TCASH Chain

Security Assessment

CertiK Assessed on Nov 20th, 2024

CertiK Assessed on Nov 20th, 2024

TCASH Chain

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES
ERC-20

ECOSYSTEM
Binance Smart Chain (BSC)

METHODS
Formal Verification, Manual Review, Static Analysis

LANGUAGE
Solidity

TIMELINE
Delivered on 11/20/2024

KEY COMPONENTS
N/A

CODEBASE
private codebase
View All in Codebase Page

COMMITS
zip hash(openssl dgst -sha256):
1d9af508826f808ef4b144324cfb5a86132c780e0cc378171e88637a574e69a6
View All in Codebase Page

Vulnerability Summary

6	0	0	0	6	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
2	Major	2 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
2	Minor	2 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
2	Informational	2 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | TCASH CHAIN

CODEBASE | TCASH CHAIN

Repository

private codebase

Commit

zip hash(openssl dgst -sha256): 1d9af508826f808ef4b144324cfb5a86132c780e0cc378171e88637a574e69a6

AUDIT SCOPE | TCASH CHAIN

1 file audited ● 1 file with Acknowledged findings

ID	Repo	File	SHA256 Checksum
● TER	CertiKProject/certik-audit-projects	contracts/TcashERC20.sol	44dfbe58d220317679e48edb6c15d1fe30d01e9e472a3927d98d59cd613806c2

APPROACH & METHODS | TCASH CHAIN

This report has been prepared for TCASH to discover issues and vulnerabilities in the source code of the TCASH Chain project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis, Formal Verification, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTES | TCASH CHAIN

Overview

TCASH is a deflationary token project. The contract owner will set a `mainPair`. Except for special addresses, transferring tokens to or receiving tokens from the `mainPair` will incur a fee.

External Dependencies

In **TCASH**, the module inherits or uses a few of the depending injection contracts or addresses to fulfill the need of its business logic. The scope of the audit treats third party entities as black boxes and assume their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets.

Addresses

The following addresses interact at some point with specified contracts, making them an external dependency. All of the following values are initialized either at deployment time or by specific functions in smart contracts.

- `buyTaxReceiver`, `sellTaxReceiver`, `mainPair`.

We assume these contracts or addresses are valid and non-vulnerable actors and implementing proper logic to collaborate with the current project.

Privileged Functions

In the **TCASH** project, several roles are adopted to operate with some of the key functionalities of the implemented contract. Such privileged roles and methods are described in the findings:

- **TER-02: Centralization Risks In TcashERC20.sol**
- **TER-03: Centralized Balance Manipulation**

The advantage of this privileged role in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community. It is also worth noting the potential drawbacks of these functions, which should be clearly stated through the client's action/plan. Additionally, if the private key of the privileged account is compromised, it could have devastating consequences for the project.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

FINDINGS | TCASH CHAIN

6	0	2	0	2	2
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for TCASH Chain. Through this audit, we have uncovered 6 issues ranging from different severity levels. Utilizing the techniques of Static Analysis, Formal Verification & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

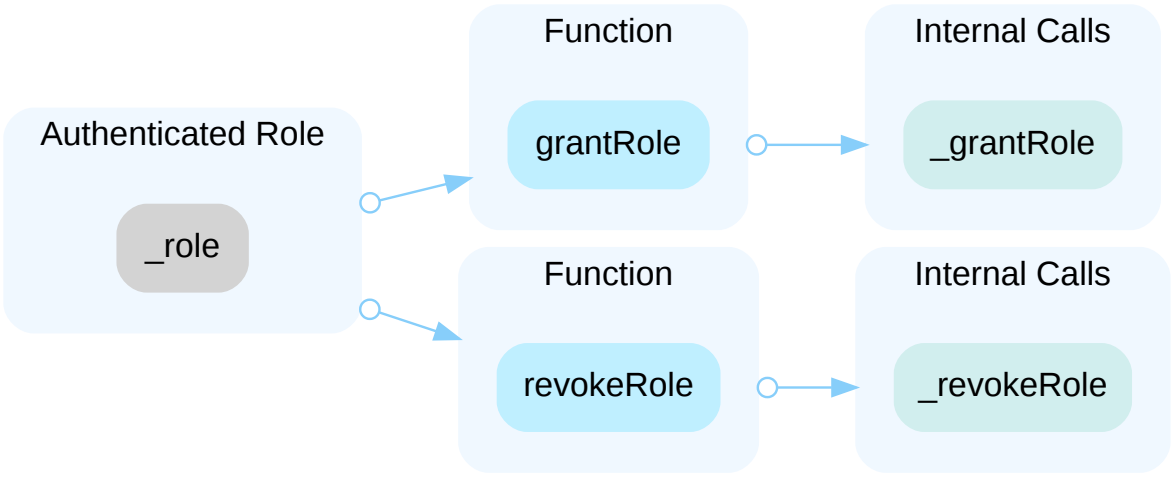
ID	Title	Category	Severity	Status
TER-02	**Centralization Risks In TcashERC20.Sol**	**Centralization**	**Major**	**Acknowledged**
TER-03	**Centralized Balance Manipulation**	**Centralization**	**Major**	**Acknowledged**
GLOBAL-01	Outdated Library And Contract Implementation	Coding Issue	Minor	Acknowledged
TER-04	Missing Zero Address Validation	Volatile Code	Minor	Acknowledged
TER-05	Missing Emit Events	Coding Style	Informational	Acknowledged
TER-06	Contracts With Todos	Coding Issue	Informational	Acknowledged

TER-02 | CENTRALIZATION RISKS IN TCASHERC20.SOL

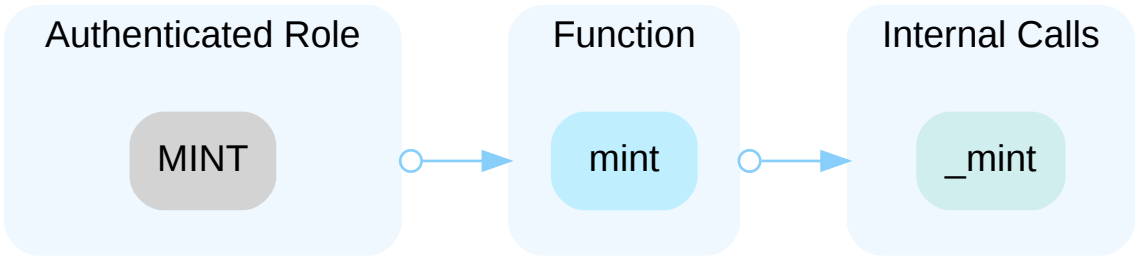
Category	Severity	Location	Status
Centralization	Major	contracts/TcashERC20.sol: 1049, 1067, 1267, 1333, 1337	Acknowledged

Description

In the contract `AccessControl`, the role `_role` has authority over the functions shown in the diagram below. Any compromise to the `_role` account may allow the hacker to take advantage of this authority and grant a role to an account, revoke a role from an account.

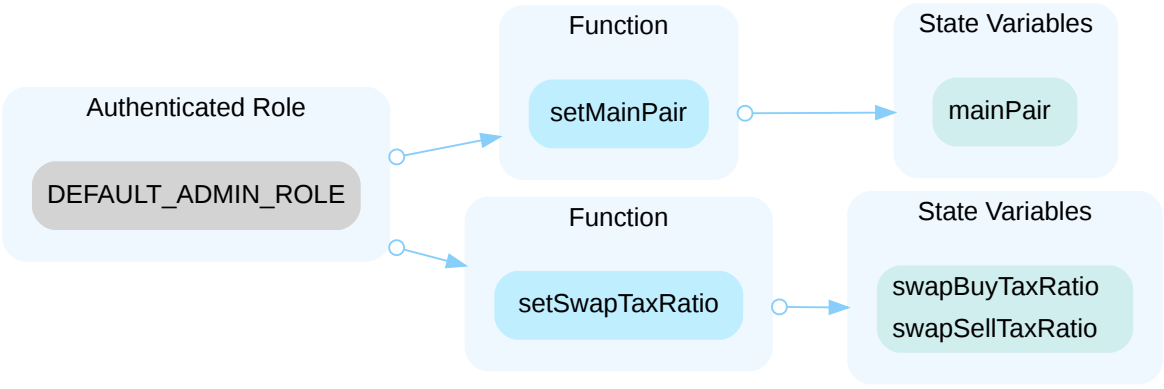


In the contract `TcashERC20Token`, the role `MINT` has authority over the functions shown in the diagram below. Any compromise to the `MINT` account may allow the hacker to take advantage of this authority and mint tokens to specified accounts.



In the contract `TcashWithPremium`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below.

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and set the main pair address, set the swap tax ratios.



Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[TCASH Team, 11/19/2024]:

Issue acknowledged. I won't make any changes for the current version.

TER-03 | CENTRALIZED BALANCE MANIPULATION

Category	Severity	Location	Status
Centralization	**Major**	**contracts/TcashERC20.sol: 1267**	**Acknowledged**

Description

In the contract `TcashERC20Token`, the role `MINT` has the authority to update the token balance of an arbitrary account without sanity restriction.

Any compromise to the `MINT` account may allow a hacker to take advantage of this authority and manipulate users' balances. For example, The hacker could also update his/her balance to a large number, sell these tokens, and cause the token price to drop.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND

- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[TCASH Team, 11/19/2024]:

Issue acknowledged. I won't make any changes for the current version.

GLOBAL-01 OUTDATED LIBRARY AND CONTRACT IMPLEMENTATION

Category	Severity	Location	Status
Coding Issue	Minor		Acknowledged

Description

The `TCash` token contract integrates several libraries and contracts to implement its functionalities. However, these libraries and contracts are outdated and may have unresolved issues. For instance, the EIP721 contract does not address the delegatecall issue#2852.

Recommendation

It is recommended to fork the latest version of these libraries and contracts from OpenZeppelin.

Alleviation

[TCASH Team, 11/19/2024]:

Issue acknowledged. I won't make any changes for the current version.

TER-04 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/TcashERC20.sol: 1324, 1325, 1334	Acknowledged

Description

The cited address input is missing a check that it is not `address(0)` .

Recommendation

We recommend adding a check the passed-in address is not `address(0)` to prevent unexpected errors.

Alleviation

[TCASH Team, 11/19/2024]:

Issue acknowledged. I won't make any changes for the current version.

TER-05 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	contracts/TcashERC20.sol: 1333	Acknowledged

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

[TCASH Team, 11/19/2024]:

Issue acknowledged. I won't make any changes for the current version.

TER-06 | CONTRACTS WITH TODOS

Category	Severity	Location	Status
Coding Issue	Informational	contracts/TcashERC20.sol: 493, 750	Acknowledged

Description

"TODO" comments within smart contract code could signal potential vulnerabilities due to the presence of undeveloped or incomplete logic. It is also possible that these comments were left behind after the completion of the intended features, indicating a lack of code cleanup and final review.

Additionally, if "TODO" features are implemented post-audit, there is a risk of introducing new vulnerabilities that were not present during the initial security assessment.

Recommendation

To mitigate this issue, it's important to:

1. Finalize all contract features and logic before deployment, removing any "TODO" comments to ensure the code is complete.
2. Conduct a comprehensive audit of the smart contract after any significant updates or additions, including those previously marked as "TODO."

Alleviation

[TCASH Team, 11/19/2024]:

Issue acknowledged. I won't make any changes for the current version.

OPTIMIZATIONS | TCASH CHAIN

ID	Title	Category	Severity	Status
TER-01	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	Acknowledged

TER-01 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/TcashERC20.sol: 1298, 1299	Acknowledged

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

Alleviation

[TCASH Team, 11/19/2024]:

Issue acknowledged. I won't make any changes for the current version.

FORMAL VERIFICATION | TCASH CHAIN

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply`, which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-transferfrom-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-transferfrom-revert-zero-argument	`transferFrom` Fails for Transfers with Zero Address Arguments
erc20-transfer-never-return-false	`transfer` Never Returns `false`
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transfer-false	If `transfer` Returns `false`, the Contract State Is Not Changed
erc20-transfer-recipient-overflow	`transfer` Prevents Overflows in the Recipient's Balance

Property Name	Title
erc20-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Transfers
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Transfers
erc20-transferfrom-fail-recipient-overflow	`transferFrom` Prevents Overflows in the Recipient's Balance
erc20-approve-never-return-false	`approve` Never Returns `false`
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-approve-false	If `approve` Returns `false`, the Contract's State Is Unchanged
erc20-approve-succeed-normal	`approve` Succeeds for Valid Inputs
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds

Verification of contracts derived from AccessControl v4.4

We verified properties of the public interface of contracts that provide an AccessControl-v4.4 compatible API. This involves:

- The `hasRole` function, which returns `true` if an account has been granted a specific `role`.
- The `getRoleAdmin` function, which returns the admin role that controls a specific `role`.
- The `grantRole` and `revokeRole` functions, which are used for granting a `role` to an account and revoking a `role` from an `account`, respectively.

- The `renounceRole` function, which allows the calling account to revoke a `role` from itself.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
accesscontrol-default-admin-role	AccessControl Default Admin Role Invariance
accesscontrol-hasrole-succeed-always	`hasRole` Function Always Succeeds
accesscontrol-renouncerole-succeed-role-renouncing	`renounceRole` Successfully Renounces Role
accesscontrol-getroleadmin-succeed-always	`getRoleAdmin` Function Always Succeeds
accesscontrol-hasrole-change-state	`hasRole` Function Does Not Change State
accesscontrol-getroleadmin-change-state	`getRoleAdmin` Function Does Not Change State
accesscontrol-revokerole-correct-role-revoking	`revokeRole` Correctly Revokes Role
accesscontrol-grantrole-correct-role-granting	`grantRole` Correctly Grants Role
accesscontrol-renouncerole-revert-not-sender	`renounceRole` Reverts When Caller Is Not the Confirmation Address

Verification Results

For the following contracts, formal verification established that each of the properties that were in scope of this audit (see scope) are valid:

Detailed Results For Contract TcashERC20Token (contracts/TcashERC20.sol) In SHA256 Checksum aff2dd101eeb21dff279291752fff310c61524a9

Verification of ERC-20 Compliance

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-change-state	True	
erc20-totalsupply-correct-value	True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	
erc20-balanceof-succeed-always	True	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-change-state	True	
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-false	True	
erc20-transferfrom-revert-zero-argument	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-fail-recipient-overflow	True	

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-never-return-false	True	
erc20-transfer-revert-zero	True	
erc20-transfer-false	True	
erc20-transfer-recipient-overflow	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-correct-amount	True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-never-return-false	True	
erc20-approve-revert-zero	True	
erc20-approve-correct-amount	True	
erc20-approve-false	True	
erc20-approve-succeed-normal	True	

Detailed Results For Contract VaultOwned (contracts/TcashERC20.sol) In SHA256 Checksum aff2dd101eeb21dff279291752fff310c61524a9

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	True	
accesscontrol-hasrole-change-state	True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-succeed-role-renouncing	True	
accesscontrol-renouncerole-revert-not-sender	True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	True	
accesscontrol-getroleadmin-change-state	True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	True	

In the remainder of this section, we list all contracts where formal verification of at least one property was not successful. There are several reasons why this could happen:

- False: The property is violated by the project.
- Inconclusive: The proof engine cannot prove or disprove the property due to timeouts or exceptions.
- Inapplicable: The property does not apply to the project.

Detailed Results For Contract TcashWithPremium (contracts/TcashERC20.sol) In SHA256 Checksum aff2dd101eeb21dff279291752fff310c61524a9

Verification of ERC-20 Compliance

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-fail-recipient-overflow	Inconclusive	
erc20-transferfrom-fail-exceed-balance	Inconclusive	
erc20-transferfrom-correct-amount	Inconclusive	
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-false	True	
erc20-transferfrom-revert-zero-argument	True	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-change-state	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-succeed-always	True	

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-recipient-overflow	Inconclusive	
erc20-transfer-exceed-balance	Inconclusive	
erc20-transfer-correct-amount	Inconclusive	
erc20-transfer-never-return-false	True	
erc20-transfer-false	True	
erc20-transfer-revert-zero	True	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	
erc20-allowance-change-state	True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-succeed-normal	True	
erc20-approve-revert-zero	True	
erc20-approve-correct-amount	True	
erc20-approve-never-return-false	True	
erc20-approve-false	True	

APPENDIX | TCASH CHAIN

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well

as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed AccessControl-v4.4 Properties

Properties related to function `DEFAULT_ADMIN_ROLE`

accesscontrol-default-admin-role

The default admin role must be invariant, ensuring consistent access control management.

Specification:

```
invariant DEFAULT_ADMIN_ROLE() == 0x00;
```

Properties related to function `hasRole`

accesscontrol-hasrole-change-state

The `hasRole` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-hasrole-succeed-always

The `hasRole` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `renounceRole`

accesscontrol-renouncerole-revert-not-sender

The `renounceRole` function must revert if the caller is not the same as `account`.

Specification:

```
reverts_when account != msg.sender;
```

accesscontrol-renouncerole-succeed-role-renouncing

After execution, `renounceRole` must ensure the caller no longer has the renounced role.

Specification:

```
ensures !hasRole(role, account);
```

Properties related to function `getRoleAdmin`

accesscontrol-getroleadmin-change-state

The `getRoleAdmin` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-getroleadmin-succeed-always

The `getRoleAdmin` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `revokeRole`

accesscontrol-revokerole-correct-role-revoking

After execution, `revokeRole` must ensure the specified account no longer has the revoked role.

Specification:

```
ensures !hasRole(role, account);
```

Properties related to function `grantRole`

accesscontrol-grantrole-correct-role-granting

After execution, `grantRole` must ensure the specified account has the granted role.

Specification:

```
ensures hasRole(role, account);
```

Description of the Analyzed ERC-20 Properties

Properties related to function `totalSupply`

erc20-totalsupply-change-state

The `totalSupply` function in contract TcashERC20Token must not change any state variables.

Specification:

```
assignable \nothing;
```

erc20-totalsupply-change-state

The `totalSupply` function in contract TcashWithPremium must not change any state variables.

Specification:

```
assignable \nothing;
```

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract TcashERC20Token.

Specification:

```
ensures \result == totalSupply();
```

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract TcashWithPremium.

Specification:

```
ensures \result == totalSupply();
```

erc20-totalsupply-succeed-always

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `balanceOf`

erc20-balanceof-change-state

Function `balanceOf` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-balanceof-correct-value

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner`.

Specification:

```
ensures \result == balanceOf(\old(account));
```

erc20-balanceof-succeed-always

Function `balanceOf` must always succeed if it does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `allowance`

erc20-allowance-change-state

Function `allowance` must not change any of the contract's state variables.

Specification:

```
assignable \nothing;
```

erc20-allowance-correct-value

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

Specification:

```
ensures \result == allowance(\old(owner), \old(spender));
```

erc20-allowance-succeed-always

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `transferFrom`

erc20-transferfrom-correct-allowance

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount`.

Specification:

```
ensures \result ==> allowance(\old(sender), msg.sender) == \old(allowance(sender,
msg.sender)) - \old(amount)
                  || (allowance(\old(sender), msg.sender) == \old(allowance(sender,
msg.sender)) && \old(allowance(sender, msg.sender)) == type(uint256).max);
```

erc20-transferfrom-correct-amount

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest`.

Specification:

```
requires recipient != sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient) +
amount)
                 && balanceOf(\old(sender)) == \old(balanceOf(sender) - amount);
  also
requires recipient == sender;
ensures \result ==> balanceOf(\old(recipient)) == \old(balanceOf(recipient));
```

erc20-transferfrom-fail-exceed-allowance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

Specification:

```
requires msg.sender != sender;
requires amount > allowance(sender, msg.sender);
ensures !\result;
```

erc20-transferfrom-fail-exceed-balance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

Specification:

```
requires amount > balanceOf(sender);
ensures !\result;
```

erc20-transferfrom-fail-recipient-overflow

Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail.

Specification:

```
requires recipient != sender;
requires balanceOf(recipient) + amount > type(uint256).max;
ensures !\result;
```

erc20-transferfrom-false

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transferfrom-never-return-false

The `transferFrom` function must never return `false`.

Specification:

```
ensures \result;
```

erc20-transferfrom-revert-zero-argument

All calls of the form `transferFrom(from, dest, amount)` must fail for transfers from or to the zero address.

Specification:

```
ensures \old(sender) == address(0) ==> !\result;
also
ensures \old(recipient) == address(0) ==> !\result;
```

Properties related to function `transfer`

erc20-transfer-correct-amount

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

Specification:

```
requires recipient != msg.sender;
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result ==> balanceOf(recipient) == \old(balanceOf(recipient) + amount)
&& balanceOf(msg.sender) == \old(balanceOf(msg.sender) - amount);
  also
requires recipient == msg.sender;
ensures \result ==> balanceOf(msg.sender) == \old(balanceOf(msg.sender));
```

erc20-transfer-exceed-balance

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

Specification:

```
requires amount > balanceOf(msg.sender);
ensures !\result;
```

erc20-transfer-false

If the `transfer` function in contract `TcashERC20Token` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transfer-false

If the `transfer` function in contract `TcashWithPremium` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-transfer-never-return-false

The transfer function must never return `false` to signal a failure.

Specification:

```
ensures \result;
```

erc20-transfer-recipient-overflow

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

Specification:

```
requires recipient != msg.sender;
requires balanceOf(recipient) + amount > type(uint256).max;
ensures !\result;
```

erc20-transfer-revert-zero

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

Specification:

```
ensures \old(recipient) == address(0) ==> !\result;
```

Properties related to function `approve`

erc20-approve-correct-amount

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`.

Specification:

```
requires spender != address(0);
ensures \result ==> allowance(msg.sender, \old(spender)) == \old(amount);
```

erc20-approve-false

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

Specification:

```
ensures !\result ==> \assigned (\nothing);
```

erc20-approve-never-return-false

The function `approve` must never returns `false`.

Specification:

```
ensures \result;
```

erc20-approve-revert-zero

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

Specification:

```
ensures \old(spender) == address(0) ==> !\result;
```

erc20-approve-succeed-normal

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

Specification:

```
requires spender != address(0);
ensures \result;
reverts_only_when false;
```

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Entire Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.