verichains

SECURITY AUDIT OF

STABLECOIN

Public Report

Sep 21, 2023

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
TomoChain Blockchain	TomoChain is a highly scalable and secure blockchain network in mainnet since 2018, taking a community-driven approach to accelerate the mass adoption of web3 applications.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Sep 21, 2023. We would like to thank Stably for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the StableCoin. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerabilities in the contract code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY 5

1.1. About StableCoin 5

1.2. Audit scope 5

1.3. Audit methodology 5

1.4. Disclaimer 6

2. AUDIT RESULT 7

2.1. Overview 7

2.2. Contract codes 7

2.2.1. TRC25Stablecoin contract 7

2.2.2. TRC25Compliance contract 8

2.2.3. ERC20StablecoinUpgradeable contract 8

2.3. Findings 9

2.3.1. Can bypass the fee charge by using a contract to transfer tokens - MEDIUM 9

2.3.2. Calculates incorrect allowance if a contract transfers tokens - MEDIUM 11

2.3.3. Redundant code for charging fee - INFORMATIVE 12

3. VERSION HISTORY 14

1. MANAGEMENT SUMMARY

1.1. About StableCoin

The StableCoin includes a Stably USD and a TRC25 stablecoin.

The Stably USD built upon the ERC20 token standard is a multichain stablecoin fully backed with liquid USD collateral assets such as cash, cash equivalents, and/or stablecoins. The collateral assets are held by Stably and/or other specified financial partners. Every Stably USD token may be minted/redeemed 1-to-1 with USD, USDC or other specified stablecoins through Stably Ramp.

The TRC25 stablecoin built upon the TomoChain TRC25 token standard is fully backed by and redeemable 1-to-1 for USD. With the TRC25 token, users and members of the TomoChain community can easily on-ramp from fiat to Web3 and DeFi on the TomoChain network.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the StableCoin.It was conducted on commit `42bab0b74aea65fd3214e7643e22089ae219a5b6` from git repository *https://github.com/stablyio/evm-token-contracts/*.

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function

- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

2.2. Contract codes

The StableCoin was written in `Solidity` language, with the required version being `^0.8.9`.

2.2.1. TRC25Stablecoin contract

A `TRC25Stablecoin` contract extends `TRC25`, `TRC25Permit`, and `TRC25Compliance`, `AccessControl` and `Pausable` contracts.

A StableCoin is a token that implements the `TRC25` standard on the TomoChain network, provides basic functionality to transfer tokens, allows tokens to be approved so they can be spent by another on-chain third party, and manages fees to prevent abuse of the feature.

The TRC25Stablecoin contract is a smart contract that manages the lifecycle of the StableCoin. It is responsible for creating a new StableCoin, destroying tokens, and managing the StableCoin's balance on the TomoChain. It also provides public functions that other contracts or end users can call to transfer StableCoins from one account to another.

`TRC25Permit` is an extension of the TRC25 standard that adds the ability to approve token transfers via a signature rather than a transaction. This is useful for cases where the user does not have TOMO to pay for gas, or when the user is signing a contract that requires a token transfer.

`TRC25Compliance` is an extension of the TRC25 standard that adds the ability to check if a token transfer is compliant with the token's compliance rules. This is useful for cases where the token has compliance rules that must be followed, such as freeze requirements.

`AccessControl` is a contract that provides access control mechanisms, including role-based access control, and the ability to check if a user has a certain role.

`Pausable` is a contract that provides a way to pause and unpause certain functions in a contract. This is useful for cases where a contract needs to be paused in order to prevent certain actions from being taken, such as when a contract is being upgraded.

The TRC25Stablecoin has 5 roles: default admin, compliance, fee, minter and pauser.

- Default admin user: A highest role in the contract, has the right to grant other roles to other users.
- Compliance user: A role that has the permission to freeze/unfreeze an account and burn (called seize) amount of token from any account.
- Fee user: A role that has the permission to set the fee rate for transfers.

- Minter user: A role that has the permission to mint new token.
- Pauser user: A role that has the permission to pause/unpause the contract.

The TRC25Stablecoin has 2 states: normal and paused. The contract is in a normal state by default. When the contract is paused, all functions that are not related to pausing/unpausing the contract are disabled.

This contract has charge fee when the user transfers token. The fee rate is set by the fee user. The fee is a fixed price that is sent to an owner of this contract.

2.2.2. TRC25Compliance contract

A `TRC25Compliance` contract extends `TRC25` contract.

`TRC25Compliance` is an extension of the TRC25 standard that adds the ability to check if an address is frozen or not. Additionally, the contract has the function to burn (called seize) amount of token from any account, freeze and unfreeze an account.

2.2.3. ERC20StablecoinUpgradeable contract

A `ERC20StablecoinUpgradeable` contract extends `ERC20Upgradeable`, `ERC20PermitUpgradeable`, `ERC20BurnableUpgradeable`, `PausableUpgradeable`, `AccessControlUpgradeable`, `ERC20FlashMintUpgradeable`, `ERC20ComplianceUpgradeable` and `Initializable` contracts.

A StableCoin is a token that implement the `ERC20` standard on the Ethereum network, provides basic functionality to transfer tokens.

The ERC20StablecoinUpgradeable contract is a smart contract that manages the lifecycle of a StableCoin. It is responsible for creating a new StableCoin, destroying tokens, and managing the StableCoin's balance on Ethereum. It also provides public functions that other contracts or end users can call to transfer StableCoins from one account to another.

`ERC20PermitUpgradeable` is an extension of the ERC20 standard that adds the ability to approve token transfers via a signature rather than a transaction. This is useful for cases where the user does not have ETH to pay for gas, or when the user is signing a contract that requires a token transfer.

`ERC20BurnableUpgradeable` is an extension of the ERC20 standard that adds the ability to burn tokens. This is useful for cases where the token needs to be burned, such as when a token is being upgraded.

`PausableUpgradeable` is a contract that provides a way to pause and unpause certain functions in a contract. This is useful for cases where a contract needs to be paused in order to prevent certain actions from being taken, such as when a contract is being upgraded.

`AccessControlUpgradeable` is a contract that provides access control mechanisms, including role-based access control, and the ability to check if a user has a certain role.

`ERC20FlashMintUpgradeable` is a contract that provides the ability to flash mint tokens. This is useful for cases where a contract needs to mint tokens in order to perform an action, such as when a contract is being upgraded.

`ERC20ComplianceUpgradeable` is an extension of the ERC20 standard that adds the ability to check if a token transfer is compliant with the token's compliance rules. This is useful for cases where the token has compliance rules that must be followed, such as freeze requirements.

The ERC20StablecoinUpgradeable has 4 roles: default admin, compliance, minter and pauser.

2.3. Findings

During the audit process, the audit team found some vulnerabilities in the given version of StableCoin.

#	Issue	Severity	Status
1	Can bypass the fee charge by using a contract to transfer tokens	MEDIUM	Acknowledged
2	Calculates incorrect allowance if a contract transfers tokens	MEDIUM	Acknowledged
3	Redundant code for charging fee	INFORMATIVE	Acknowledged

2.3.1. Can bypass the fee charge by using a contract to transfer tokens - MEDIUM

Affected files:

- contracts/bases/TRC25.sol

A TRC25 contract is like an ERC20 token, but with some additional features, including a fee on transfer.

However, the fee can be bypassed by using a contract to transfer tokens. See the code

```
function transfer(
    address recipient,
    uint256 amount
) external override returns (bool) {
    uint256 fee = estimateFee(amount);
    _transfer(msg.sender, recipient, amount);
    _chargeFeeFrom(msg.sender, recipient, fee);
```

```
    return true;
}

function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) external override returns (bool) {
    uint256 fee = estimateFee(amount);
    require(
        _allowances[sender][msg.sender] >= amount + fee,
        "TRC25: amount exeeds allowance"
    );

    _allowances[sender][msg.sender] =
        _allowances[sender][msg.sender] -
        amount -
        fee;
    _transfer(sender, recipient, amount);
    _chargeFeeFrom(sender, recipient, fee);
    return true;
}

function _chargeFeeFrom(
        address sender,
        address recipient,
        uint256 amount
    ) internal {
    if (address(msg.sender).code.length > 0) {
        return;
    }
    if (amount > 0) {
        _transfer(sender, _owner, amount);
        emit Fee(sender, recipient, _owner, amount);
    }
}
```

Following the code, when you transfer tokens to another user, you will be charged a fee; a contract transfer will not be charged a fee because of the check `address(msg.sender).code.length > 0` in the `_chargeFeeFrom` function.

If the user transfers tokens to a contract, the contract can transfer tokens to another one without charging a fee.

RECOMMENDATION

We recommend charging a fee to any receiver instead of the sender. Example patched code:

```
function transfer(
    address recipient,
```

```
    uint256 amount
) external override returns (bool) {
    uint256 fee = estimateFee(amount);
    _transfer(msg.sender, recipient, amount - fee);
    return true;
}
```

UPDATES

The Stably team has acknowledged this issue and given feedback that this is a design decision. The purpose of charging a fee is for the owner to sponsor gas for the token holder when they perform normal actions with the token. They purposefully do not sponsor smart contract transactions since it will break various DeFi protocols such as AMMs.

2.3.2. Calculates incorrect allowance if a contract transfers tokens - MEDIUM

Affected files:

- contracts/bases/TRC25.sol

The `TRC25` contract has a function to charge a fee when transferring tokens. However, the allowance is calculated incorrectly if a contract transfers tokens. The allowance subtracts the fee even though it does not charge a fee for a contract.

```
function transferFrom(
        address sender,
        address recipient,
        uint256 amount
) external override returns (bool) {
    uint256 fee = estimateFee(amount);
    require(
        _allowances[sender][msg.sender] >= amount + fee,
        "TRC25: amount exeeds allowance"
    );


    _allowances[sender][msg.sender] =
        _allowances[sender][msg.sender] -
        amount -
        fee;
    _transfer(sender, recipient, amount);
    _chargeFeeFrom(sender, recipient, fee);
    return true;
}

function _chargeFeeFrom(
        address sender,
        address recipient,
        uint256 amount
```

```
    ) internal {
    if (address(msg.sender).code.length > 0) {
        return;
    }
    if (amount > 0) {
        _transfer(sender, _owner, amount);
        emit Fee(sender, recipient, _owner, amount);
    }
}
```

RECOMMENDATION

We recommend subtracting the allowance of the fee only if the sender is a user, not a contract.

UPDATES

The Stably team has acknowledged this issue and given feedback that this is a departure from the typical way "gas fees" are charged in crypto. In the context of TRC25 and Tomoz, the goal is to help less experienced crypto users interact with Web3, and requiring the spender to hold the same token as the approver would increase that difficulty. Thus, it's an intentional decision to charge the approver a fee and include the fee in the allowance to prevent abuse of approval amounts.

2.3.3. Redundant code for charging fee - INFORMATIVE

Affected files:

- contracts/bases/TRC25.sol
- contracts/extensions/TRC25Permit.sol

The `TRC25` contract has a function to charge a fee when transferring tokens. However, some line of codes are redundant by charge zero fee at line 170 and 209.

Similarly, the redundant also appears in the `TRC25Permit` contract at line 66.

```
function approve(
        address spender,
        uint256 amount
    ) external override returns (bool) {
    uint256 fee = estimateFee(0);
    _approve(msg.sender, spender, amount);
    _chargeFeeFrom(msg.sender, address(this), fee);
    return true;
}
```

RECOMMENDATION

We recommend removing the redundant code.

UPDATES

The Stably team has acknowledged this issue and given feedback that the intention is that every public action should be protected by a fee to prevent abuse. Note that an estimate of 0 does not mean that no fee is charged, it just means that the amount being estimated is 0.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Sep 15, 2023*	Private Report	Verichains Lab
1.1	*Sep 21, 2023*	Public Report	Verichains Lab

Table 2. Report versions history