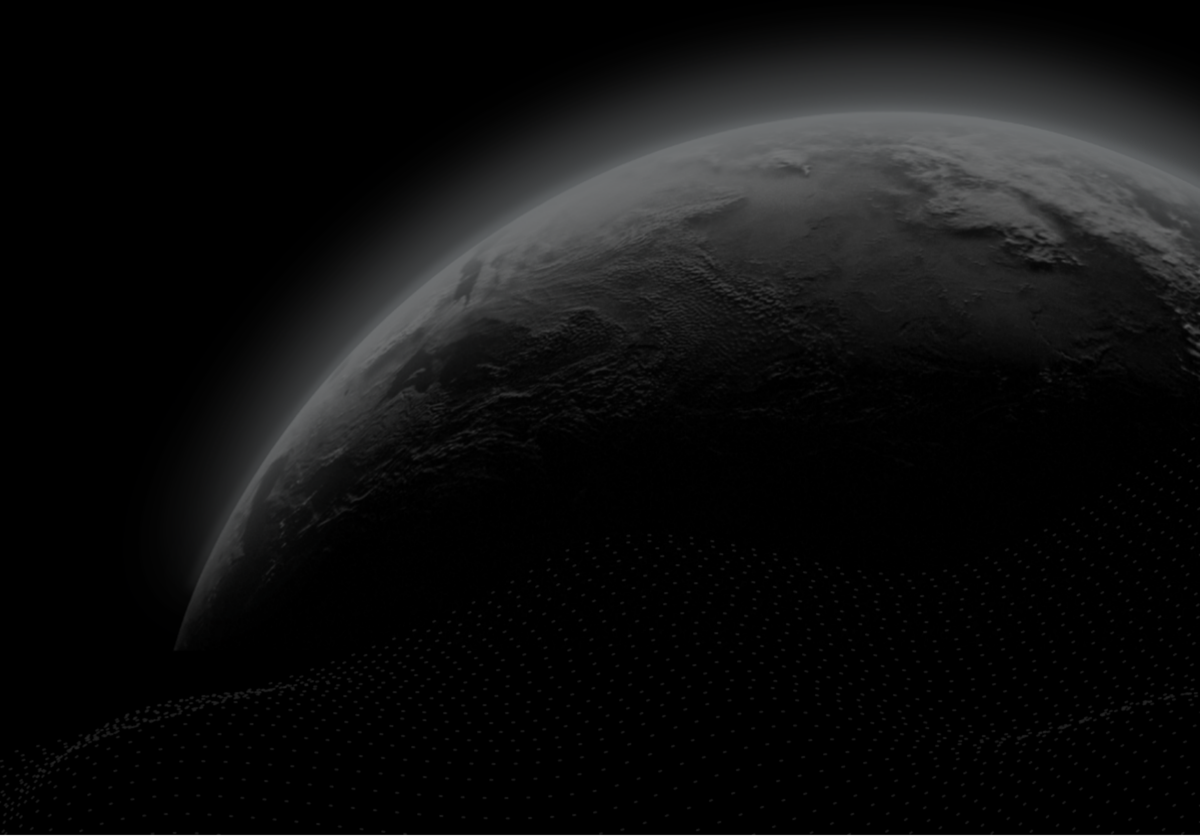
Security Assessment

Tradinggpt - Audit

CertiK Assessed on Jan 22nd, 2024



CertiK Assessed on Jan 22nd, 2024

Tradinggpt - Audit

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

Executive Summary

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

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 01/22/2024	N/A

CODEBASE

https://bscscan.com/token/0x819266d89504dc4dd3cedb440bff29da083446c0#code

View All in Codebase Page

Highlighted Centralization Risks

- Fees are unbounded
- Initial owner token share is 100%
- Has blacklist/whitelist

Vulnerability Summary

7	0	0	0	7	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.
2	Medium	2 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
3	Minor	3 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.
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 | TRADINGGPT - AUDIT

CODEBASE | TRADINGGPT - AUDIT

Repository

https://bscscan.com/token/0x819266d89504dc4dd3cedb440bff29da083446c0#code

AUDIT SCOPE | TRADINGGPT - AUDIT

1 file audited ● 1 file with Acknowledged findings

ID	Repo	File	SHA256 Checksum
● TGP	mainnet	TradingGPT.sol	edf4e5a0fef8f928c654f4c740f92a9a051b5d0f4cc291dde65d81e56130e5bf

APPROACH & METHODS | TRADINGGPT - AUDIT

This report has been prepared for Tradinggpt to discover issues and vulnerabilities in the source code of the Tradinggpt - Audit 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 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.

FINDINGS | TRADINGGPT - AUDIT

7	0	2	2	3	0
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Tradinggpt - Audit. Through this audit, we have uncovered 7 issues ranging from different severity levels. Utilizing the techniques of Static Analysis & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
TGP-02	**Initial Token Distribution**	**Centralization**	**Major**	**Acknowledged**
TGP-03	**Centralization Risks In TradingGPT.Sol**	**Centralization**	**Major**	**Acknowledged**
TGP-04	No Upper Limit For Fee	Logical Issue	Medium	Acknowledged
TGP-05	Locked Blockchain Native Tokens	Inconsistency	Medium	Acknowledged
TGP-06	Third-Party Dependency Usage	Design Issue	Minor	Acknowledged
TGP-07	Missing Zero Address Validation	Volatile Code	Minor	Acknowledged
TGP-08	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Acknowledged

TGP-02 | INITIAL TOKEN DISTRIBUTION

Category	Severity	Location	Status
Centralization	Major	TradingGPT.sol: 1497, 1497	Acknowledged

Description

All of the `TradingGPT` tokens are sent to the externally-owned account (EOA) address `poolWallet` . This is a centralization risk because the owner(s) of the EOAs can distribute tokens without obtaining the consensus of the community. Any compromise to these addresses may allow a hacker to steal and sell tokens on the market, resulting in severe damage to the project.

Recommendation

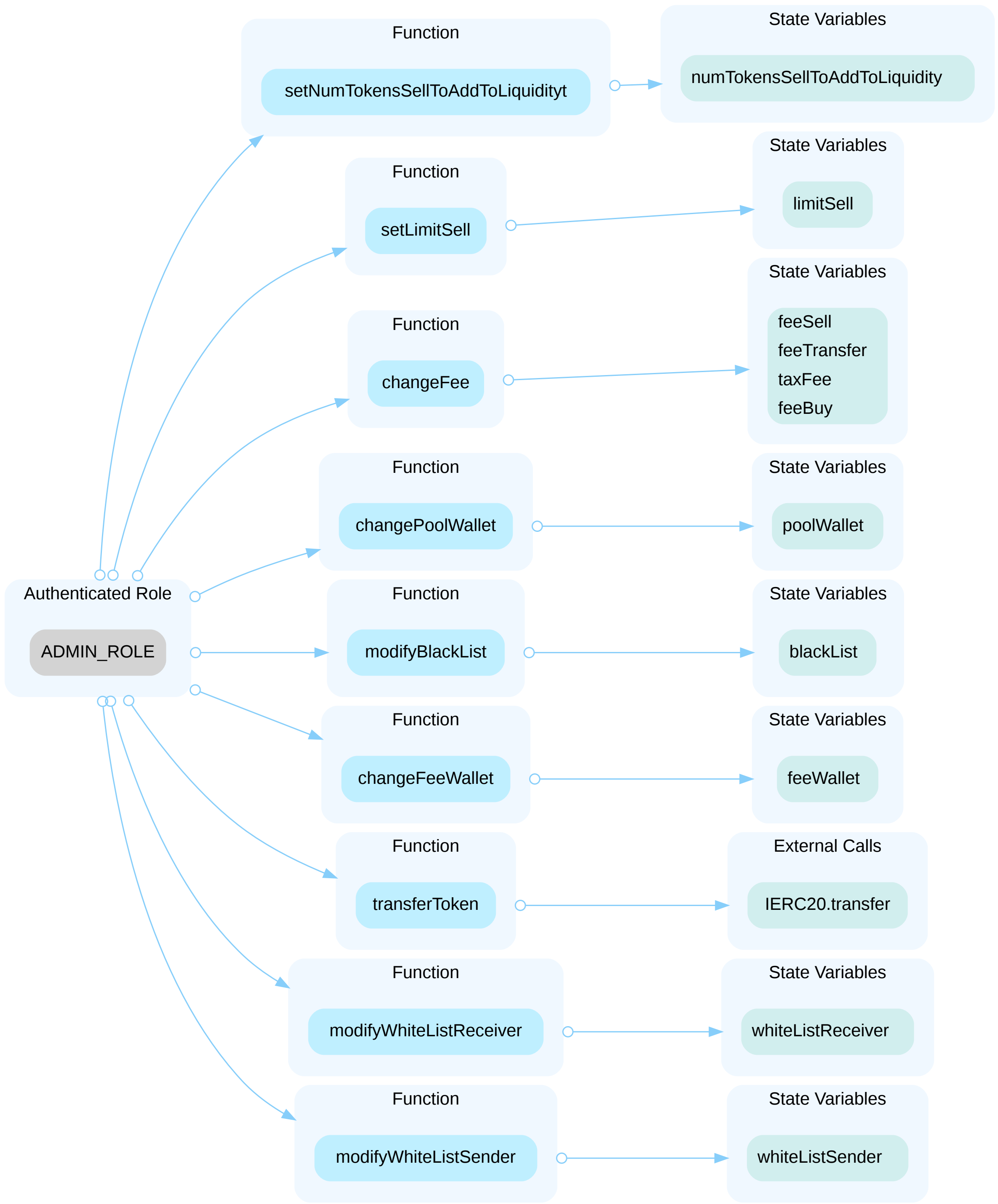
It is recommended that the team be transparent regarding the initial token distribution process. The token distribution plan should be published in a public location that the community can access. The team should make efforts to restrict access to the private keys of the deployer account or EOAs. A multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. Additionally, the team can lock up a portion of tokens, release them with a vesting schedule for long-term success, and deanonymize the project team with a third-party KYC provider to create greater accountability.

TGP-03 | CENTRALIZATION RISKS IN TRADINGGPT.SOL

Category	Severity	Location	Status
Centralization	**Major**	**TradingGPT.sol: 1001, 1009, 1123, 1138, 1158, 1891, 1906, 1921, 1936, 1941, 1945, 1949, 1953, 1965**	**Acknowledged**

Description

In the contract `TradingGPT` the role `ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `ADMIN_ROLE` account may allow the hacker to take advantage of this authority and steal tokens from the contract, change various configurations to make the contract malfunction.



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

[CertiK, 01/22/2024]: It should be noted that the centralization risk issue still exists. CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

TGP-04 | NO UPPER LIMIT FOR FEE

Category	Severity	Location	Status
Logical Issue	Medium	TradingGPT.sol: 1954, 1955, 1956	Acknowledged

Description

There are no upper boundaries for `feeTransfer/feeBuy/feeSell` . It is possible to set the those fees to any arbitrary amount.

Recommendation

Introduce a maximum fee threshold in the function to ensure fee values remain within acceptable limits. This safeguard will provide predictability and fairness in fee-related operations.

TGP-05 | LOCKED BLOCKCHAIN NATIVE TOKENS

Category	Severity	Location	Status
Inconsistency	Medium	TradingGPT.sol: 1972	Acknowledged

Description

The contract has a `receive()` function or payable functions, making it able to receive native tokens. However, it does not have a function to withdraw the funds, which can lead to permanently locked tokens within the contract.

```
1972        receive() external payable {}
```

Recommendation

It is suggested to either remove the `receive()` function and the payable attribute, or add a withdraw function with proper access control mechanisms.

TGP-06 | THIRD-PARTY DEPENDENCY USAGE

Category	Severity	Location	Status
Design Issue	Minor	TradingGPT.sol: 1472	Acknowledged

Description

The contract is serving as the underlying entity to interact with one or more third party protocols. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

```
1472        IUniswapV2Router02 public immutable uniswapV2Router;
```

- The contract `TradingGPT` interacts with third party contract with `IUniswapV2Router02` interface via `uniswapV2Router` .

Recommendation

The auditors understood that the business logic requires interaction with third parties. It is recommended for the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

TGP-07 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	TradingGPT.sol: 1481, 1482, 1483, 1486, 1946, 1950	Acknowledged

Description

Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities. For example, transferring tokens to a zero address can result in a permanent loss of those tokens.

```
1481            poolWallet = _poolWallet;
```

- `_poolWallet` is not zero-checked before being used.

```
1482            feeWallet = _feeWallet;
```

- `_feeWallet` is not zero-checked before being used.

```
1483            routerAddress = _routerAddress;
```

- `_routerAddress` is not zero-checked before being used.

```
1486            uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).
createPair(address(this), _usdt);
```

- `_usdt` is not zero-checked before being used.

```
1946            poolWallet = wallet;
```

- `wallet` is not zero-checked before being used.

```
1950            feeWallet = wallet;
```

- `wallet` is not zero-checked before being used.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

TGP-08 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	Minor	TradingGPT.sol: 1968	Acknowledged

Description

The return values of the `transfer()` and `transferFrom()` calls in the smart contract are not checked. Some ERC-20 tokens' transfer functions return no values, while others return a bool value, they should be handled with care. If a function returns `false` instead of reverting upon failure, an unchecked failed transfer could be mistakenly considered successful in the contract.

```
1968        IERC20(_tokenAddress).transfer(receiveWallet, value);
```

Recommendation

It is advised to use the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if false is returned, making it compatible with all ERC-20 token implementations.

OPTIMIZATIONS | TRADINGGPT - AUDIT

ID	Title	Category	Severity	Status
TGP-01	Unnecessary Use Of SafeMath	Coding Issue	Optimization	Acknowledged

TGP-01 | UNNECESSARY USE OF SAFEMATH

Category	Severity	Location	Status
Coding Issue	Optimization	TradingGPT.sol: 712, 1572, 1575, 1577, 1582, 1645, 1648, 1650, 1655	Acknowledged

Description

The `SafeMath` library is used unnecessarily. With Solidity compiler versions 0.8.0 or newer, arithmetic operations will automatically revert in case of integer overflow or underflow.

```
712  library SafeMath {
```

- An implementation of `SafeMath` library is found.

```
1449      using SafeMath for uint256;
```

- `SafeMath` library is used for `uint256` type in `TradingGPT` contract.

```
1572                  fee = amount.mul(feeBuy).div(percentage);
```

- `SafeMath.mul` is called in `transfer` function of `TradingGPT` contract.

Note: Only a sample of 2 `SafeMath` library usage in this contract (out of 15) are shown above.

Recommendation

We advise removing the usage of `SafeMath` library and using the built-in arithmetic operations provided by the Solidity programming language.

FORMAL VERIFICATION | TRADINGGPT - AUDIT

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 (note that overflow properties were excluded from the verification):

Property Name	Title
erc20-approve-succeed-normal	`approve` Succeeds for Valid Inputs
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-allowance-succeed-always	`allowance` Always Succeeds
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable

Property Name	Title
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-transferfrom-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-totalsupply-change-state	`totalSupply` 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-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-succeed-normal	`transferFrom` Succeeds on Valid Transfers
erc20-transferfrom-revert-zero-argument	`transferFrom` Fails for Transfers with Zero Address Arguments
erc20-transfer-never-return-false	`transfer` Never Returns `false`
erc20-transfer-false	If `transfer` Returns `false`, the Contract State Is Not Changed
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Transfers
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transfer-succeed-normal	`transfer` Succeeds on Valid Transfers
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Transfers
erc20-approve-never-return-false	`approve` Never Returns `false`
erc20-approve-false	If `approve` Returns `false`, the Contract's State Is Unchanged

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 TradingGPT (TradingGPT.sol) In Commit 0x819266d89504dc4dd3cedb440bff29da083446c0

Verification of ERC-20 Compliance

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	

Detailed Results for Function `allowance`

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

Detailed Results for Function `balanceOf`

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

Detailed Results for Function `totalSupply`

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

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-never-return-false	● True	
erc20-transferfrom-false	● True	
erc20-transferfrom-fail-exceed-balance	● True	
erc20-transferfrom-fail-exceed-allowance	● True	
erc20-transferfrom-correct-allowance	● True	
erc20-transferfrom-succeed-normal	● Inapplicable	The property does not apply to the contract
erc20-transferfrom-revert-zero-argument	● True	
erc20-transferfrom-correct-amount	● Inapplicable	The property does not apply to the contract

Detailed Results for Function `transfer`

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

APPENDIX | TRADINGGPT - AUDIT

Finding Categories

Categories	Description
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

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 ERC-20 Properties

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;
```

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 `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 `totalSupply`

erc20-totalsupply-change-state

The `totalSupply` function in contract TradingGPT 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 TradingGPT.

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 `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-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;
```

erc20-transferfrom-succeed-normal

All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from` ,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from` ,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

Specification:

```
requires recipient != address(0) && sender != address(0) && recipient != sender;
requires amount <= balanceOf(sender);
requires amount <= allowance(sender, msg.sender);
requires balanceOf(recipient) + amount <= type(uint256).max;
ensures \result;
reverts_only_when false;
```

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 `TradingGPT` 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-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;
```

erc20-transfer-succeed-normal

All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender`,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

Specification:

```
requires recipient != address(0) && recipient != msg.sender;
requires amount <= balanceOf(msg.sender);
requires balanceOf(recipient) + amount <= type(uint256).max;
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.

CertiK | **Securing** the **Web3** World

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.

