Contract Audit: WSTA Contract

Preamble

This audit report was undertaken by @adamdossa for the purpose of providing feedback to the community. It has been written without any express or implied warranty.

This audit was done on the code deployed on the Ethereum mainnet at: https://etherscan.io/address/0xedeec5691f23e4914cf0183a4196bbeb30d027a0#code

Disclosure

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. There is no owed duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

Audit Scope

The scope of this audit is:

1. Ensure that the WSTA contract is a standard ERC20 contract, constructed primarily from the OpenZeppelin 0.5.0 token contracts.
2. Ensure that there are no minter rights to any external address or contract - i.e. that the token cannot be unilaterally minted (without wrapping Statera tokens).
3. Ensure that the `wrap` function operates as expected - accepts the underlying Statera token, and mints an appropriate amount (i.e. value minus fee) of WSTA.
4. Ensure that the `unwrap` function operates as expected - burns WSTA and supplies an appropriate amount of Statera (STA) token (i.e. value minus fee).

Contract Behaviour

The WSTA contract at https://etherscan.io/address/0xedeec5691f23e4914cf0183a4196bbeb30d027a0#code wraps the underlying Statera (STA) token.

The underlying STA token is a deflationary token that burns 1% of the amount transferred (rounded up in the case this is fractional), each time a transfer takes place.

In other words, if a user transfers 100 units of STA (i.e. 100 / 10^18 tokens) then 1 unit (i.e. 1 / 10^18) will be burnt. Similarily, if a user transfers 50 units of STA, then 1 unit will be burnt (since 0.5 units is rounded up to 1).

The WSTA contract allows a user to wrap STA tokens, and emits a WSTA token to represent the amount wrapped. The WSTA tokens can be freely transferred (without the STA fee) and unwrapped at a later time. Since the `wrap` and `unwrap` functions involve transferring units of the underlying STA token, these will attract the 1% fee above, so the user will end up with 1% fewer wrapped / unwrapped tokens than the amounts being transferred.

Audit

OpenZeppelin Usage

The WSTA contract uses the OpenZeppelin contracts from version 2.5.1:
https://github.com/OpenZeppelin/openzeppelin-contracts/releases/tag/v2.5.1

Other than the WSTA contract itself (which has the `wrap` and `unwrap` functions) the contracts imported by the WSTA contract were all identical to their versions in the above OpenZeppelin release.

Minting Rights

The WSTA contract inherits from Context, ERC20Detailed, ERC20Mintable, ERC20Burnable. However, during initialisation the `_removeMinter` function is called, removing the default minter (`msg.sender`).

This can also be seen by examining the storage of the contract, and ensuring that the contract creation address (0x20da02990156BBE00be10C813da0d04d094fC8Ca) returns false from the `isMinter` function. Whilst it is possible for an existing minter to add a new minter, in this case, since the minting rights of the token deployer address were removed on initialisation, no new minters can be added.

In summary, no external address or contract (i.e. other than the WSTA contract itself) has the right to mint additional units of WSTA.

Wrap and Unwrap Functions

The `wrap` and `unwrap` functions are implemented in the WSTA contract.

The `wrap` function allows a user to transfer the underlying STA tokens to the WSTA contract, and receive an appropriate amount of WSTA.

The amount of WSTA to mint is determined by checking the contracts STA balance, before and after the transfer of STA, and taking the difference. This allows the WSTA contract to account for the burnt fee of the underlying STA contract (which in practice is 1% as above).

For example, suppose that a user approves the WSTA contract for 100 STA tokens (an amount of `100*10^18`). The user then calls `wrap(100*10^18)`. The user would receive `99*10^18` WSTA tokens, since 1% of the amount (i.e. `1*10^18` STA tokens) would have been burnt when transferring the `100*10^18` tokens from the user to the WSTA contract.

Once wrapped, WSTA tokens can be freely transferred, without attracting the 1% fee of the underlying STA tokens.

The `unwrap` function allows a user to burn their WSTA tokens, and receive STA tokens of the same amount minus the 1% fee involved in transferring them back to the user.

The contract will transfer an amount of STA equal to the number of burnt WSTA tokens to the user, but the STA contract will burn 1% of this amount, meaning the user will receive 99% of the burnt WSTA tokens as a STA balance. The WSTA contract relies on the STA balance to correctly calculate the fee.

For example, suppose that a user calls `unwrap(100*10^18)` - assuming that they have at least `100*10^18` WSTA tokens, they will receive `99*10^18` STA tokens, and their WSTA balance will be reduced by `100*10^18`.

In summary, the `wrap` and `unwrap` functions work as intended, accounting for the underlying transfer fees of the STA contract correctly.

Notes

As per the usual ERC20 semantics, a user must first approve the WSTA contract for the STA token, before calling the `wrap` function. The amount of tokens wrapped (i.e. the value of the `_amount` parameter when calling `wrap`) must be less than or equal to the approved amount.

If a user were to directly transfer tokens to the WSTA contract, without using the `wrap` function, those tokens would be lost, since the WSTA contract would have no way to track such a deposit.

The WSTA contract relies on the correct functioning of the STA contract. Whilst this audit did review the STA contract to ensure that the transfer fees are correctly accounted for in the WSTA contract, this was not a full and complete audit of the STA contract.

Summary

The WSTA contract fulfils the audit scope, providing a simple wrapper functionality on top of the STA contract. It relies on the underlying STA contract to function correctly.