cNGN stablecoin

Security Assessment

CertiK Assessed on Dec 12th, 2024

CertiK Assessed on Dec 12th, 2024

cNGN stablecoin

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
ERC-20	Ethereum (ETH)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 12/12/2024	N/A

CODEBASE	COMMITS
Base	432c992cf5a1db6f843630a805454ad0092c8292
Update1	7e318f9676596236761ea11f88609aba7c3b70a0
Update2	9d8e59e32361b96f9f3c4ef38b4ee331eeeaaf3c
View All in Codebase Page	View All in Codebase Page

Vulnerability Summary

19	12	0	1	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.
3	Major	1 Resolved, 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.
6	Medium	6 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
5	Minor	3 Resolved, 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.
5	Informational	2 Resolved, 1 Partially Resolved, 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 | CNGN STABLECOIN

CODEBASE | CNGN STABLECOIN

Repository

Base

Update1

Update2

Commit

432c992cf5a1db6f843630a805454ad0092c8292

7e318f9676596236761ea11f88609aba7c3b70a0

9d8e59e32361b96f9f3c4ef38b4ee331eeeaaf3c

800c31ac7b8d80f3a4d7abd2452c9e1f886e9cd9

369572fc9e7111e26fe9e533e08e60b65003f90c

AUDIT SCOPE | CNGN STABLECOIN

4 files audited ● 4 files with Acknowledged findings

ID	File	SHA256 Checksum
OPE	contracts/Operations.sol	79ee35d3c9aceea26bd4bea8a29d8a31b8220a234ddd277241df991de1ba5dc8
CNG	contracts/cngn.sol	fe6eb24f901e14ee282be587e4a6c8d95c66232826c331b04b49d32edb6365e2
FOR	contracts/forwarder.sol	ade770f2fb84abdb70a22241f8408557469938a44a0a219590f206e765ac1944
CNN	tron-contract/contracts/cngn.sol	3086a186586ecbf79b967021115756b714a761894ae861d2e9fd2172eccd0420

APPROACH & METHODS | CNGN STABLECOIN

This report has been prepared for WrappedCBDC to discover issues and vulnerabilities in the source code of the cNGN stablecoin 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 | CNGN STABLECOIN

19	0	3	6	5	5
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for cNGN stablecoin. Through this audit, we have uncovered 19 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
432-01	**Centralization Related Risks**	**Centralization**	**Major**	**Acknowledged**
CON-02	**Centralized Control Of Contract Upgrade**	**Centralization**	**Major**	**Acknowledged**
OPE-01	Lack Of Blacklisting And Minting Amount Safeguards	Logical Issue	Major	Resolved
CNG-02	Unused Return Value	Volatile Code	Medium	Resolved
CON-03	`initialize()` Is Unprotected	Logical Issue	Medium	Resolved
FOR-02	Forwarder Implementation	Design Issue	Medium	Resolved
FOR-04	Missing Blacklist Check	Access Control	Medium	Resolved
FOR-05	Unused Pausable Features	Volatile Code	Medium	Resolved
FOR-09	EIP712 Standard Not Followed	Design Issue	Medium	Resolved
CNN-01	Pull-Over-Push Pattern In `transferOwnership()` Function	Logical Issue	Minor	Acknowledged
CNN-02	Minting And Burning Not Pausable	Volatile Code	Minor	Resolved

ID	Title	Category	Severity	Status
CON-04	Missing Zero Address Validation	Volatile Code	Minor	Acknowledged
FOR-07	`draft-EIP712` Is Deprecated	Language Version	Minor	Resolved
OPE-02	Privileged Roles Not Removable From Blacklisted Addresses	Inconsistency	Minor	Resolved
432-02	Event Not Indexed	Design Issue	Informational	Resolved
432-03	Missing Error Messages	Coding Style	Informational	Acknowledged
CNN-03	Experimental ABIEncoderV2 With Solidity Version Above 0.8	Language Version	Informational	Resolved
CON-05	Missing Emit Events	Coding Style	Informational	Partially Resolved
STA-01	Inconsistency Between Transfer And TransferFrom Logic	Inconsistency	Informational	Acknowledged

432-01 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	Major	contracts/Operations.sol (Base): 84, 96, 106, 115, 125, 140, 149, 164, 174, 181, 189; contracts/cngn.sol (Base): 54, 61, 100, 107, 206, 233, 241, 246; contracts/forwarder.sol (Base): 51~53, 111~114, 131~132, 135~136, 139~140, 144~145; tron-contract/contracts/cngn.sol (Base): 224, 232, 341, 347, 353, 428, 434, 439, 444	Acknowledged

Description

MinimalForwarder

In the contract `MinimalForwarder`, the role `onlyOwner` has authority over the following functions:

- `updateAdminOperationsAddress()`;
- `execute()`;
- `pause()`;
- `unpause()`;
- `authorizeBridge()`;
- `deauthorizeBridge()`;

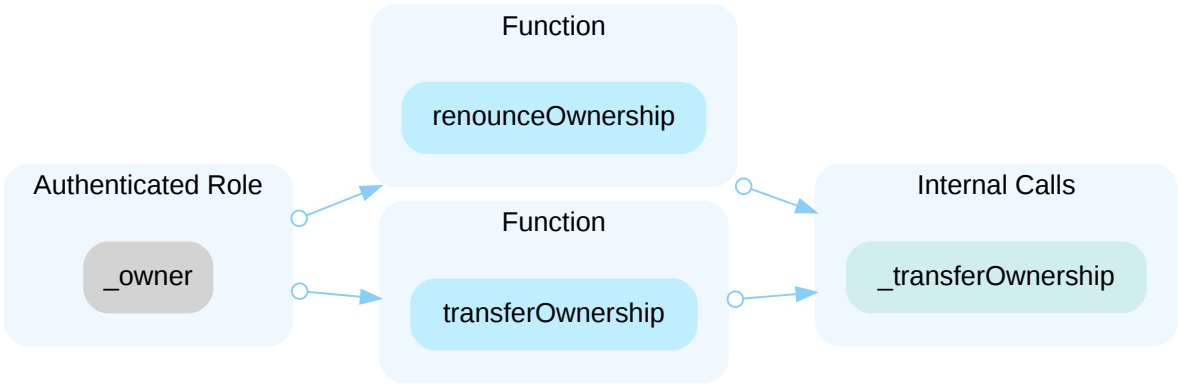
Any compromise to the `owner` account may allow a hacker to take advantage of this authority and change the admin contract, execute transactions, pause or unpause the contract, set or unset any address as a bridge.

In the contract `MinimalForwarder`, the role `onlyAuthorizedBridge` has authority over the function `executeByBridge()`.

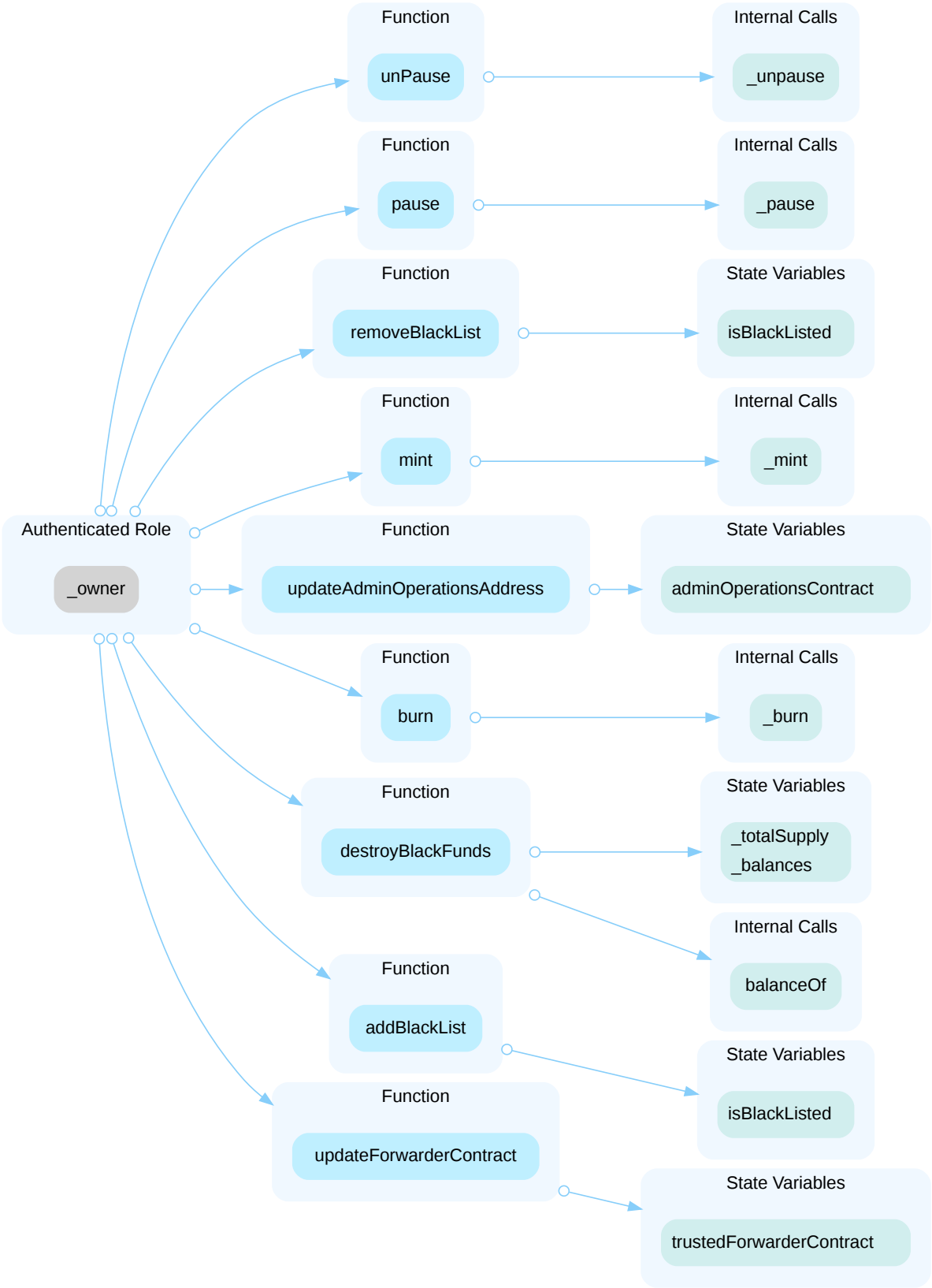
Any compromise to an `authorizedBridges` account may allow a hacker to take advantage of this authority and execute a transaction as a bridge.

tron-contract/cngn

In the contract `Ownable`, the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and renounce current ownership rights or transfer ownership to a new owner.



In the contract `cngn`, the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and pause or unpause the contract, remove or add a user to the blacklist, mint tokens to a specified address, burn a specified amount from the sender's balance, remove blacklisted user's funds.



contracts/cngn

In the contract `cngn`, the role `_owner` has authority over the following functions:

- `updateAdminOperationsAddress()`;

- `updateForwarderContract()`;
- `pause()`;
- `unpause()`;

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and change the admin contract address, change the forwarder contract address, pause or unpause the transfer.

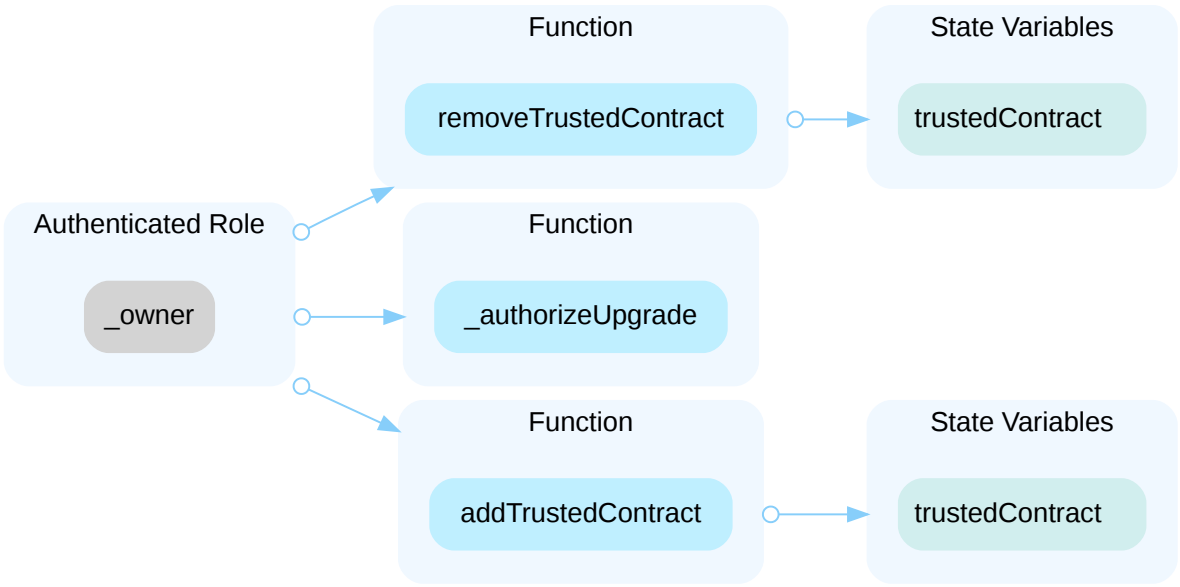
In the contract `cngn`, the role `onlyDeployerOrForwarder` has authority over the following functions:

- `mint()`;
- `burnByUser()`;

Any compromise to the `_owner` account or the `trustedForwarderContract` may allow the hacker to take advantage of this authority and mint or burn tokens.

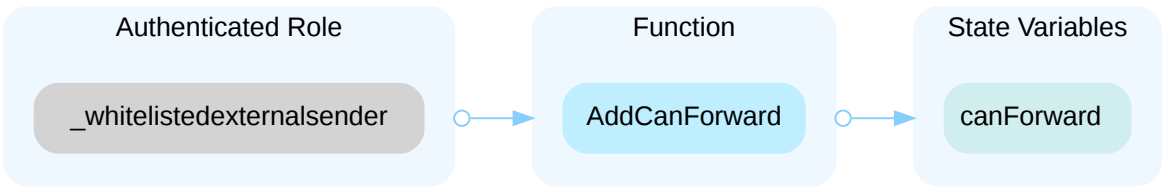
Admin

In the contract `Admin`, the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and set an address as trusted, set an address as untrusted, and upgrade the contract's implementation.

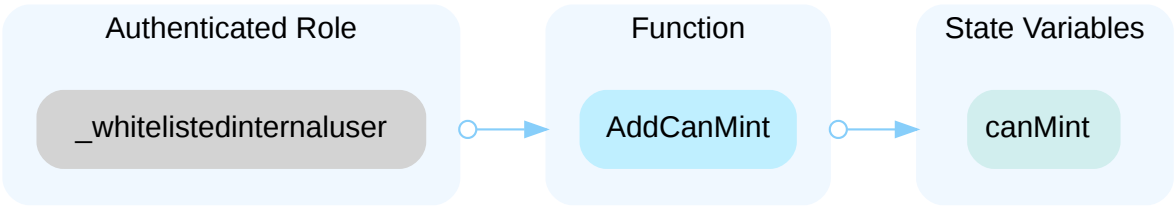


In the contract `Admin`, the role `_ownerortrustedcontract` has authority over the functions shown in the diagram below. Any compromise to a trusted address or the `_owner` account may allow the hacker to take advantage of this authority and whitelist an internal user, add a user as a forwarder, add a user to the minter whitelist, whitelist an external sender, remove minting permission from a user, blacklist an external sender, remove a user from forwarding permissions, and blacklist an internal user.

In the contract `Admin`, the role `_whitelistedexternalsender` has authority over the functions shown in the diagram below. Any compromise to the `_whitelistedexternalsender` account may allow the hacker to take advantage of this authority and add new users as forwarders.



In the contract `Admin`, the role `_whitelistedinternaluser` has authority over the functions shown in the diagram below. Any compromise to the `_whitelistedinternaluser` account may allow the hacker to take advantage of this authority and add new users as minters.



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.

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

`[WrappedCBDC, 2024/11/25]`: "This is well noted. We operate a stablecoin and minting keys are stored in a secured keyvault."

`[CertiK, 2024/12/06]`: In the new commit 9d8e59e32361b96f9f3c4ef38b4ee331eeeaaf3c:

MinimalForwarder

In the contract `MinimalForwarder`, the role `onlyOwner` has authority over the following functions:

- `updateAdminOperationsAddress()`;
- `execute()`;
- `pause()`;
- `unpause()`;
- `authorizeBridge()`;
- `deauthorizeBridge()`;

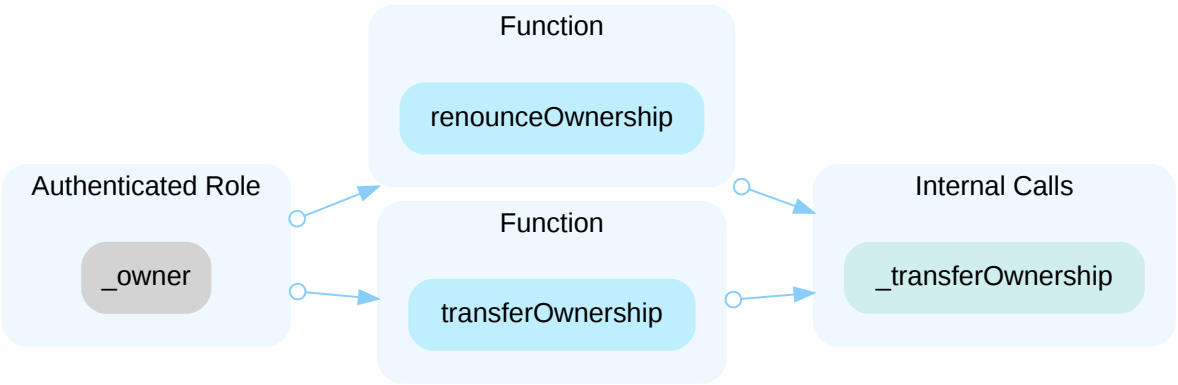
Any compromise to the `owner` account may allow a hacker to take advantage of this authority and change the admin contract, execute transactions, pause or unpause the contract, set or unset any address as a bridge.

In the contract `MinimalForwarder`, the role `onlyAuthorizedBridge` has authority over the function `executeByBridge()`.

Any compromise to an `authorizedBridges` account may allow a hacker to take advantage of this authority and execute a transaction as a bridge.

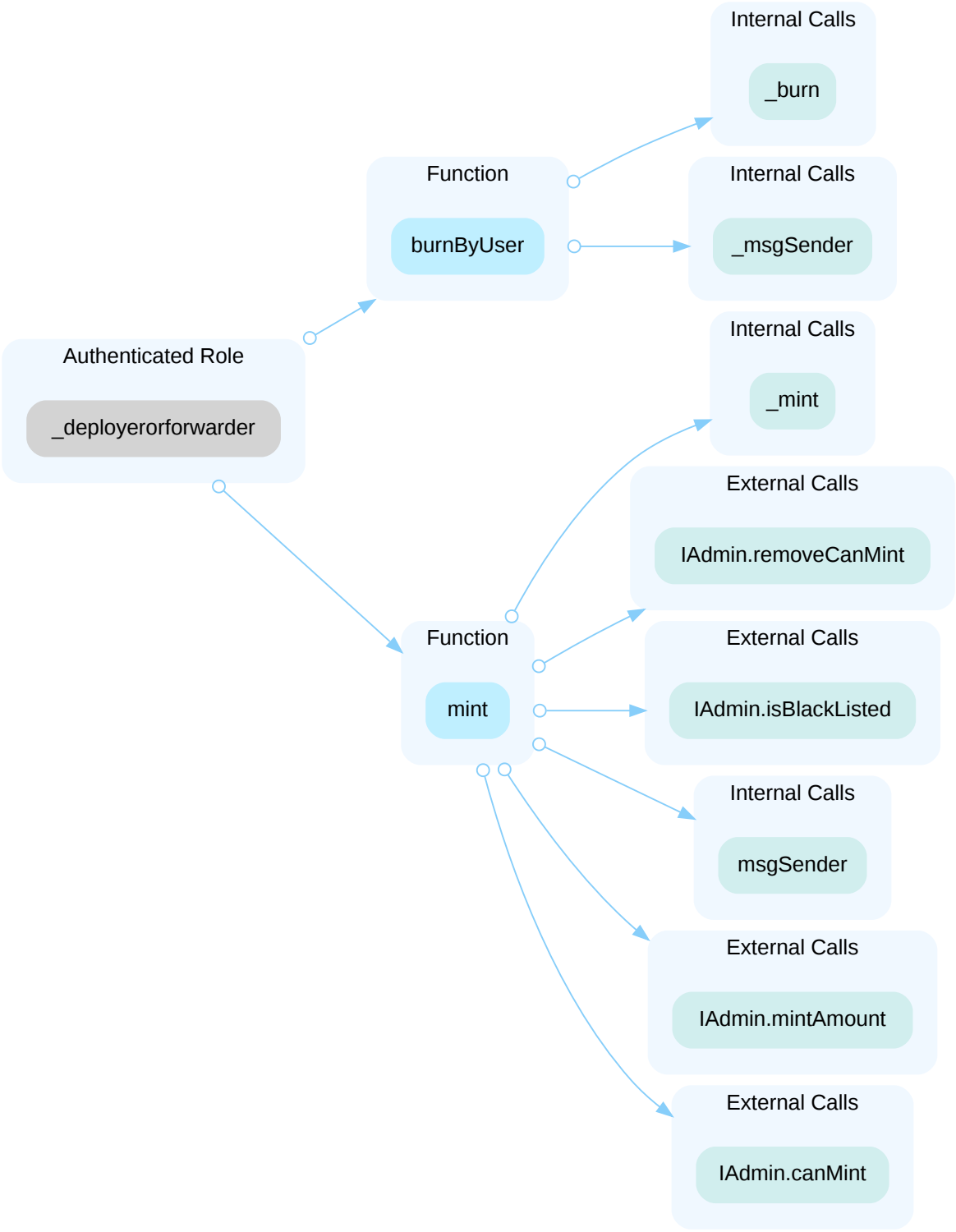
contracts/cngn

In the contract `Ownable`, the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and renounce or transfer the contract ownership.

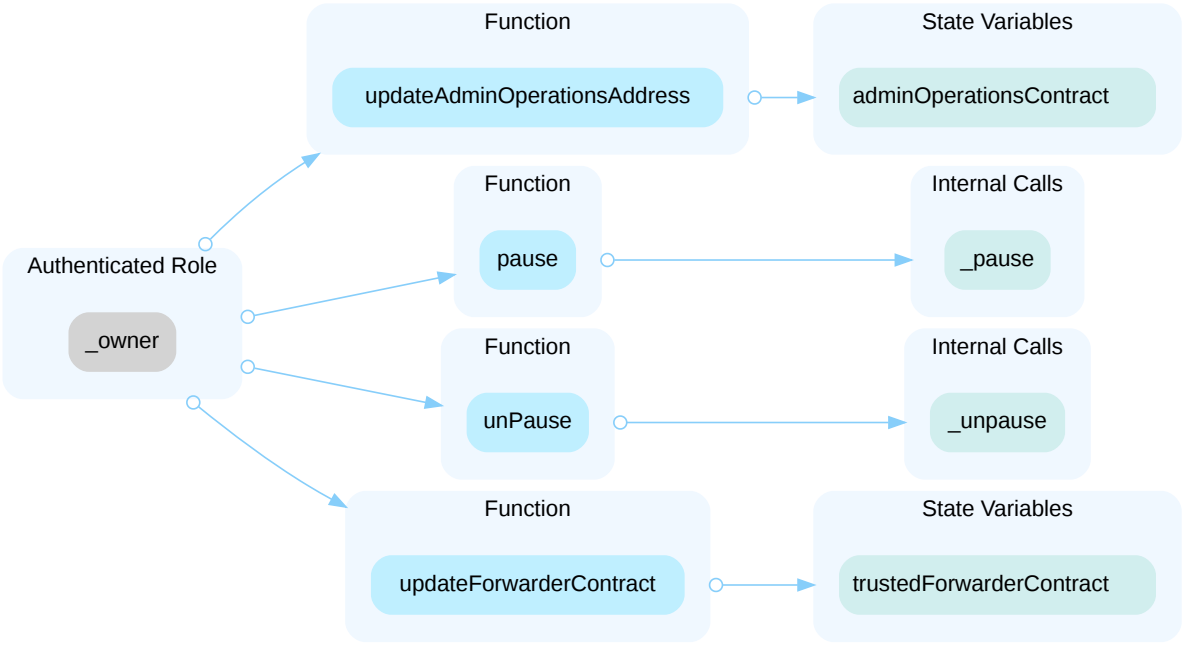


In the contract `cngn`, the role `_deployerorforwarder` has authority over the functions shown in the diagram below. Any compromise to the `_deployerorforwarder` account may allow the hacker to take advantage of this authority and burn

tokens by user, or mint tokens to a specified address.

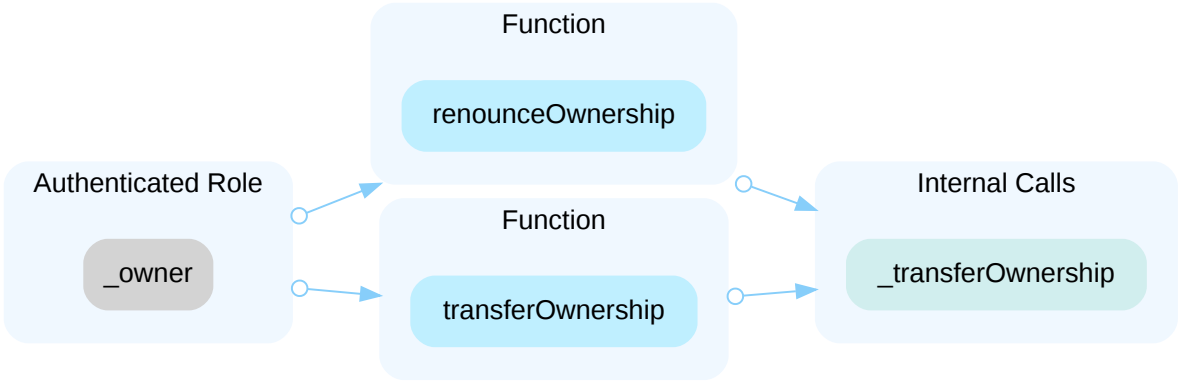


In the contract `cngn`, the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and update the admin operations address, pause contract operations, unpause the contract, and update the forwarder contract address.

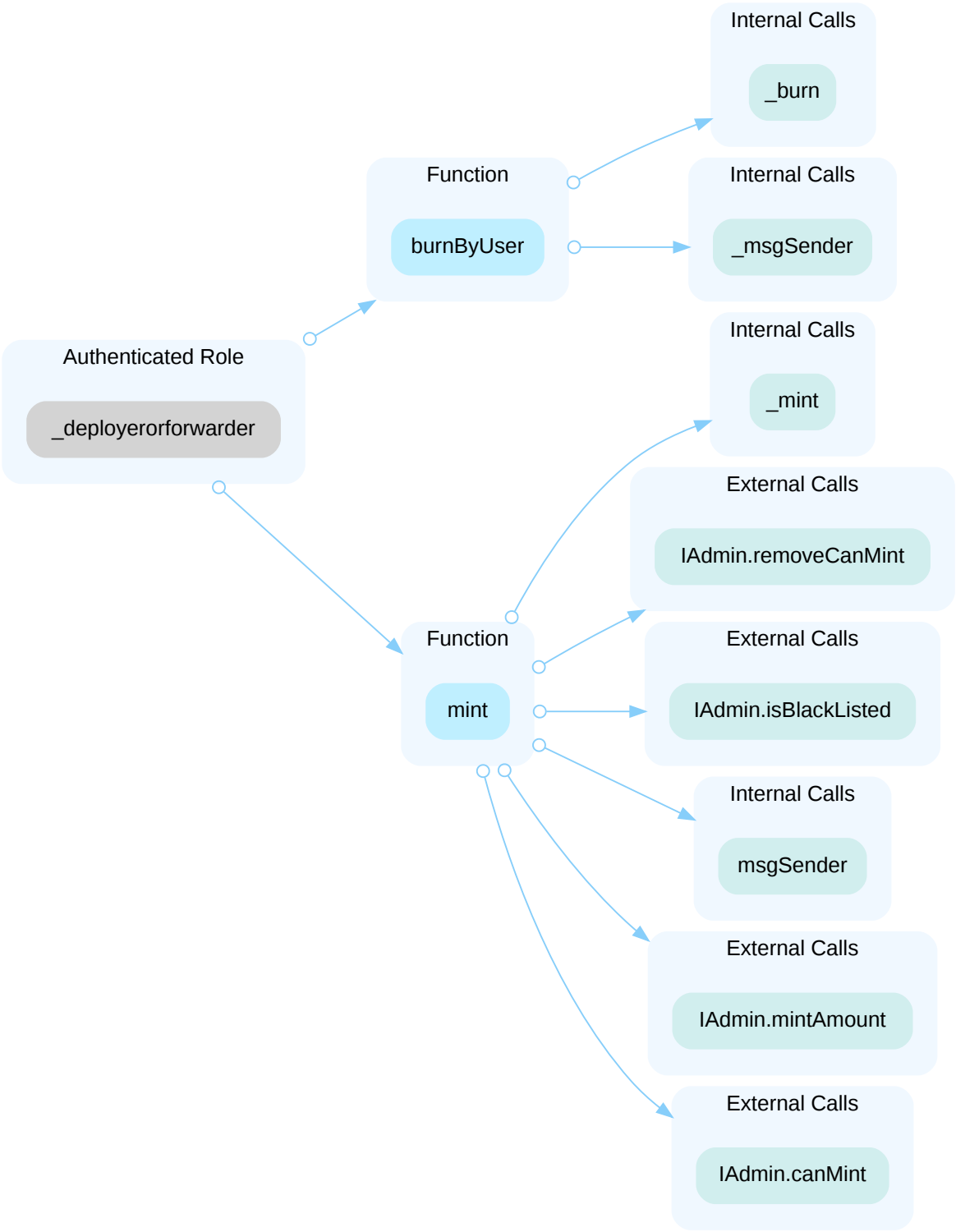


tron-contract/cngn

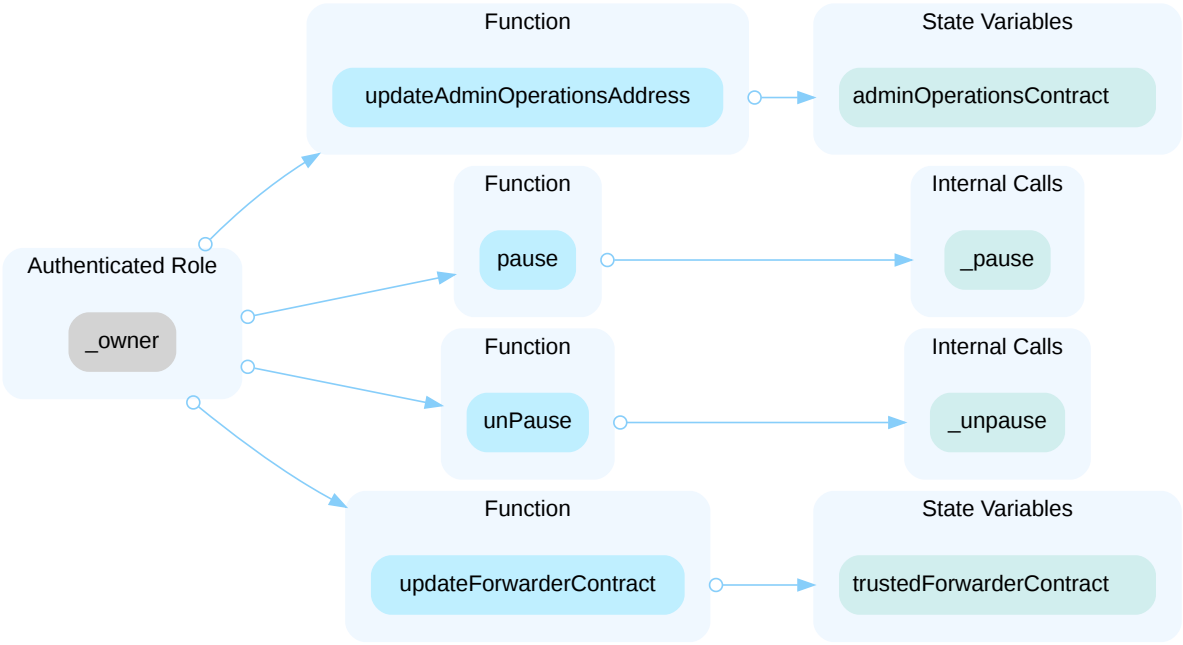
In the contract `Ownable` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and renounce or transfer the contract ownership.



In the contract `cngn` the role `_deployerorforwarder` has authority over the functions shown in the diagram below. Any compromise to the `_deployerorforwarder` account may allow the hacker to take advantage of this authority and burn tokens by user, or mint tokens to a specified address.

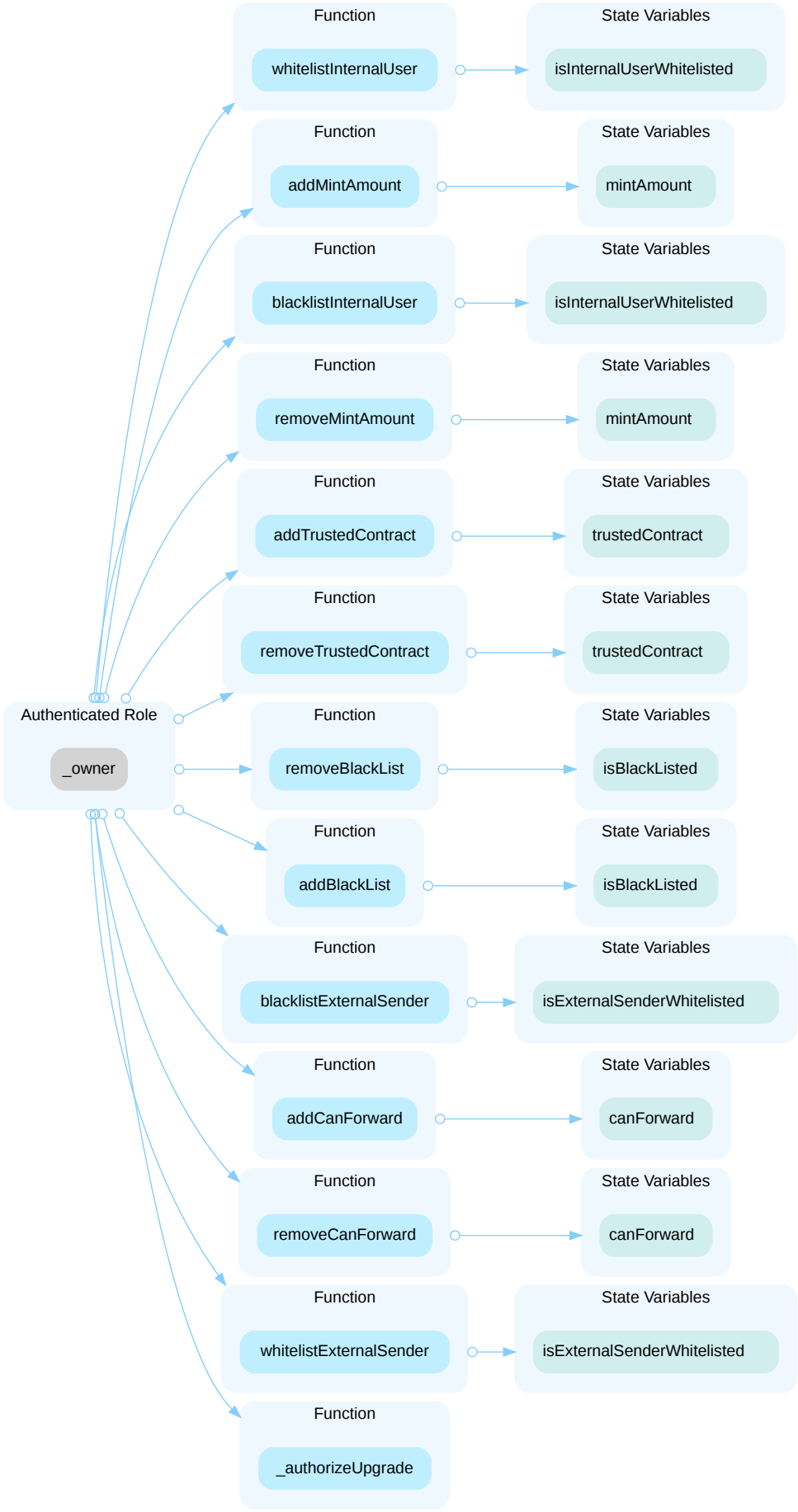


In the contract `cngn` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and update the admin operations address, pause contract operations, unpause the contract, and update the forwarder contract address.

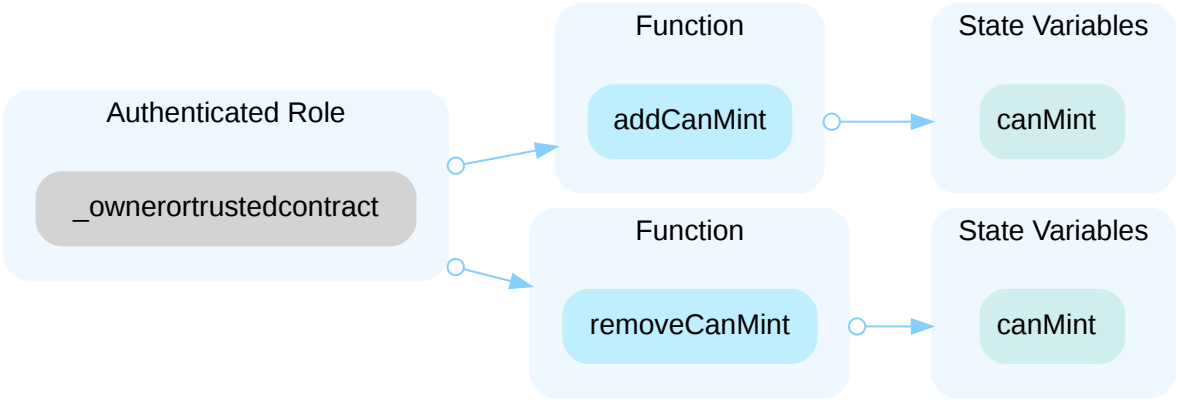


Admin

In the contract `Admin`, the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and whitelist or blacklist internal users, add or remove mint amounts for users, add or remove trusted contracts, whitelist or blacklist external senders, add or remove a user's forwarder status, and authorize contract upgrade implementations.


Function
whitelistInternalUser
State Variables
isInternalUserWhitelisted
Function
addMintAmount
State Variables
mintAmount
Function
blacklistInternalUser
State Variables
isInternalUserWhitelisted
Function
removeMintAmount
State Variables
mintAmount
Function
addTrustedContract
State Variables
trustedContract
Function
removeTrustedContract
State Variables
trustedContract
Authenticated Role
_owner
Function
removeBlackList
State Variables
isBlackListed
Function
addBlackList
State Variables
isBlackListed
Function
blacklistExternalSender
State Variables
isExternalSenderWhitelisted
Function
addCanForward
State Variables
canForward
Function
removeCanForward
State Variables
canForward
Function
whitelistExternalSender
State Variables
isExternalSenderWhitelisted
Function
_authorizeUpgrade

In the contract `Admin` , the role `_ownerortrustedcontract` has authority over the functions shown in the diagram below. Any compromise to the `_ownerortrustedcontract` account may allow the hacker to take advantage of this authority and add users to the can mint list or remove the minting ability from a user.



CON-02 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	**Major**	**contracts/Operations.sol (Base): 9; contracts/cngn.sol (Base): 14**	**Acknowledged**

Description

The contracts `cngn` and `Admin` are upgradable, therefore a centralized role has the authority to update the implementation contracts.

Any compromise to those centralized roles account may allow a hacker to take advantage of this authority and change the implementation contracts and therefore execute potential malicious functionality.

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a 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 migrate to a new implementation contract.

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 combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
  AND
- 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 deployed time-lock address.
- 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 combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

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

- Provide the deployed time-lock address.
- 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:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

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

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

`[WrappedCBDC, 2024/11/24]`: "The cNGN smart contract is closely coupled with our API service that allows for atomic execution of functions like minting of token. Our API service has a base protocol which is Bantu(a fork of stellar blockchain) and we are presently using the Multi-Sig features for minting and burning."

OPE-01 LACK OF BLACKLISTING AND MINTING AMOUNT SAFEGUARDS

Category	Severity	Location	Status
Logical Issue	Major	contracts/Operations.sol (Base): 18~19, 20~21	Resolved

Description

In the `Admin` contract, the following mappings are defined:

```
18      mapping(address => uint256) public mintAmount;
19  .....
20      mapping(address => bool) public isBlackListed;
```

However, there are no functions to utilize these mappings. As a result, it is not possible to blacklist a malicious address or set the amount of tokens a user can mint. Consequently, the `cngn` contract lacks important security features.

Recommendation

We recommend implementing those safety mechanisms.

Alleviation

`[CertiK, 2024/11/25]`: The client made changes resolving the finding in commit c11d7b9decef2b807921fb385bfc2682e3a5475a.

CNG-02 | UNUSED RETURN VALUE

Category	Severity	Location	Status
Volatile Code	Medium	contracts/cngn.sol (Base): 206~231	Resolved

Description

The smart contract does not check or store the return value of an external call in a local or state variable, which may introduce vulnerabilities due to the unhandled outcome.

```
229         IAdmin(adminOperationsContract).RemoveCanMint(signer);
```

Recommendation

It is suggested to ensure proper error handling by checking or using the return values of all external function calls, and storing them in appropriate local or state variables if necessary.

Alleviation

`[CertiK, 2024/11/25]`: The client made changes resolving the finding in commit 6126826c554c1a35a76c0692b29b499090e953f6.

CON-03 | `initialize()` IS UNPROTECTED

Category	Severity	Location	Status
Logical Issue	Medium	contracts/Operations.sol (Base): 43~44; contracts/cngn.sol (Base): 69~72	Resolved

Description

The `Admin` and `cngn` logic contracts do not protect the initializer. An attacker can front-run the `initialize` call and assume ownership of the logic contract. Once in control, the attacker can perform privileged operations, misleading users into believing that they are interacting with the legitimate owner of the upgradeable contract.

Recommendation

We recommend adding

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() initializer {}
```

The addition will prevent the function `initialize()` from being called directly in the implementation contract, but the proxy will still be able to `initialize()` its storage variables.

Alleviation

`[CertiK, 2024/11/25]`: The client made changes resolving the finding in commit 28ae61d7eca4b2546ce420e50144b63af20c06ff.

FOR-02 | FORWARDER IMPLEMENTATION

Category	Severity	Location	Status
Design Issue	Medium	contracts/forwarder.sol (Base): 1	Resolved

Description

The `MinimalForwarder` implementation seems to be a modified version of the `MinimalForwarder` from OpenZeppelin which is "mainly meant for testing, as it is missing features to be a good production-ready forwarder. This contract does not intend to have all the properties that are needed for a sound forwarding system".

Recommendation

We recommend implementing the ERC2771Forwarder to ensure the protocol adheres to the `ERC2771` standard and therefore allows secure meta transactions.

Alleviation

`[CertiK, 2024/12/12]`: The client made changes resolving the finding in commit 369572fc9e7111e26fe9e533e08e60b65003f90c.

FOR-04 | MISSING BLACKLIST CHECK

Category	Severity	Location	Status
Access Control	Medium	contracts/forwarder.sol (Base): 82~83	Resolved

Description

In the `MinimalForwarder` contract, the function `_executeTransaction()` does not verify that `req.from` is not blacklisted.

Recommendation

We recommend adding the following check:

```
        require(
            !IAdmin(adminOperationsContract).isBlackListed(req.from),
            "Blacklisted"
        );
```

Alleviation

`[CertiK, 2024/11/25]`: The client made changes resolving the finding in commit b73de618ab26214917cd3751eb119066b19311cb.

FOR-05 | UNUSED PAUSABLE FEATURES

Category	Severity	Location	Status
Volatile Code	Medium	contracts/forwarder.sol (Base): 8~9	Resolved

Description

In the `MinimalForwarder` contract, the `Pausable` contract is imported and inherited, and the `pause()` and `unpause()` functions are implemented. However The modifiers `whenPaused` and `whenNotPaused` are never used.

As a result, the `Pausable` feature cannot be used.

Recommendation

We recommend using the modifiers to make the contract effectively pausable.

Alleviation

`[CertiK, 2024/11/25]`: The client made changes resolving the finding in commit 25eef082445aabcc8536559ec7db3f62d927a580.

FOR-09 | EIP712 STANDARD NOT FOLLOWED

Category	Severity	Location	Status
Design Issue	Medium	contracts/forwarder.sol (Base): 6~7, 23, 38~42	Resolved

Description

The **EIP-712 standard** defines a structured approach to hashing and signing data to ensure integrity, uniqueness, and compatibility across applications. However, the current implementation does not fully adhere to this standard due to the following issues:

1. **Typehash Not Used**:
   The `_TYPEHASH`, is defined as:

   ```
   bytes32 public constant _TYPEHASH =
       keccak256("ForwardRequest(address from,address to,uint256 value,uint256 nonce,bytes data)");
   ```

   is present in the `MinimalForwarder` contract but is never used to validate signatures. This undermines the structured hashing and signing process specified by EIP-712.

2. **ChainID Validation Missing**:
   The current implementation does not validate the `chainId`, an essential part of EIP-712. Without this, the mechanism lacks the ability to distinguish between different blockchain environments, which is a core requirement of the standard.

3. **Unused EIP-712 Library**:
   The `EIP712` library, though imported and inherited, is not utilized. This library offers built-in functionalities to simplify the creation of compliant hashing and signing schemes, which are not leveraged in the current implementation.

These deviations from the standard highlight incomplete or improper adoption of EIP-712, reducing the protocol's adherence to industry best practices for data signing.

Recommendation

We recommend fully adhering to the EIP-712 standard:

1. **Incorporate `_TYPEHASH` in Signature Validation**:
   Use the `_TYPEHASH` as part of the structured data hashing process to follow the standard's requirements.

2. **Add ChainID Validation**:
   Include the `chainId` in the domain separator or explicitly verify it in the signature validation process to ensure compliance.

3. **Leverage EIP-712 Library Functions**:
   Utilize the `EIP712` library to generate the domain separator, hash structured data, and validate signatures as specified by the standard.

Implementing these changes will ensure that the contract follows the EIP-712 standard, providing robust and standardized data signing and validation mechanisms.

Alleviation

`[WrappedCBDC, 2024/11/24]`: "A function to prevent replay attack is already included and only allowed minted amount can be sent to the forwarder".

`[CertiK, 2024/11/25]`: The **typehash** has been removed from the implementation instead of being used in the signature scheme. The typehash is a critical component of **EIP-712**, ensuring signed data is uniquely tied to its intended purpose and preventing misuse or replay in other contexts.

Removing the typehash weakens the security of the mechanism. We recommend reintroducing it to align with EIP-712 and enhance protection.

`[CertiK, 2024/12/12]`: The client made changes resolving the finding in commit f7df9c1e04cb0cde8facb0f7e6f0262d6c2eed92.

CNN-01 PULL-OVER-PUSH PATTERN IN `transferOwnership()` FUNCTION

Category	Severity	Location	Status
Logical Issue	Minor	tron-contract/contracts/cngn.sol (Base): 240~245	Acknowledged

Description

The change of `_owner` by function `transferOwnership()` overrides the previously set `_owner` with the new one without guaranteeing the new `_owner` is able to actuate transactions on-chain.

Recommendation

We advise the pull-over-push pattern to be applied here whereby a new `owner` is first proposed and consequently needs to accept the `_owner` status ensuring that the account can actuate transactions on-chain. The following code snippet can be taken as a reference:

```
address public potentialOwner;

function transferOwnership(address pendingOwner) external onlyOwner {
    require(pendingOwner != address(0), "potential owner can not be the zero
address.")
    potentialOwner = pendingOwner;
    emit OwnerNominated(pendingOwner);
}

function acceptOwnership() external {
    require(msg.sender == potentialOwner, 'You must be nominated as potential owner
before you can accept ownership');
    emit OwnerChanged(_owner, potentialOwner);
    _owner = potentialOwner;
    potentialOwner = address(0);
}
```

Alleviation

`[CertiK, 2024/11/25]`: The client acknowledges the finding and opts to make no change at this time.

CNN-02 | MINTING AND BURNING NOT PAUSABLE

Category	Severity	Location	Status
Volatile Code	Minor	tron-contract/contracts/cngn.sol (Base): 428~429, 434~435	Resolved

Description

In the contract `cngn`, the transfer functions are pausable, however the functions `mint()` and `burn()` cannot be paused.

Recommendation

We recommend extending the pausable feature to the tokens minting and burning.

Alleviation

`[CertiK, 2024/12/03]`: The client made changes resolving the issue in commit 4cf0d529d0314be1c4b5ed3a47e270ae880a9c0e.

CON-04 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/cngn.sol (Base): 70, 71, 79, 80, 101, 103, 108, 110; contracts/forwarder.sol (Base): 46, 48, 52, 54	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.

```
110          trustedForwarderContract = _newForwarderContract;
```

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

```
48          adminOperationsContract = _adminOperationsContract;
```

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

```
54          adminOperationsContract = _newAdmin;
```

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

```
103          adminOperationsContract = _newAdmin;
```

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

```
79          trustedForwarderContract = _trustedForwarderContract;
```

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

```
80          adminOperationsContract = _adminOperationsContract;
```

- `_adminOperationsContract` 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.

Alleviation

`[CertiK, 2024/11/25]`: The client acknowledges the finding and opts to make no change at this time.

FOR-07 | `draft-EIP712` IS DEPRECATED

Category	Severity	Location	Status
Language Version	Minor	contracts/forwarder.sol (Base): 6~7	Resolved

Description

The contract `MinimalForwarder` imports the Open Zeppelin library `draft-EIP712.sol` which has been deprecated since the release of version 4.8.0.

Recommendation

We recommend updating the import as follows:

```
import "@openzeppelin/contracts/utils/cryptography/EIP712.sol";
```

Alleviation

`[CertiK, 2024/11/25]`: The client made changes resolving the finding in commit 4b6d8f051cfd09860203c51a42757346c5db16fb.

OPE-02 PRIVILEGED ROLES NOT REMOVABLE FROM BLACKLISTED ADDRESSES

Category	Severity	Location	Status
Inconsistency	Minor	contracts/Operations.sol (Base): 142, 166	Resolved

Description

In the contract `Admin`, both functions `RemoveCanMint()` and `RemoveCanForward()` require the address to not be blacklisted. However, removing minting and forwarder roles from a blacklisted address should not be prevented.

Recommendation

We recommend enforcing a blacklisted address to not have privileged roles, moreover, the functions allowing to remove privileges should not be restricted to only whitelisted addresses.

Alleviation

`[CertiK, 2024/11/25]`: The client made changes resolving the finding in commit 8da6b9291e0895fa6af97124ae411e8aeeb235b6.

432-02 | EVENT NOT INDEXED

Category	Severity	Location	Status
Design Issue	Informational	contracts/cngn.sol (Base): 34, 39; tron-contract/contracts/cngn.sol (Base): 326	Resolved

Description

If an event is not indexed in a smart contract, it means that the event's parameters are not tagged with the `indexed` keyword. This has implications for how the event data can be searched and filtered when looking through blockchain logs.

Without indexing, the event will still emit the data as part of the transaction log, but users won't be able to query for these events using the parameters. They'll have to retrieve the entire set of logs and manually sift through them to find events with the specific data. This can be less efficient and more time-consuming, especially on a blockchain with a high volume of transactions and events.

Recommendation

To mitigate this issue, it is recommended to index the most relevant parameters in the event to be defined.

Alleviation

`[CertiK, 2024/11/25]`: The client made changes partially resolving the finding in commit 4abe7a4e33550be015e7cb782fe4facfe2d31225.
Events in `tron-contract/contracts/cngn.sol` are still not indexed.

`[CertiK, 2024/12/03]`: The client made changes resolving the finding in commit fd819048fb07c2b8baada53f17fdf8b81455d793.

432-03 | MISSING ERROR MESSAGES

Category	Severity	Location	Status
Coding Style	Informational	contracts/cngn.sol (Base): 141, 142, 170, 171, 172; tron-contract/contracts/cngn.sol (Base): 354, 383, 384, 401, 402, 403, 429	Acknowledged

Description

The **require** can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise adding error messages to the linked **require** statements.

Alleviation

`[CertiK, 2024/12/03]`: The client acknowledges the finding and opts to make no change at this time.

CNN-03 EXPERIMENTAL ABIENCODERV2 WITH SOLIDITY VERSION ABOVE 0.8

Category	Severity	Location	Status
Language Version	Informational	tron-contract/contracts/cngn.sol (Base): 4~5	Resolved

Description

The contract `cngn` utilizes the experimental `ABIEncoderV2` feature, even though the Solidity compiler version specified is above 0.8. Solidity versions 0.8 and above natively include the functionalities provided by `ABIEncoderV2`, rendering the experimental directive unnecessary.

Recommendation

We recommend removing the `pragma experimental ABIEncoderV2` to maintain cleaner code and reduce any ambiguity about the use of experimental features.

Alleviation

`[CertiK, 2024/11/25]`: The client made changes resolving the finding in commit 736c055158f2c5f5d46b106f74ec3ebbf385c0dc.

CON-05 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	contracts/Operations.sol (Base): 174, 181; contracts/cngn.sol (Base): 100, 107	Partially Resolved

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

`[CertiK, 2024/11/25]`: The client made changes partially resolving the finding in commit 4abe7a4e33550be015e7cb782fe4facfe2d31225.
The sensitive functions in `contracts/cngn.sol` are still not emitting events.

STA-01 INCONSISTENCY BETWEEN TRANSFER AND TRANSFERFROM LOGIC

Category	Severity	Location	Status
Inconsistency	Informational	contracts/cngn.sol (Update1): 140; contracts/cngn.sol (Update2): 171; tron-contract/contracts/cngn.sol (Update2): 505	Acknowledged

Description

The `transfer()` function contains different restrictions or additional logic compared to the `transferFrom()` function, resulting in inconsistent behavior between these two core ERC-20 functions. This inconsistency can lead to a situation where a transfer might fail when using `transfer()` due to the imposed restrictions, but the same transfer could succeed using `transferFrom()`, potentially bypassing the intended security checks or business logic.

Recommendation

To ensure consistent behavior between `transfer()` and `transferFrom()`, apply the following steps:

1. **Harmonize Function Logic**: Review and align the restrictions and checks in both `transfer()` and `transferFrom()` functions so that they enforce the same rules.
2. **Reevaluate Business Logic**: Assess the necessity of the restrictions in the `transfer()` function. If they are essential for the token's integrity and security, ensure that the same logic is implemented in the `transferFrom()` function.
3. **Thorough Testing**: Test both functions extensively to ensure that they behave consistently under various scenarios, including edge cases.

Alleviation

`[WrappedCBDC, 2024/11/26]`: "The transfer function is serving two purposes. The If conditional block is for our swap feature when internalwhitelisted users want to move their token back to our platform, they transfer to an address provided by the platform and we equally burn the token out of circulation. The else logic is for normal users that are not within the platform to be able to transfer or transact with the token."

OPTIMIZATIONS | CNGN STABLECOIN

ID	Title	Category	Severity	Status
CNG-01	Unused Mappings	Volatile Code, Code Optimization	Optimization	Resolved
CON-01	Unused Event	Code Optimization	Optimization	Resolved
FOR-01	Unused Modifier	Access Control, Code Optimization	Optimization	Resolved

CNG-01 | UNUSED MAPPINGS

Category	Severity	Location	Status
Volatile Code, Code Optimization	Optimization	contracts/cngn.sol (Base): 24~26	Resolved

Description

In the contract `cngn`, the following mappings:

```
24      mapping(address => bool) private isBlackListed;
25      mapping(address => bool) private isWhiteListed;
26      mapping(address => mapping(address => uint256)) private isMintAllow;
```

are defined but never used.

Recommendation

We recommend verifying if those mappings are necessary to implement safety features and to use or remove them

Alleviation

`[CertiK, 2024/11/25]`: The client made changes resolving the finding in commit ee0eacc68c9b4a2f3dd5cbf51cd4d203c9647db2.

CON-01 | UNUSED EVENT

Category	Severity	Location	Status
Code Optimization	Optimization	contracts/Operations.sol (Base): 27, 28, 29, 30, 35, 36; contracts/cngn.sol (Base): 34, 35, 36, 37, 38, 39	Resolved

Description

The linked events are never emitted, which can lead to confusion and code maintainability issues.

Recommendation

It is recommended to remove the unused events or emit them in the intended functions to improve code clarity and maintainability.

Alleviation

`[CertiK, 2024/11/25]`: The client made changes resolving the finding. Commit 7e318f9676596236761ea11f88609aba7c3b70a0.

FOR-01 | UNUSED MODIFIER

Category	Severity	Location	Status
Access Control, Code Optimization	Optimization	contracts/forwarder.sol (Base): 159~162	Resolved

Description

In the `forwarder` contract, the modifier `onlyAdmin` is defined but never used.

Recommendation

We recommend using the modifier or removing it if unnecessary.

Alleviation

`[CertiK, 2024/11/25]`: The client made changes resolving the finding in commit 7e318f9676596236761ea11f88609aba7c3b70a0.

FORMAL VERIFICATION | CNGN STABLECOIN

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 Pausable ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the pausable 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-transferfrom-revert-zero-argument	`transferFrom` Fails for Transfers with Zero Address Arguments
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Transfers
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Transfers

Property Name	Title
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-approve-false	If `approve` Returns `false`, the Contract's State Is Unchanged
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-approve-succeed-normal	`approve` Succeeds for Valid Inputs
erc20-approve-never-return-false	`approve` Never Returns `false`
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address
erc20pausable-transfer-revert-paused	`transfer` Fails for a Paused Contract
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20pausable-transferfrom-revert-paused	`transferFrom` Fails for a Paused Contract
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-transfer-false	If `transfer` Returns `false`, the Contract State Is Not Changed
erc20-transfer-never-return-false	`transfer` Never Returns `false`
erc20-transferfrom-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address

Verification of Standard Ownable Properties

We verified *partial* properties of the public interfaces of those token contracts that implement the Ownable interface. This involves:

- function `owner` that returns the current owner,
- functions `renounceOwnership` that removes ownership,

- function `transferOwnership` that transfers the ownership to a new owner.

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

Property Name	Title
ownable-transferownership-correct	Ownership is Transferred
ownable-owner-succeed-normal	`owner` Always Succeeds
ownable-renounceownership-correct	Ownership is Removed
ownable-renounce-ownership-is-permanent	Once Renounced, Ownership Cannot be Regained

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 MinimalForwarder (contracts/forwarder.sol) In Commit 432c992cf5a1db6f843630a805454ad0092c8292

Verification of Standard Ownable Properties

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	True	

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	True	
ownable-renounce-ownership-is-permanent	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 cngn (contracts/cngn.sol) In Commit 432c992cf5a1db6f843630a805454ad0092c8292

Verification of Pausable ERC-20 Compliance

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-zero-argument	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-fail-exceed-allowance	Inapplicable	The property does not apply to the contract
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-correct-allowance	Inapplicable	The property does not apply to the contract
erc20pausable-transferfrom-revert-paused	True	
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-false	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 `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 `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 `transfer`

Property Name	Final Result	Remarks
erc20-transfer-exceed-balance	Inapplicable	The property does not apply to the contract
erc20-transfer-correct-amount	Inapplicable	The property does not apply to the contract
erc20pausable-transfer-revert-paused	True	
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	
erc20-transfer-revert-zero	True	

Detailed Results for Function `approve`

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

Detailed Results For Contract Admin (contracts/Operations.sol) In Commit 432c992cf5a1db6f843630a805454ad0092c8292

Verification of Standard Ownable Properties

Detailed Results for Function `owner`

Property Name	Final Result	Remarks
ownable-owner-succeed-normal	True	

Detailed Results for Function `renounceOwnership`

Property Name	Final Result	Remarks
ownable-renounceownership-correct	True	
ownable-renounce-ownership-is-permanent	Inapplicable	The property does not apply to the contract

Detailed Results for Function `transferOwnership`

Property Name	Final Result	Remarks
ownable-transferownership-correct	True	

Detailed Results For Contract cngn (tron-contract/contracts/cngn.sol) In Commit 432c992cf5a1db6f843630a805454ad0092c8292

Verification of Pausable ERC-20 Compliance

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-false	True	
erc20-transferfrom-never-return-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	
erc20pausable-transferfrom-revert-paused	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-revert-zero	True	
erc20-approve-correct-amount	True	
erc20-approve-never-return-false	True	
erc20-approve-false	True	
erc20-approve-succeed-normal	True	

Detailed Results for Function `totalSupply`

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

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	True	
erc20-transfer-never-return-false	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-correct-amount	True	
erc20pausable-transfer-revert-paused	True	
erc20-transfer-false	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	

APPENDIX | CNGN STABLECOIN

Finding Categories

Categories	Description
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.
Language Version	Language Version findings indicate that the code uses certain compiler versions or language features with known security issues.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
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-Pausable Properties

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;
```

erc20pausable-transferfrom-revert-paused

Any call of the form `transferFrom(from, dest, amount)` must fail for a paused contract.

Specification:

```
reverts_when paused();
```

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 cngn 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 cngn.

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 `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 `cngn` 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;
```

erc20pausable-transfer-revert-paused

Any invocation of `transfer(recipient, amount)` must fail if the contract is paused.

Specification:

```
reverts_when paused();
```

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;
```

Description of the Analyzed Ownable Properties

Properties related to function `transferOwnership`

ownable-transferownership-correct

Invocations of `transferOwnership(newOwner)` must transfer the ownership to the `newOwner`.

Specification:

```
ensures this.owner() == newOwner;
```

Properties related to function `owner`

ownable-owner-succeed-normal

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

Specification:

```
reverts_only_when false;
```

Properties related to function `renounceOwnership`

ownable-renounce-ownership-is-permanent

The contract must prohibit regaining of ownership once it has been renounced.

Specification:

```
constraint \old(owner()) == address(0) ==> owner() == address(0);
```

ownable-renounceownership-correct

Invocations of `renounceOwnership()` must set ownership to address(0).

Specification:

```
ensures this.owner() == address(0);
```

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.