

Security Audit

of TRADECLOUD TOKEN Smart Contracts

May 13, 2019










Produced for



by



Table Of Contents

Foreword	1
Executive Summary	1
Audit Overview	2
1. Scope of the Audit	2
2. Depth of Audit	2
3. Terminology	2
Limitations	4
System Overview	5
Best Practices in TRADECLOUD's project	6
1. Hard Requirements	6
2. Soft Requirements	6
Security Issues	7
Trust Issues	8
1. Owner is also pauser and whitelist admin  	8
Design Issues	9
1. Unused import  	9
2. Redundant modifier  	9
3. Locked tokens possible  	9
4. No checking of error messages in tests  	9
5. Fix OpenZeppelin dependency  	10
Recommendations / Suggestions	11
Disclaimer	12

Foreword

We first and foremost thank TRADECLOUD for giving us the opportunity to audit their smart contracts. This documents outlines our methodology, limitations, and results.

– ChainSecurity

Executive Summary

The TRADECLOUD smart contracts have been analyzed under different aspects, with a variety of tools for automated security analysis of Ethereum smart contracts.

Overall, TRADECLOUD is using pure OpenZeppelin contracts for their token implementation. The OpenZeppelin implementation is a well written, common and widely used library for solidity smart contracts. It's battle tested and considered safe. As TRADECLOUD just uses OpenZeppelin code CHAINSECURITY did not find major issues but still identified some minor ones. After the intermediate report, all raised issues were addressed and fixed. CHAINSECURITY has no remaining security concerns about the TRADECLOUD smart contract.

Audit Overview

Scope of the Audit

The scope of the audit is limited to the following source code files. They have been reviewed based on SOLC compiler, version 0.5.6 and EVM version PETERSBURG. All of these source code files were received on April 11, 2019 as part of the git commit 4e6b112ed3f8dbac6595813e64471d83fa05a08b. The latest update has been received on April 26, 2019 with the commit 552a3e699f583e8cb2c542b7705573b9a1bb852d.

File	SHA-256 checksum
TradecloudToken.sol	b685f040c91e6cac908690ceda505b8204692cb95b9c2d5d64f121969a6e934d

Depth of Audit

The scope of the security audit conducted by CHAINSECURITY was restricted to:

- Scan the contracts listed above for generic security issues using automated systems and manually inspect the results.
- Manual audit of the contracts listed above for security issues.

Terminology





For the purpose of this audit, we adopt the following terminology. For security vulnerabilities, we specify the *likelihood*, *impact* and *severity* (inspired by the OWASP risk rating methodology¹).

Likelihood represents the likelihood of a security vulnerability to be encountered or exploited in the wild.










Impact specifies the technical and business related consequences of an exploit.

Severity is derived based on the likelihood and the impact calculated previously.

We categorize the findings into 4 distinct categories, depending on their severities:

-  Low: can be considered as less important
-  Medium: should be fixed
-  High: we strongly suggest to fix it before release
-  Critical: needs to be fixed before release

These severities are derived from the likelihood and the impact using the following table, following a standard approach in risk assessment.

LIKELIHOOD	IMPACT		
	High	Medium	Low
High			
Medium			
Low			

¹https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

During the audit concerns might arise or tools might flag certain security issues. After careful inspection of the potential security impact, we assign the following labels:

- **✓ No Issue**: no security impact
- **✓ Fixed**: during the course of the audit process, the issue has been addressed technically
- **✓ Addressed**: issue addressed otherwise by improving documentation or further specification
- **✓ Acknowledged**: issue is meant to be fixed in the future without immediate changes to the code

Findings that are labelled as either **✓ Fixed** or **✓ Addressed** are resolved and therefore pose no security threat. Their severity is still listed, but just to give the reader a quick overview what kind of issues were found during the audit.

Limitations

Security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a secure smart contract. However, auditing allows to discover vulnerabilities that were overlooked during development and areas where additional security measures are necessary.

In most cases, applications are either fully protected against a certain type of attack, or they lack protection against it completely. Some of the issues may affect the entire smart contract application, while some lack protection only in certain areas. We therefore carry out a source code review trying to determine all locations that need to be fixed. Within the customer-determined timeframe, CHAINSECURITY has performed auditing in order to discover as many vulnerabilities as possible.

System Overview

Token Name & Symbol	TRADECLOUD TOKEN, TCST
Decimals	0
Tokens issued	50'000'000 TCST
Token Type	ERC 20
Token Generation	Pre-Minted

Table 1: Facts about the TCST token and the Token Sale.

The TCST token is a plain ERC20 token which has a `Pausable` contract and `WhitelistedRole` as extra features. All contracts are from the OpenZeppelin library.

Best Practices in TRADECLOUD's project

Projects of good quality follow best practices. In doing so, they make audits more meaningful, by allowing efforts to be focused on subtle and project-specific issues rather than the fulfillment of general guidelines.

Avoiding code duplication is a good example of a good engineering practice which increases the potential of any security audit.

We now list a few points that should be enforced in any good project that aims to be deployed on the Ethereum blockchain. The corresponding box is ticked when TRADECLOUD's project fitted the criterion when the audit started.

Hard Requirements

These requirements ensure that the TRADECLOUD's project can be audited by CHAINSECURITY.

- ☒ The code is provided as a Git repository to allow the review of future code changes.
- ☒ Code duplication is minimal, or justified and documented.
- ☒ Libraries are properly referred to as package dependencies, including the specific version(s) that are compatible with TRADECLOUD's project. No library file is mixed with TRADECLOUD's own files.
- ☒ The code compiles with the latest Solidity compiler version. If TRADECLOUD uses an older version, the reasons are documented.
- ☒ There are no compiler warnings, or warnings are documented.

Soft Requirements

Although these requirements are not as important as the previous ones, they still help to make the audit more valuable to TRADECLOUD.

- ☒ There are migration scripts.
- ☒ There are tests.
- ☒ The tests are related to the migration scripts and a clear separation is made between the two.
- ☒ The tests are easy to run for CHAINSECURITY, using the documentation provided by TRADECLOUD.
- ☒ The test coverage is available or can be obtained easily.
- ☒ The output of the build process (including possible flattened files) is not committed to the Git repository.
- ☒ The project only contains audit-related files, or, if not possible, a meaningful separation is made between modules that have to be audited and modules that CHAINSECURITY should assume correct and out of scope.
- ☒ There is no dead code.
- ☒ The code is well documented.
- ☒ The high-level specification is thorough and allow a quick understanding of the project without looking at the code.
- ☒ Both the code documentation and the high-level specification are up to date with respect to the code version CHAINSECURITY audits.
- ☒ There are no getter functions for public variables, or the reason why these getters are in the code is given.
- ☒ Function are grouped together according either to the Solidity guidelines², or to their functionality.

²<https://solidity.readthedocs.io/en/latest/style-guide.html#order-of-functions>

Security Issues

This section relates our investigation into security issues. It is meant to highlight whenever we found specific issues but also mention what vulnerability classes do not appear, if relevant.

CHAINSECURITY has no concern to raise in this category of the report.

Trust Issues

This section mentions functionality which is not fixed inside the smart contract and hence requires additional trust into TRADECLOUD, including in TRADECLOUD's ability to deal with such powers appropriately.

Owner is also pauser and whitelist admin

The different roles in the TRADECLOUD smart contracts are documented as follows:

```
Owner: can managed lifecycle of smart contract like deploy
TokengateAdmin: can whitelist and pause token
TradeCloudAdmin: own all token and can authorize tokengateAdmin to transfer
                token to final investor
Investor(s) are persons participating in STO
```

CHAINSECURITY wants to point out that the account deploying the contracts (called the Owner) automatically assumes the roles of pauser and whitelistAdmin. As he does not renounce these roles after having assigned TokengateAdmin and TradeCloudAdmin their respective roles inside the constructor, the owner account keeps these powers which is not clearly documented.

Fixed: Inside the constructor, the deploying account now renounces the roles of pauser and whitelistAdmin after having assigned these roles to the designated addresses.

Design Issues

This section lists general recommendations about the design and style of TRADECLOUD's project. They highlight possible ways for TRADECLOUD to further improve the code.

Unused import

TRADECLOUD imports `import "openzeppelin-solidity/contracts/token/ERC20/ERC20Burnable.sol"` but does not use the contract anywhere. In case, TRADECLOUD imported ERC20Burnable with the intention to enable the burning of tokens, note, that this is not possible in the current implementation as the TradecloudToken does not inherit from the imported burnable contract.

Fixed: TRADECLOUD confirmed the TCST token should not be burnable and removed the unused import.

Redundant modifier

The functions `transfer` in `transferFrom` in the TradeCloudToken contract are using the `whenNotPaused` modifier and call their corresponding parent (`super.`) functions. These are `transfer` and `transferFrom` in the ERC20Pausable contract. The only purpose of these functions is, the additional `whenNotPaused` modifier compared to their corresponding parents (`super.`) functions. Hence, adding `whenNotPaused` is redundant and not needed in this case.

Fixed: TRADECLOUD removed the redundant modifier.

Locked tokens possible

If tokens especially TCSTs are accidentally or intentionally sent to the TradecloudToken contract, the tokens will be locked, with no way to recover them. Incidents in the past showed this is a real issue as there always will be user sending tokens to the token contract due to unknown reasons. See here for some examples³.

Note:

ETH and tokens can be forced into any contract and be locked there if no "recover" function exists. If contracts are not supposed to handle/process Ether or token transfers, CHAINSECURITY does not further report this kind of locked tokens. But CHAINSECURITY reports locked token or locked ETH for contracts which are supposed to handle/process tokens or ETH in some way.

Fixed: TRADECLOUD added a `validDestination(address)` modifier to prevent accidental TCST tokens transfer to the zero address or to the token contract. `transferFrom()` also checks that the from address fulfills these criteria which CHAINSECURITY thinks is unnecessary.

No checking of error messages in tests

Inside the truffle tests TRADECLOUD currently does not check the message of thrown errors. A test which is assumed to throw a certain error might actually throw a different error, leading to the assumption that the code is working as expected while it actually isn't.

In TRADECLOUD's test suite, there are tests ensuring that certain functions revert when the token is paused. All of these test cases pass successfully as the calls throw, however most of them throw for another reason and not because the token is paused.

The following example is expected to throw because the token is paused which inhibits `transferFrom()`:

³<https://coincentral.com/erc223-proposed-erc20-upgrade/>

```
it('should fail to transfer', async function () {
  // Act + Assert
  await token.pause();
  await utils.expectThrow(token.transferFrom(alice, bob, 1, {from: owner}));
});
```

This throws as expected, however the reason is not that the token is paused but because the from address is not whitelisted. When testing the pausable functionality, all function calls should succeed when the token is not paused and only throw if the token is paused.

To be sure a specific error is thrown, TRADECLOUD should consider checking the error message inside the truffle tests.

Fixed: TRADECLOUD fixed and improved the tests, the tests now fail because of the expected reasons. Whenever possible TRADECLOUD ensures that the expected error is thrown. Note that OpenZeppelin Pausable and WhitelistedRole do not throw an error message in their **require** statement, thus it's not possible to assure easily that this was the **require** that actually threw.

Fix OpenZeppelin dependency

Currently the OpenZeppelin version used is not fixed and set to **"openzeppelin-solidity": "^2.2.0"** inside **package.json**. This ensures version 2.2.0 or more recent is used. However projects should be deployed with the version they have been thoroughly tested and been audited with in order to prevent unexpected behaviour. To prevent accidental deployment with a more recent version, CHAINSECURITY recommends to lock the OpenZeppelin version.

Fixed: TRADECLOUD fixed the OpenZeppelin dependency to 2.2.0.

Recommendations / Suggestions

✓ MAXIMUM_SUPPLY is calculated with $1 * (10 ** 6) * (10 ** \text{uint256}(\text{decimals}))$. Multiplying with 1 does not change anything.

✓ Update README.md

- Outdated reference to `initShell.sh` which doesn't exist anymore in the current commit
- instructions use different package manager, `npm` and `yarn`. Use one consistently.
- The API documentation is imprecise. Under Investor can transfer only `token.transfer()` is listed. There is nothing preventing users from transferring tokens using `transferFrom()` if all involved accounts are white listed. `transferFrom()` is only mentioned for `TokenGateAdmin` and `TradeCloudAdmin`, where `transfer()` is not mentioned. Both of these can also transfer funds using `transfer()`, especially note that whoever controls the `TradeCloudAdmin` where the total supply gets minted to initially has full control over these funds. Furthermore the following may be misunderstood:

```
TradeCloudAdmin:
Owns all tokens and can let tokengate transfer token to beneficiary
account using token.transferFrom(address from, address to, uint
tokens)
```

To formulate this more clearly, `TradeCloudAdmin` can use `approve()` to let `TokenGateAdmin` transfer tokens using `transferFrom()`.

Disclaimer

UPON REQUEST BY TRADECLOUD, CHAINSECURITY LTD. AGREES MAKING THIS AUDIT REPORT PUBLIC. THE CONTENT OF THIS AUDIT REPORT IS PROVIDED "AS IS", WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND, AND CHAINSECURITY LTD. DISCLAIMS ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT. COPYRIGHT OF THIS REPORT REMAINS WITH CHAINSECURITY LTD..