206 to specify the transfer target precisely.

Update: *Fixed in this commit improving both the TokenSale and Trusteecontracts.*

Two unchecked math operations involving constant values

There are some math operations that aren't checked like calculating the number of bought tokens orthe actual cost for the partial amount of tokens. It's always better to be safe and perform checked operations.

It's worth mentioning that the safety of all the other calculations were correctly ensured with the `SafeMath` library.

Consider using the `SafeMath` library, or performing pre-condition checks on **all** of the math operations.

Update: *Fixed in this commit by introducing the _ `PURCHASE_DIVIDER` constant.*

In the `TokenSale` contract, there is validation for most of the configuration constants which is a very good practice. The validation skips checking the correctness of `TOKENS_SALE` parameter which can allow creation of a contract that is unable to sell any tokens at all.

We recommend adding the missing validation to the `TokenSale` constructor.

Update: *The Simple Token team indicated that, `TOKENS_SALE` is effectively validated by (1) the constructor confirming that all of the token values add up to `TOKENS_MAX` and (2) `TokenSale.initialize` confirming that the balance of `TokenSale` is equal to `TOKENS_SALE`. It is true that if `TOKEN_SALE` were set to `0` in `TokenSaleConfig` (and other values adjusted) and if we either transfer `0 ST` to `TokenSale` or do not transfer anything at all, those validations would pass. However, the team thinks they must surely be allowed to assume that their config contract is correct.*

Access modifier conflicting with documentation and function body

The `onlyOps` modifier restricts access to the `processAllocation` function, allowing only an account that matches the `opsAddress`. Such a behaviour conflicts with the documentation that describes the function as *Push model which allows **_the owner*** to transfer tokens to the beneficiary_. Moreover, this is also inconsistent with the function body which contains logic that requires a message sender to be the owner or the admin.

We recommend resolving the conflict by either removing the `onlyOps` restriction access modifier or updating the documentation and removing irrelevant code fragments lines 167–170) from the function body.

Update: *Fixed in this commit by updating comments and removing the unreachable code.*

Possible to initiate Ownership Transfer with the null address

Function `initiateOwnershipTransfer` allows initiating a transfer passing the `0x0` address as a `_proposedOwner.` This action will be ineffective as it's impossible to call the `completeOwnershipTransfer` function from the `0x0` address. The null address is also used as a special value to mark that the transfer was completed so it'd be the best if it could be set only by internal invocation.

Update: *Fixed in this pull request. The team decided to postpone merging the changes to the master branch as it will require excessive refactor of all the tests.*

Duplicated logic in mock contracts

Contracts `TokenSaleMock` and `FutureTokenSaleLockBoxMock` have almost identical content duplicating the code that implements time manipulation behaviour. Not only increases it the maintenance costs but also brings the risk of updating only one of the contracts and skipping another.

We recommend extracting the common logic into an abstract `Mock` contract that will be a base class for both `TokenSaleMock` and `FutureTokenSaleLockBoxMock`.

Update: *While the Simple Token team agree with the points, they do not feel that this is a concern that merits changing the code.*

Imprecise unlock date documentation

The definition of `unlockDate` from `FutureTokenSaleLockBox` contract (The unlock date is initially six months after `26 weeks`. The difference is dependent on an exact `unlockDate` and requires complex calendar logic to be precisely calculated.

We recommend documenting the periods in fixed time units such as days or weeks.

Update: *Fixed in this commit.*

Notes & Additional Information

- Congratulations on writing such a thorough test suite! There are 244 tests in total, with most of them covering the integration of all contracts into the token sale contract.
- There is a typo in `Trustee`, it saysacount where it should say *account*. (**Update**: *fixed in this commit*).
- The comparison `ok == true` is redundant as the `ok` variable is already a boolean value. The same comment applies to checking the function result by `require(hasUnlockDatePassed() == true).` (**Update**: *fixed in this commit*).

facilitate calculating the previously mentioned values (**Update**: the team decided to leave the *code as it is, because the constants are unrelated in general and the similar calculation occurs in the current version only*).

- `initialize` function is described as: *…link the sale token with the token contract _while the linking process is executed not in this function but in the constructor. Consider updating the comment to explain that the function is only checking correctness of configuration parameters. (**Update**: _fixed in this commit*).
- Consider turning magic numbers such as “18” and “3” that are repeatably used in `buyTokens` functions to predefined constants. (**Update**: *fixed in this commit*).
- Consider using full `ERC20Interface` instead of the `TokenInterface` stub to get the compilation time syntax checks. (**Update**: *fixed in this commit*).
- Keep in mind that there is a possible attack vector on the `approve` / `transferFrom` functionality of ERC20 tokens, described here. Consider implementing one of the proposed mitigations, or using the ERC20 implementation from OpenZeppelin which already has one in place. (**Update**: explanation note added *in this pull request*).
- In `finalize` function of the `TokenSale` contract there is a suggestion that: *The owner will also need to finalize the token contract so that token transfers are enabled*. It is actually the `admin` that needs to `finalize` the token, not the `owner`. We recommend to update the documentation or change the permission scheme, so the description is compatible with the documentation. (**Update**: *fixed in this commit*).
- The contracts `SafeMath` and `Pausable` are very similar to OpenZeppelin library contracts sharing the same names. Consider avoiding code repetition, which can bring regression problems and introduce unexpected bugs. We recommend using the standard modules from OpenZeppelin. (**Update**: *the team decided to wait with directly linking to the OpenZeppelin library until it offers full support for the new version of solidity compiler*).
- Consider removing the call to the `finalizeInternal` function from the publicly accessible `buyTokens` method. It’s safer to keep the critical functionality in the hands of admin/owner than to leave it open to the public. (**Update**: *the team decided to leave the current version as they have enough control by using the `SimpleToken.finalize` function*).

Conclusion

follow best practices and reduce the potential attack surface.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Simple Token Sale contracts. We have not reviewed the related Simple Token project. The above should not be construed as investment advice. For general information about smart contract security, check out our thoughts here.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits

OpenZeppelin