quackai.ai

Security Assessment

CertiK Assessed on Sept 1st, 2025

CertiK Assessed on Sept 1st, 2025

quackai.ai

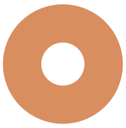
The security assessment was prepared by CertiK.

Executive Summary

TYPES	ECOSYSTEM	METHODS
ERC-20	Binance Smart Chain (BSC)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE
Solidity	Preliminary comments published on 09/01/2025 Final report published on 09/01/2025

Vulnerability Summary



2	0	0	2	0
Total Findings	Resolved	Partially Resolved	Acknowledged	Declined

2	Centralization	2 Acknowledged	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.

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.
0	Major	Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
0	Medium	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
0	Minor	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.
0	Informational	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 | QUACKAI.AI

CODEBASE | QUACKAI.AI

Repository

https://bscscan.com/address/0xc07e1300dc138601FA6B0b59f8D0FA477e690589#code

AUDIT SCOPE | QUACKAI.AI

mainnet
src/tokens/BaseToken.sol
src/tokens/PeerToken.sol

APPROACH & METHODS | QUACKAI.AI

This audit was conducted for quackai to evaluate the security and correctness of the smart contracts associated with the quackai.ai project. The assessment included a comprehensive review of the in-scope smart contracts. The audit was performed using a combination of Static Analysis and Manual Review.

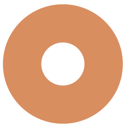
The review process emphasized the following areas:

- Architecture review and threat modeling to understand systemic risks and identify design-level flaws.
- Identification of vulnerabilities through both common and edge-case attack vectors.
- Manual verification of contract logic to ensure alignment with intended design and business requirements.
- Dynamic testing to validate runtime behavior and assess execution risks.
- Assessment of code quality and maintainability, including adherence to current best practices and industry standards.

The audit resulted in findings categorized across multiple severity levels, from informational to critical. To enhance the project's security and long-term robustness, we recommend addressing the identified issues and considering the following general improvements:

- Improve code readability and maintainability by adopting a clean architectural pattern and modular design.
- Strengthen testing coverage, including unit and integration tests for key functionalities and edge cases.
- Maintain meaningful inline comments and documentations.
- Implement clear and transparent documentation for privileged roles and sensitive protocol operations.
- Regularly review and simulate contract behavior against newly emerging attack vectors.

FINDINGS | QUACKAI.AI



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

This report has been prepared for quackai to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the audit, a total of 2 issues were identified. Leveraging a combination of Static Analysis & Manual Review the following findings were uncovered:

ID	Title	Category	Severity	Status
QUA-01	**Centralized Balance Manipulation**	**Centralization**	**Centralization**	**Acknowledged**
QUA-02	**Centralization Risks In PeerToken.Sol**	**Centralization**	**Centralization**	**Acknowledged**

QUA-01 | Centralized Balance Manipulation

Category	Severity	Location	Status
Centralization	Centralization	PeerToken.sol: 31	Acknowledged

Description

In the contract `PeerToken` , the `minter` role has the authority to arbitrarily increase the token balance of any account by minting new tokens, without any cap or sanity restriction. This is done via the `mint(address _account, uint256 _amount) function` .

Any compromise to the `minter` account may allow a hacker to take advantage of this authority and manipulate users' balances. For example, the hacker could mint a massive number of tokens to an address they control. They could then sell these tokens on a decentralized exchange, draining the liquidity pool of its valuable assets and causing the token's price to collapse, thereby inflicting significant financial losses on all other token holders.

Recommendation

We recommend the team makes efforts to restrict access to the private key of the privileged account. A strategy of multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to mint more tokens or engage in similar balance-related operations.
Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently *fully* resolve the risk:

Short Term:

A multi signature (⅔, ⅗) wallet *mitigate* the risk by avoiding a single point of key management failure.

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers' addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A DAO for controlling the operation *mitigate* the risk by applying transparency and decentralization.

- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the multi-signers' addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

The following actions can *fully* resolve the risk:

- Renounce the ownership and never claim back the privileged role.
  OR
- Remove the risky functionality.
  OR
- Add minting logic (such as a vesting schedule) to the contract instead of allowing the owner account to call the sensitive function directly.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[quackai, 09/01/2025]: We understand the concern regarding unlimited mint rights. However, in our deployment the minter is not an externally controlled wallet, but the NTT Manager contract provided by Wormhole's Native Token Transfer (NTT) framework. This significantly changes the risk model:

Minter Role = NTT Manager The PeerToken's minter address has been set to the NTT Manager contract (0x50885eD2D42FaaD83915c8524F6288afec6444ED on BNB Chain).

This contract is not able to arbitrarily mint; it only mints when triggered by a valid Wormhole attestation. Minting Requires a Valid VAA Any call to mint() inside the NTT Manager requires a Verified Action Approval (VAA) signed by the Wormhole Guardian network. The Guardian set consists of 19 independent nodes, and at least 13 of 19 must co-sign a message before a VAA is valid. Without such decentralized consensus, the NTT Manager cannot mint tokens - the transaction will revert.

Cross-Chain Security This ensures that supply changes on BNB Chain (mint) or Ethereum (burn) are fully secured by decentralized, off-chain consensus, rather than controlled by any single developer or admin. In practice, this is equivalent to saying that our token supply extension is gated by Wormhole's security model, not by project owners.

QUA-02 | Centralization Risks In PeerToken.Sol

Category	Severity	Location	Status
Centralization	Centralization	PeerToken.sol: 31, 35	Acknowledged

Description

The `PeerToken` contract designates two privileged roles, `owner` and `minter`, which serve as single points of failure for the entire token ecosystem. The security of all token holders' assets is dependent on the operational security of the private keys controlling these two accounts. A compromise of either key would allow an attacker to inflict damage on the token's value and market liquidity.

1. The `minter` Role
   The `minter` role has the authority to create an unlimited number of new tokens and assign them to any address by calling the mint() function. An attacker who compromises the `minter` account can mint an arbitrarily large supply of tokens to their own wallet. They can then sell these tokens on a decentralized exchange, draining the liquidity pools of all valuable assets (e.g., ETH, stablecoins) and causing a complete price collapse, effectively destroying the token's value for all other holders.

2. The `owner` Role
   The `owner` role has authority over the token supply's control. By calling the setMinter() function, the owner can appoint any address as the new `minter`. An attacker who compromises the `owner` account can immediately call setMinter() to designate an address they control as the new official `minter`. This effectively grants the attacker the same unlimited minting privileges described above, leading to the economic attack. The `owner` role represents an indirect but control over the token's monetary policy.

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

[quackai, 09/01/2025]: We acknowledge CertiK's observation regarding centralization risks in the owner and minter roles of the PeerToken contract. To remediate these risks, we have implemented the following governance and security measures:

Owner Role Secured by Timelock The owner role of the PeerToken contract has been transferred to a dedicated OpenZeppelin TimelockController contract (address: 0xf4c4776830352a230560e59bed5e251dac470cc4). The Timelock is configured with a minimum delay of 48 hours (172,800 seconds) before any owner action (e.g., assigning a new minter) can be executed. This ensures that sensitive changes must follow the sequence: propose, queue, delay, execute, preventing instant modifications.

Multisignature Control of Timelock Roles All privileged roles of the Timelock (ADMIN_ROLE, PROPOSER_ROLE, EXECUTOR_ROLE) are controlled by a 2-of-3 Safe multisignature wallet (address: 0xD46CFe41A9FECcA97C3119226e386837e08324C8). This setup eliminates single-key control and ensures that no individual can unilaterally trigger governance operations. The combination of multisig + timelock directly implements CertiK's recommendation to avoid single points of failure by requiring both multi-party consent and a time delay.

Standards and Audit Alignment The Timelock uses the OpenZeppelin standard implementation, which is widely adopted and recognized as an industry best practice. Its security properties and delay mechanism have been extensively battle-tested across DeFi protocols. This governance setup is fully on-chain, transparent, and can be independently verified by external parties (auditors, exchanges, community). Transparency Commitments We will publish and maintain records of the Safe multisig address and Timelock configuration so that the community and partners can independently verify our governance setup. All governance actions will be observable on-chain, with a 48-hour window providing sufficient time for monitoring and community oversight. Future Governance Roadmap While the current Safe + Timelock structure provides strong operational security and aligns with CertiK's short-term recommendations, we are committed to further decentralization.

As the protocol matures, governance over token roles (e.g., owner/minter management) will progressively transition to the Quack AI DAO, enabling token-holder based decision-making with increased transparency and decentralization.

FORMAL VERIFICATION | QUACKAI.AI

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-change-state	`totalSupply` Does Not Change the Contract's State
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Transfers
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Transfers
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-fail-recipient-overflow	`transferFrom` Prevents Overflows in the Recipient's Balance
erc20-transfer-recipient-overflow	`transfer` Prevents Overflows in the Recipient's Balance

Property Name	Title
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-transfer-never-return-false	`transfer` Never Returns `false`
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-transferfrom-false	If `transferFrom` Returns `false` , the Contract's State Is Unchanged
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address
erc20-transferfrom-revert-zero-argument	`transferFrom` Fails for Transfers with Zero Address Arguments
erc20-transfer-false	If `transfer` Returns `false` , the Contract State Is Not Changed
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20-approve-never-return-false	`approve` Never Returns `false`
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address
erc20-approve-false	If `approve` Returns `false` , the Contract's State Is Unchanged
erc20-approve-succeed-normal	`approve` Succeeds for Valid Inputs

Verification Results

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 PeerToken (src/tokens/PeerToken.sol) In Commit

0xc07e1300dc138601fa6b0b59f8d0fa477e690589

Verification of ERC-20 Compliance

Detailed Results for Function `totalSupply`

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

Detailed Results for Function `balanceOf`

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

Detailed Results for Function `transferFrom`

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

Detailed Results for Function `transfer`

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

Detailed Results for Function `allowance`

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

Detailed Results for Function `approve`

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

Detailed Results For Contract BaseToken (src/tokens/BaseToken.sol) In Commit 0xc07e1300dc138601fa6b0b59f8d0fa477e690589

Verification of ERC-20 Compliance

Detailed Results for Function `totalSupply`

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

Detailed Results for Function `transferFrom`

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

Detailed Results for Function `transfer`

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

Detailed Results for Function `allowance`

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

Detailed Results for Function `balanceOf`

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

Detailed Results for Function `approve`

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

APPENDIX | QUACKAI.AI

Finding Categories

Categories	Description
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.

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 ERC-20 Properties

Properties related to function `totalSupply`

erc20-totalsupply-change-state

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

Specification:

```
assignable \nothing;
```

erc20-totalsupply-change-state

The `totalSupply` function in contract BaseToken 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 BaseToken.

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 PeerToken.

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 `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 `PeerToken` 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 `BaseToken` 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 `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 `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 Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is the largest 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.