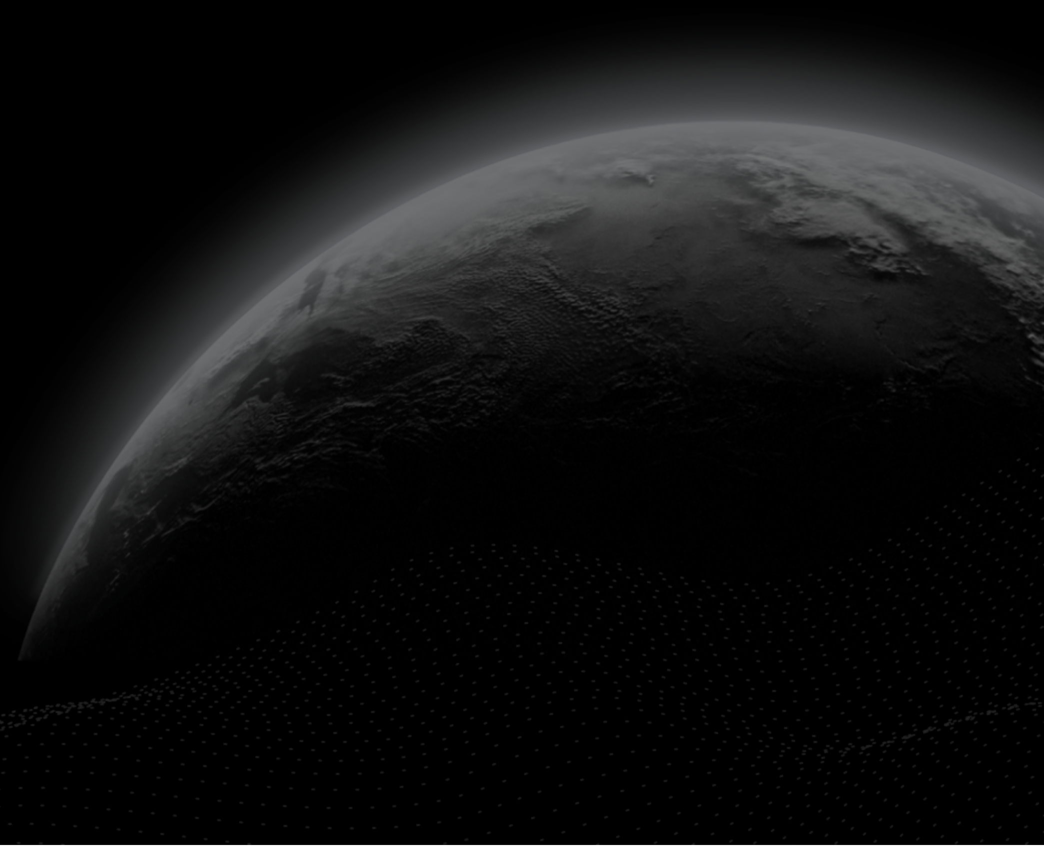
Security Assessment

Stablecoin Studio

CertiK Assessed on Aug 28th, 2023



CertiK Assessed on Aug 28th, 2023

Stablecoin Studio

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

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	EVM Compatible	Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 08/28/2023	N/A

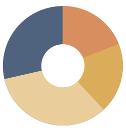
CODEBASE
https://github.com/hashgraph/hedera-accelerator-stablecoin/tree/main/contracts/contracts
View All in Codebase Page

COMMITS
573b9861882437018fe7974df4a019675db31c31
View All in Codebase Page

Highlighted Centralization Risks

- Initial owner token share is 100%
- Contract upgradeability
- Transfers can be paused
- Privileged role can mint tokens

Vulnerability Summary



21	7	0	1	13	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.
4	Major	1 Resolved, 3 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.
4	Medium	2 Resolved, 2 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
7	Minor	3 Resolved, 4 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.

6	Informational	1 Resolved, 1 Partially Resolved, 4 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 | STABLECOIN STUDIO

CODEBASE | STABLECOIN STUDIO

Repository

https://github.com/hashgraph/hedera-accelerator-stablecoin/tree/main/contracts/contracts

Commit

573b9861882437018fe7974df4a019675db31c31

AUDIT SCOPE | STABLECOIN STUDIO

35 files audited ● 22 files with Acknowledged findings ● 13 files without findings

ID	Repo	File	SHA256 Checksum
IWI	CertiKProject/certik-audit-projects	contracts/extensions/Interfaces/IWipeable.sol	b0f584e762461aa23b89acf06f579d4e3d9ce94ab93c3807f6ee769467b459d5
BCK	CertiKProject/certik-audit-projects	contracts/extensions/Burnable.sol	757b7392bf4ef6db7e9bee685bd254687bc6a822153db11826aa96cd9c1e1175
CIC	CertiKProject/certik-audit-projects	contracts/extensions/CashIn.sol	0b7661d83e515bf5bc88aa95b584e05575431ea8749b67368cc709bae9713464
DCK	CertiKProject/certik-audit-projects	contracts/extensions/Deletable.sol	259f8a528d91123a48ff48ef042e851d004a992fdcc37fdb5f6eff323ed859d7
FCK	CertiKProject/certik-audit-projects	contracts/extensions/Freezable.sol	17bcf2519f78be9ded36b9e15ed413efa3f5abb9f8cc40f0bef5eecd440fb71a
KYC	CertiKProject/certik-audit-projects	contracts/extensions/KYC.sol	42b6455996ce31c46e0d73aa256085470dff9fead668b992d64fb4fce8f4a4b2
PCK	CertiKProject/certik-audit-projects	contracts/extensions/Pausable.sol	ba9170c8bbe7906420eb1a977feda0217a376f516ae68d0b3bf5ab564152e46e
RCK	CertiKProject/certik-audit-projects	contracts/extensions/Rescatable.sol	bf13d0a22f7ecce93eeb2590ad1ecc5dfae5e34f7c2a59ae11a9dd299db83c65
RCP	CertiKProject/certik-audit-projects	contracts/extensions/Reserve.sol	470c957320e558a0f9bccf9bb9cb070fa2850a425b55b59d748e0d252a5dc9d0
RMC	CertiKProject/certik-audit-projects	contracts/extensions/RoleManagement.sol	7ff8d42f7bd4f1ee965dc10b10e911c8eadc2e8c4b49cedaf5311fd19a004e0c
RKP	CertiKProject/certik-audit-projects	contracts/extensions/Roles.sol	4c3879ddc89d9925dfa88ddd0cb43a20d3454062093d3ee0c537d4b53df6439b
SAC	CertiKProject/certik-audit-projects	contracts/extensions/SupplierAdmin.sol	c20355974b7eab338991593ceb88c195a53c712eb458a6eacd01213ab05be83d
TOC	CertiKProject/certik-audit-projects	contracts/extensions/TokenOwner.sol	c08cfdc1c2ef91a572b0ed52a574842d0c2853a75df47e53aa29134c7f397368

ID	Repo	File	SHA256 Checksum
WCK	CertiKProject/certik-audit-projects	contracts/extensions/Wipeable.sol	d5ae02a8418a01f5bfa34e2ff640f3951c3bff1b24a285d89754c21f1e30034e
IHT	CertiKProject/certik-audit-projects	contracts/Interfaces/IHederaTokenManager.sol	fe26b2be5a27cb70c8f1299486848a6a97b618a334ceda7cf3752f04ccdc3335
ISC	CertiKProject/certik-audit-projects	contracts/Interfaces/IStableCoinFactory.sol	b006283df0807c36ad97197f8cbcd59f52bc26b1012cf09a052899bbb55f319b
HRK	CertiKProject/certik-audit-projects	contracts/hts-precompile/HederaResponseCodes.sol	8b9e4ab063b4ccb2d26b3965f25ab087b7deef9844486c566d114e0607921020
IHS	CertiKProject/certik-audit-projects	contracts/hts-precompile/IHederaTokenService.sol	7844928f9c85d5958ba4f7604d6ae88aa4e40da9057a1b1ac281463c30b16a9d
KLC	CertiKProject/certik-audit-projects	contracts/library/KeysLib.sol	c7261f18a295946e64ae1e32e4fd08deea008a3c0d7ee82a707a736eb4c80e4c
HRC	CertiKProject/certik-audit-projects	contracts/HederaReserve.sol	6fb82a45f92c3dbce0d1cdee3112a2c6c93030a06f93eb1807d084800ec7b144
HTM	CertiKProject/certik-audit-projects	contracts/HederaTokenManager.sol	602082c768d41b05455c5d0adc824c94a13a9e90ff03081140ed479f34a59359
SCF	CertiKProject/certik-audit-projects	contracts/StableCoinFactory.sol	20c9fce832596ff78a71416cfe3a9fbce03464a3be70c99184e39fc8e6a16280
IBI	CertiKProject/certik-audit-projects	contracts/extensions/Interfaces/IBurnable.sol	8abcd2329e2923bc898798aefa7bd216b34b31c16488d7a85202209faf12a6c6
ICI	CertiKProject/certik-audit-projects	contracts/extensions/Interfaces/ICashIn.sol	f09646b882fa4797c61cb25d2e8a98d0be1d085a550785ce4f2b5c24f20416f5
IDI	CertiKProject/certik-audit-projects	contracts/extensions/Interfaces/IDeletable.sol	42e71b48725543321ea27829024622a26d2df0e92ae9ae78a9ca81d4381e62b5
IFI	CertiKProject/certik-audit-projects	contracts/extensions/Interfaces/IFreezable.sol	d9dd56ef77afdfff2b3ab4d311ec90ab8f7d458db0f3a57a2f7bc167baf6f368
IKY	CertiKProject/certik-audit-projects	contracts/extensions/Interfaces/IKYC.sol	af1f5b54a78793ee7d741031c0832db02a938a807489934ba1b6088d170b1af3
IPI	CertiKProject/certik-audit-projects	contracts/extensions/Interfaces/IPausable.sol	602905462b294fc2a763494c0aaffb26f3babad7063018aaa79d1acf2b5b81

ID	Repo	File	SHA256 Checksum
IRI	CertiKProject/certik-audit-projects	contracts/extensions/Interfaces/IRescatable.sol	4882f54b27ff9fe13443ee93c8c5ba3e1b254d2e16af8308d4eadc7572632c97
IRC	CertiKProject/certik-audit-projects	contracts/extensions/Interfaces/IReserve.sol	d819bb76eb36809719cdf930820db7c97b7c97a109b5e30e04b904af5ee96444
IRM	CertiKProject/certik-audit-projects	contracts/extensions/Interfaces/IRoleManagement.sol	aa7777d2df5c8da16d38256c4ea08c63a27a28785c689f6d2a93575a63c25a83
IRK	CertiKProject/certik-audit-projects	contracts/extensions/Interfaces/IRoles.sol	4bea2b8051bab03e348a5cf6240d719526c9e2e874625983fc66d14cb1ecea80
ISA	CertiKProject/certik-audit-projects	contracts/extensions/Interfaces/ISupplierAdmin.sol	b67ed92e550ca874100dfe2cb3325e22fdd9c67df2d4e6aec150daee743613d2
ITO	CertiKProject/certik-audit-projects	contracts/extensions/Interfaces/ITokenOwner.sol	2e87fa764c7d33db5c1c7425edc7bf4531fec9de3fa081f4a05a10d28855b3f2
IHR	CertiKProject/certik-audit-projects	contracts/Interfaces/IHederaReserve.sol	f783798435f10cb26282d95c7b5a3c3dd2773aea1a234c0526a90101c005af6f

APPROACH & METHODS | STABLECOIN STUDIO

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

The auditing process pays special attention to the following considerations:

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

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

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

REVIEW NOTES | STABLECOIN STUDIO

Overview

The Hedera Stablecoin Studio (used to be referred as "Stablecoin Accelerator") is a comprehensive set of tools and resources designed to enable developers to build applications and services that make use of the stable coin, built on Hedera Hashgraph ecosystem. With the Hedera Stablecoin Studio, developers can easily integrate the stable coin into their own applications or create new applications or services that make use of the stable coin's unique features.

The project consists of solidity smart contracts used in the Hedera stable coin project. The Hedera Token Service (HTS) functionality is exposed through an HTS precompiled smart contract implemented and managed by Hedera. We treat the HTS precompiled smart contract as black box during the security auditing and assume its correctness.

The StableCoinFactory contract is responsible for creating new stable coins. It handles the deployment and initialization of multiple smart contracts and the creation of an underlying token through the HTS precompiled contract.

Additionally, the project includes various smart contracts for stable coin operations. These contracts are located in the "extensions" folder and include functionality such as burn, cash-in, delete, freeze/unfreeze, KYC grant/revoke, pause/unpause, rescue, reserve management, role management, supplier admin, token ownership, and wipe. Each contract implements specific operations related to stable coins.

In the context of the project, the Stablecoin Studio introduces multiple roles for operations like burning, pausing, and more. Stable coins split the supply role into cash-in and burn roles, allowing separate accounts to manage these functions. The cash-in role enables the minting and assignment of tokens to any account, either with unlimited or limited minting capabilities. The rescue role allows accounts to manage tokens and HBAR held by the stable coin's smart contract, including transferring tokens from the treasury account.

The project's stable coin extension implementation involves creating a new Hedera Token for each stable coin. The stable coin proxy smart contract is deployed, which points to the HederaTokenManager logic smart contract. This proxy architecture enables upgradability of stable coins.

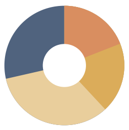
Economic model

A stable coin is a type of cryptocurrency that is designed to maintain a stable value relative to a specific asset or basket of assets. The proof of reserve of the stable coin is, an external feed that provides the backing of the tokens in real world. This may be FIAT or other assets.

Given that the Stablecoin Studio streamlines the process of creating stable coins. The stability of each created stable coin is independent. The proof of reserve of each stable coin requires a third party oracle to feed off-chain data to the Hedera Hashgraph. The current implementation supports the Chainlink proof of reserve interface specification.

It is mentioned in the documentation that stable coins created with the current Stablecoin Studio should be linked to a reserve and ensure an existing data feed is provided through Chainlink Data Feed or compatible protocols. Setting up the reserve and data feed might also require certificates or notarizations from trusted accounting or financial firms.

FINDINGS | STABLECOIN STUDIO



21	0	4	4	7	6
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Stablecoin Studio. Through this audit, we have uncovered 21 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
CON-09	**Centralization Related Risks**	**Centralization**	**Major**	**Acknowledged**
HTM-01	**Initial Token Distribution**	**Centralization**	**Major**	**Acknowledged**
RCP-03	Sum Of Amounts With Different Decimals In Function `_checkReserveAmount()`	Logical Issue	Major	Resolved
SCF-02	**Contract Upgrade Centralization Risk**	**Centralization**	**Major**	**Acknowledged**
CKP-08	Incorrect Return Value In `_checkReserveAmount()`	Volatile Code	Medium	Acknowledged
CON-10	Potential Failure Of Mint	Logical Issue	Medium	Resolved
RKP-01	Admin Role Is Not Strictly Controlled	Logical Issue	Medium	Acknowledged
SCF-01	Logical Issue About The Reserve Feed	Logical Issue	Medium	Resolved
CKP-09	Third-Party Dependency Usage	Design Issue	Minor	Acknowledged
CON-03	Unsafe Integer Cast	Incorrect Calculation	Minor	Resolved

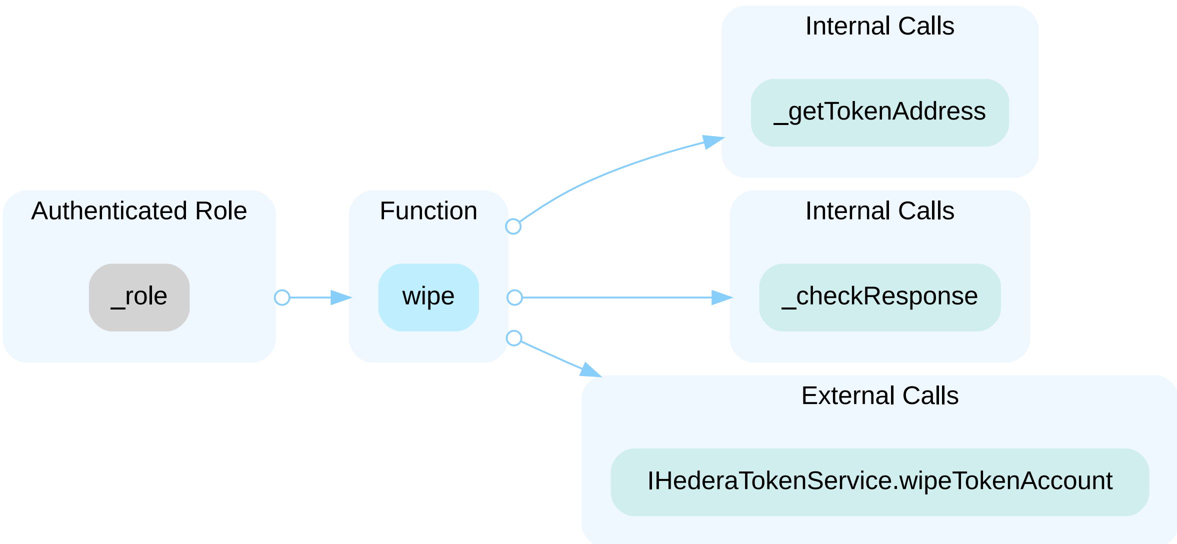
ID	Title	Category	Severity	Status
CON-04	Inappropriate Data Type For Parameters And Fields	Volatile Code	Minor	Acknowledged
HRC-01	Decimals Too Small	Logical Issue	Minor	Acknowledged
HTM-03	Pull-Over-Push Pattern	Logical Issue	Minor	Acknowledged
RCP-04	Missing Zero Address Validation	Volatile Code	Minor	Resolved
SAC-01	Missing Validations When Increase And Decrease Supplier Allowance	Inconsistency, Logical Issue	Minor	Resolved
CON-05	Inconsistent Solidity Versions	Language Version	Informational	Acknowledged
CON-07	Redundant Code Components	Volatile Code	Informational	Partially Resolved
CON-08	Incorrect Variable Data Types	Logical Issue	Informational	Acknowledged
CON-11	Information About `generateKey()`	Logical Issue	Informational	Acknowledged
HTM-02	Unused Return Variable	Logical Issue	Informational	Resolved
HTM-04	Information About `_hederaTokenManagerAddress`	Logical Issue	Informational	Acknowledged

CON-09 | CENTRALIZATION RELATED RISKS

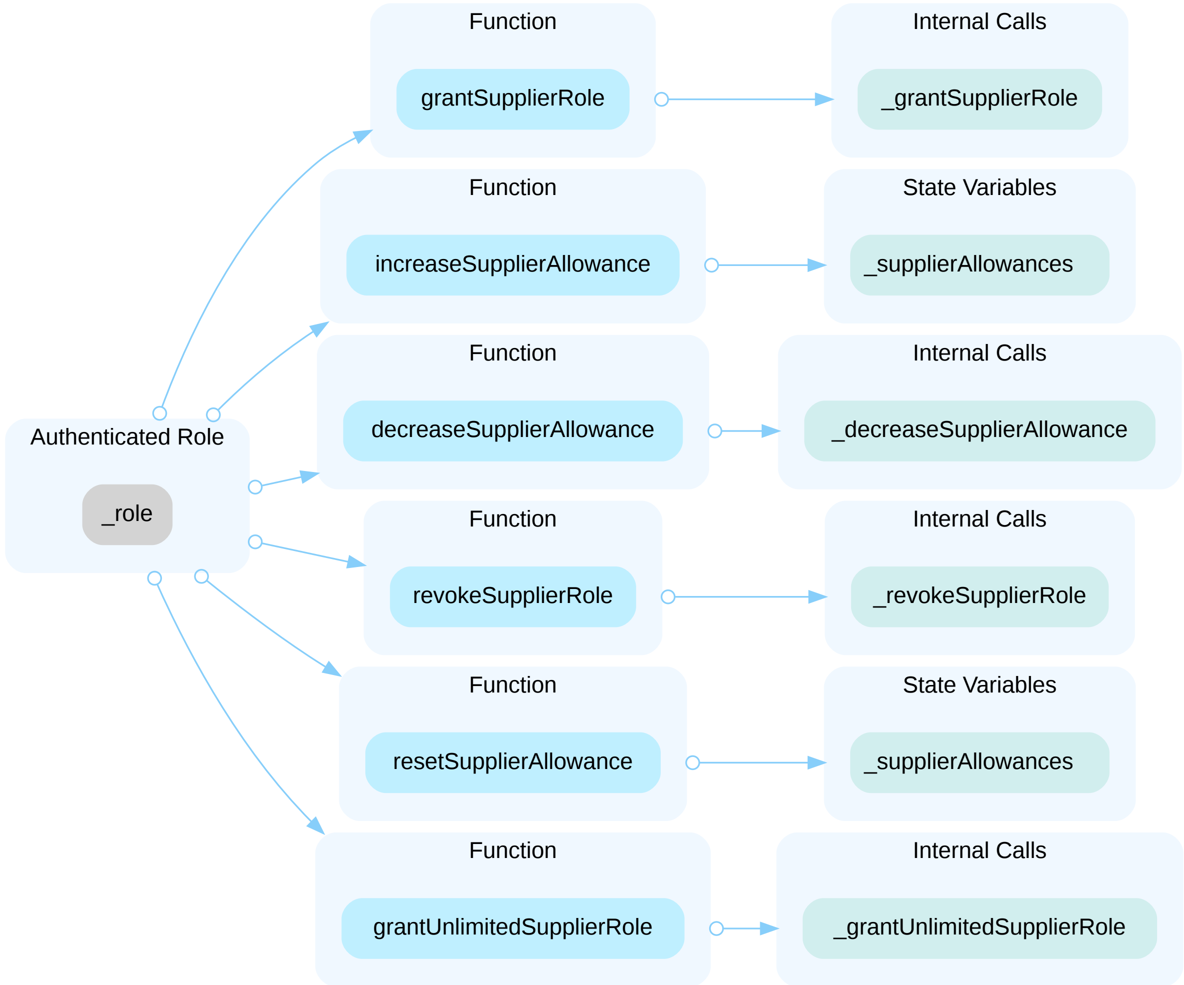
Category	Severity	Location	Status
Centralization	Major	contracts/HederaReserve.sol: 65, 75; contracts/HederaTokenManager.sol: 155, 216, 242; contracts/StableCoinFactory.sol: 199, 230, 249, 262; contracts/extensions/Burnable.sol: 15; contracts/extensions/CashIn.sol: 17; contracts/extensions/Deletable.sol: 14; contracts/extensions/Freezable.sol: 15, 41; contracts/extensions/KYC.sol: 15, 41; contracts/extensions/Pausable.sol: 14, 36; contracts/extensions/Rescatable.sol: 25, 63; contracts/extensions/Reserve.sol: 57, 78; contracts/extensions/RoleManagement.sol: 15, 48; contracts/extensions/Roles.sol: 161, 177, 227; contracts/extensions/SupplierAdmin.sol: 47, 67, 85, 103, 132, 164; contracts/extensions/Wipeable.sol: 21	Acknowledged

Description

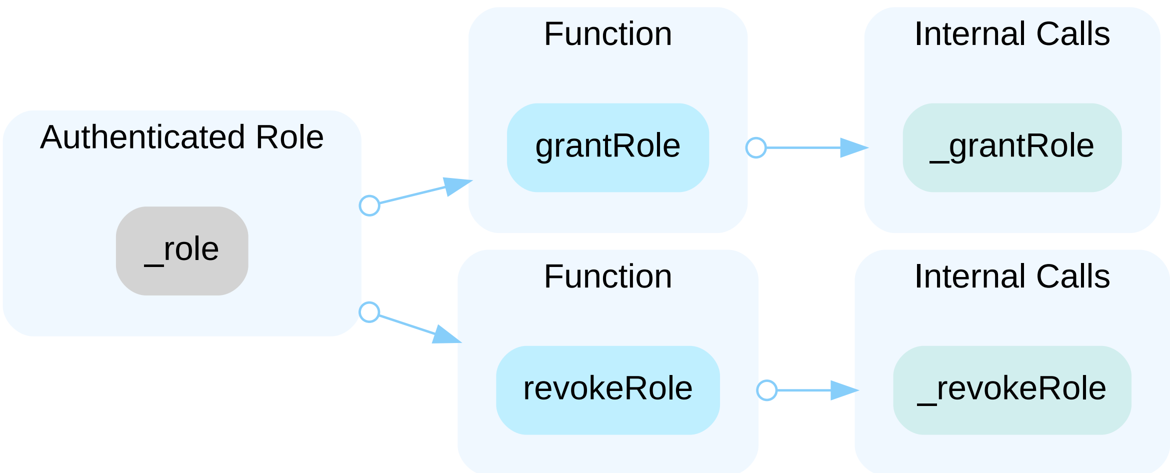
In the contract `Wipeable` the role `WIPE` has authority over the functions shown in the diagram below. Any compromise to the `WIPE` account may allow the hacker to take advantage of this authority to wipe a token amount from an account.



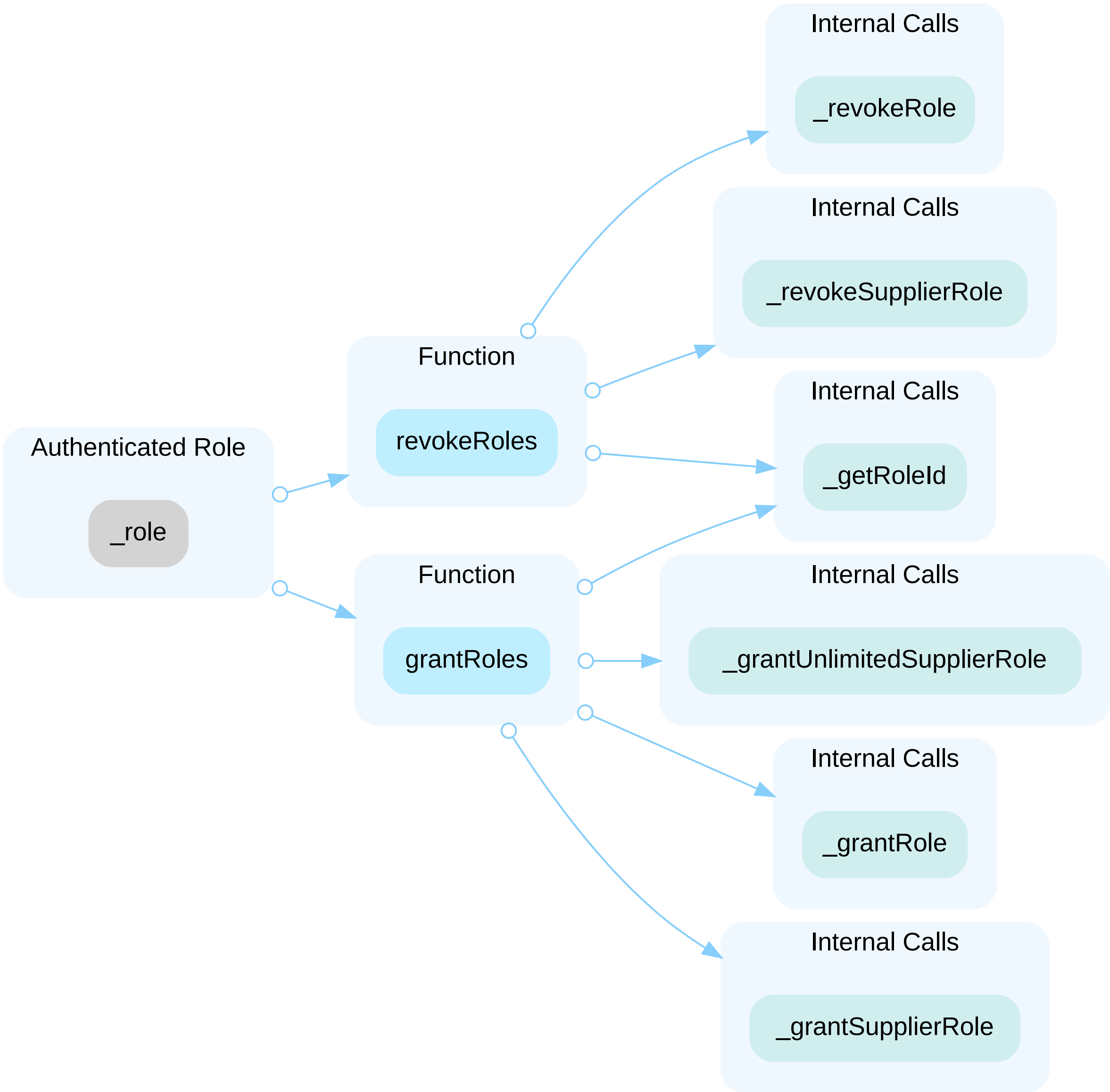
In the contract `SupplierAdmin` the role `ADMIN` has authority over the functions shown in the diagram below. Any compromise to the `ADMIN` account may allow the hacker to take advantage of this authority to manage the suppliers and their allowances.



In the contract `Roles` 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 to grant/revoke a role to/from an account.

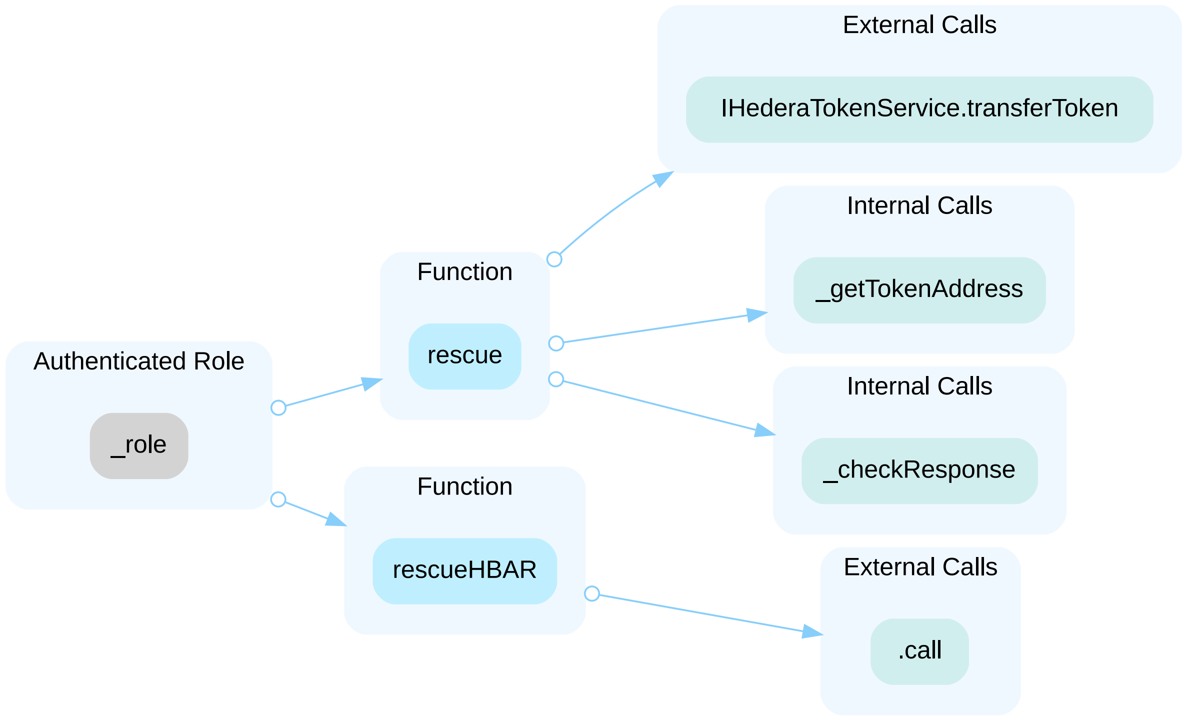


In the contract `RoleManagement` the role `ADMIN` has authority over the functions shown in the diagram below. Any compromise to the `ADMIN` account may allow the hacker to take advantage of this authority to grant/revoke the provided "roles" to/from all the "accounts".

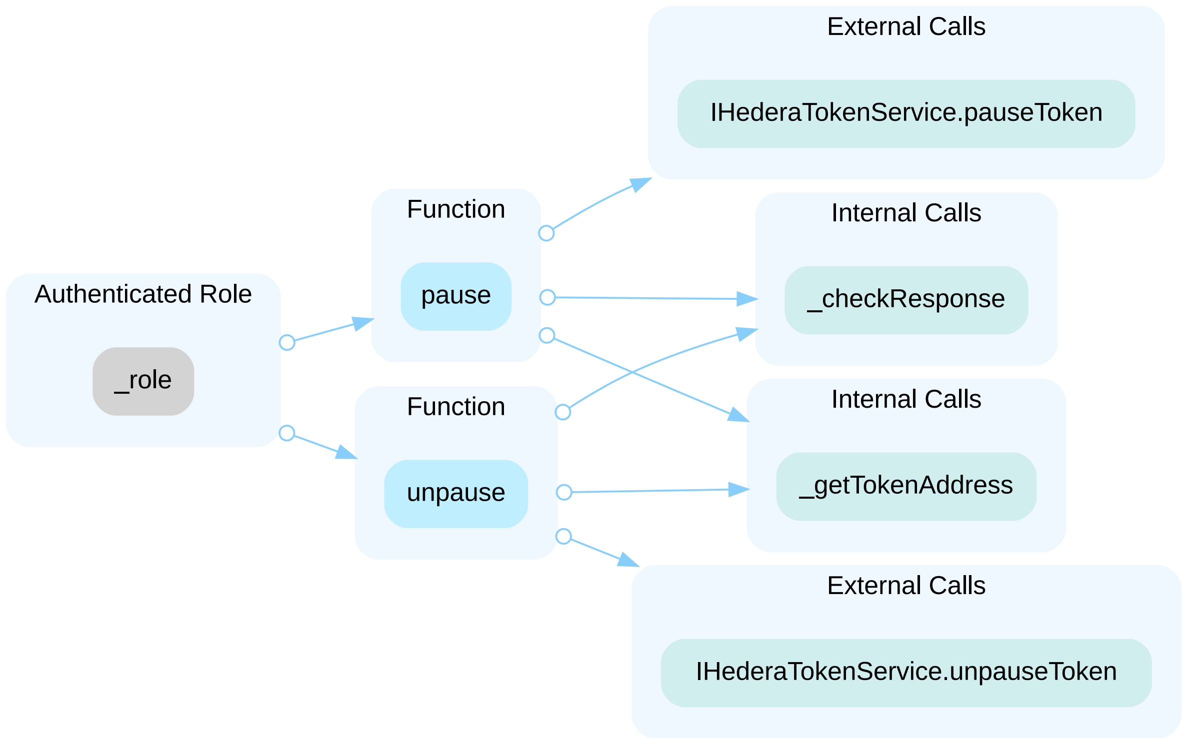


In the contract `Reserve` the role `ADMIN` has authority over the functions shown in the diagram below. Any compromise to the `ADMIN` account may allow the hacker to take advantage of this authority to update the reserve address.

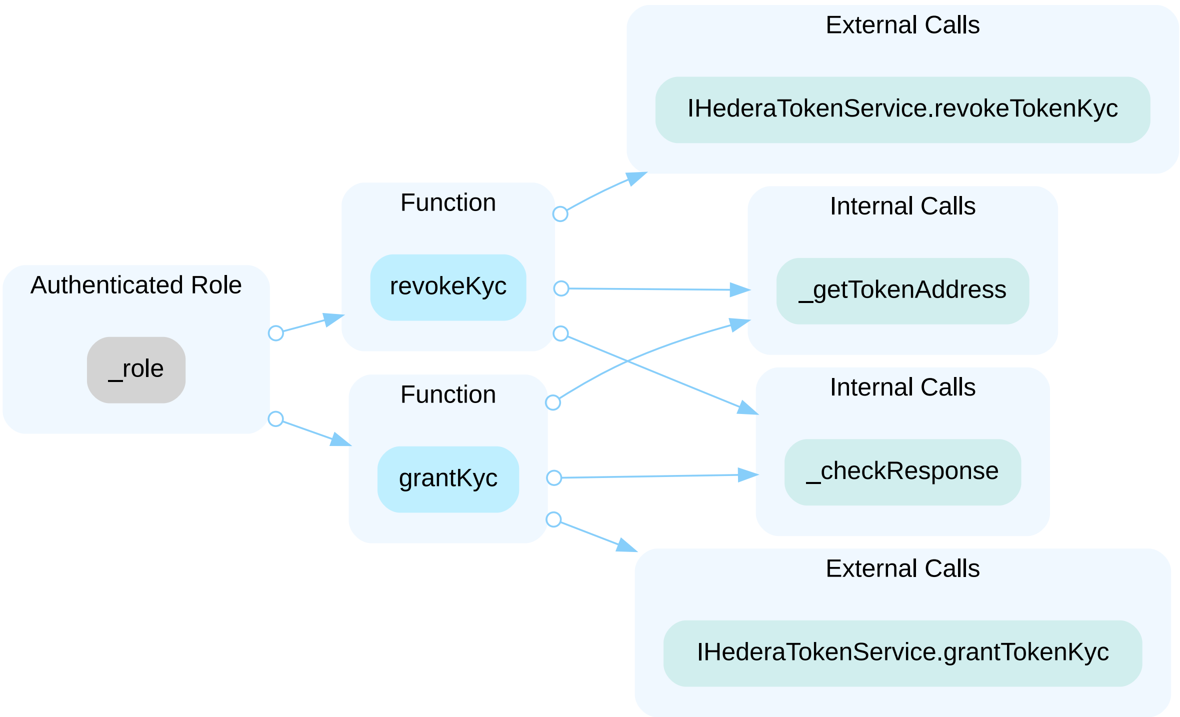
In the contract `Rescatable` the role `RESCUE` has authority over the functions shown in the diagram below. Any compromise to the `RESCUE` account may allow the hacker to take advantage of this authority to rescue tokens and HBARs from contractTokenOwner to the rescuer.



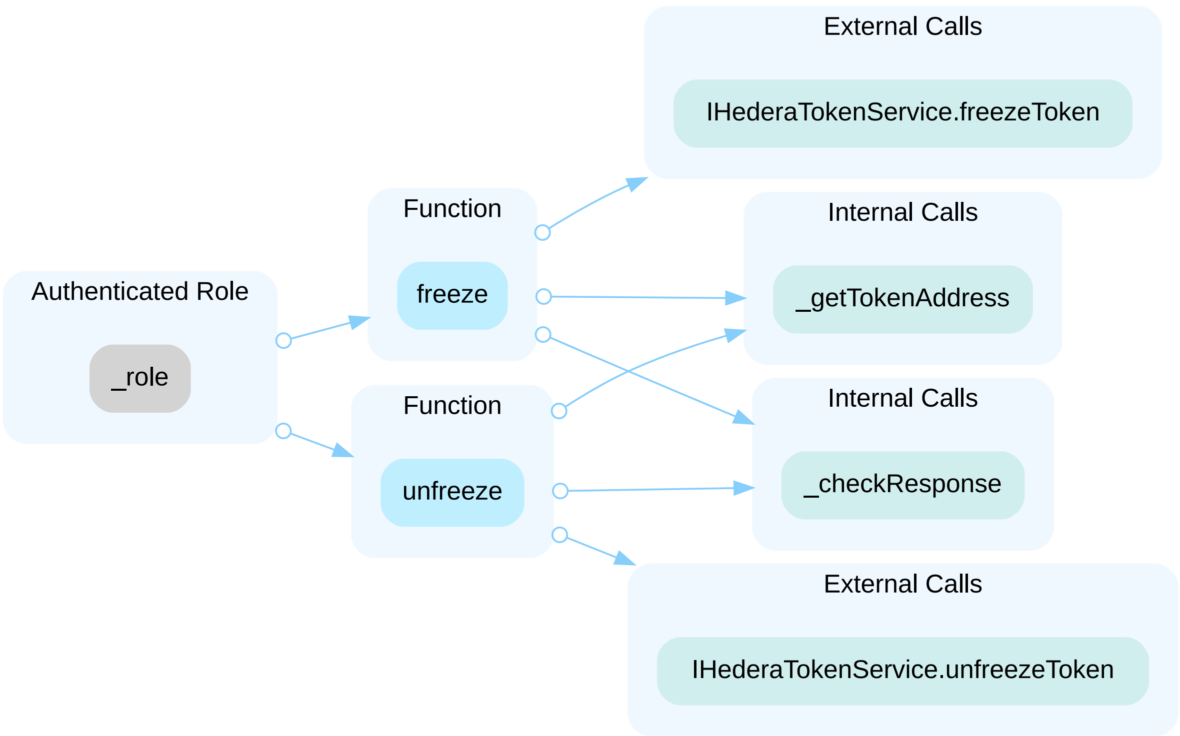
In the contract `Pausable` the role `PAUSE` has authority over the functions shown in the diagram below. Any compromise to the `PAUSE` account may allow the hacker to take advantage of this authority to pause the token in order to prevent it from being involved in any kind of operation.



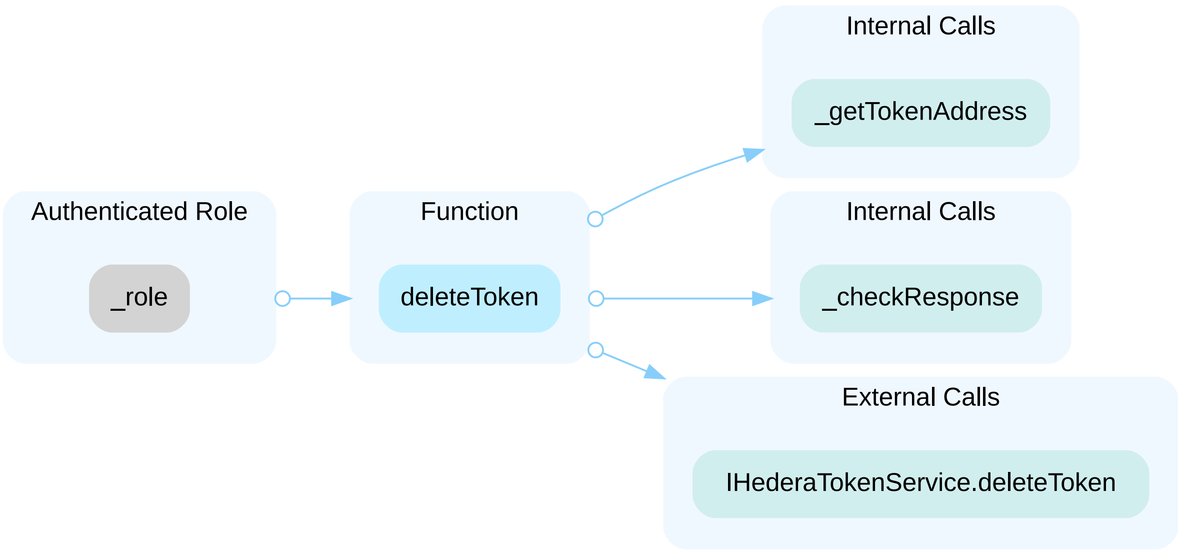
In the contract `KYC` the role `KYC` has authority over the functions shown in the diagram below. Any compromise to the `KYC` account may allow the hacker to take advantage of this authority to grant KYC to account for the token.



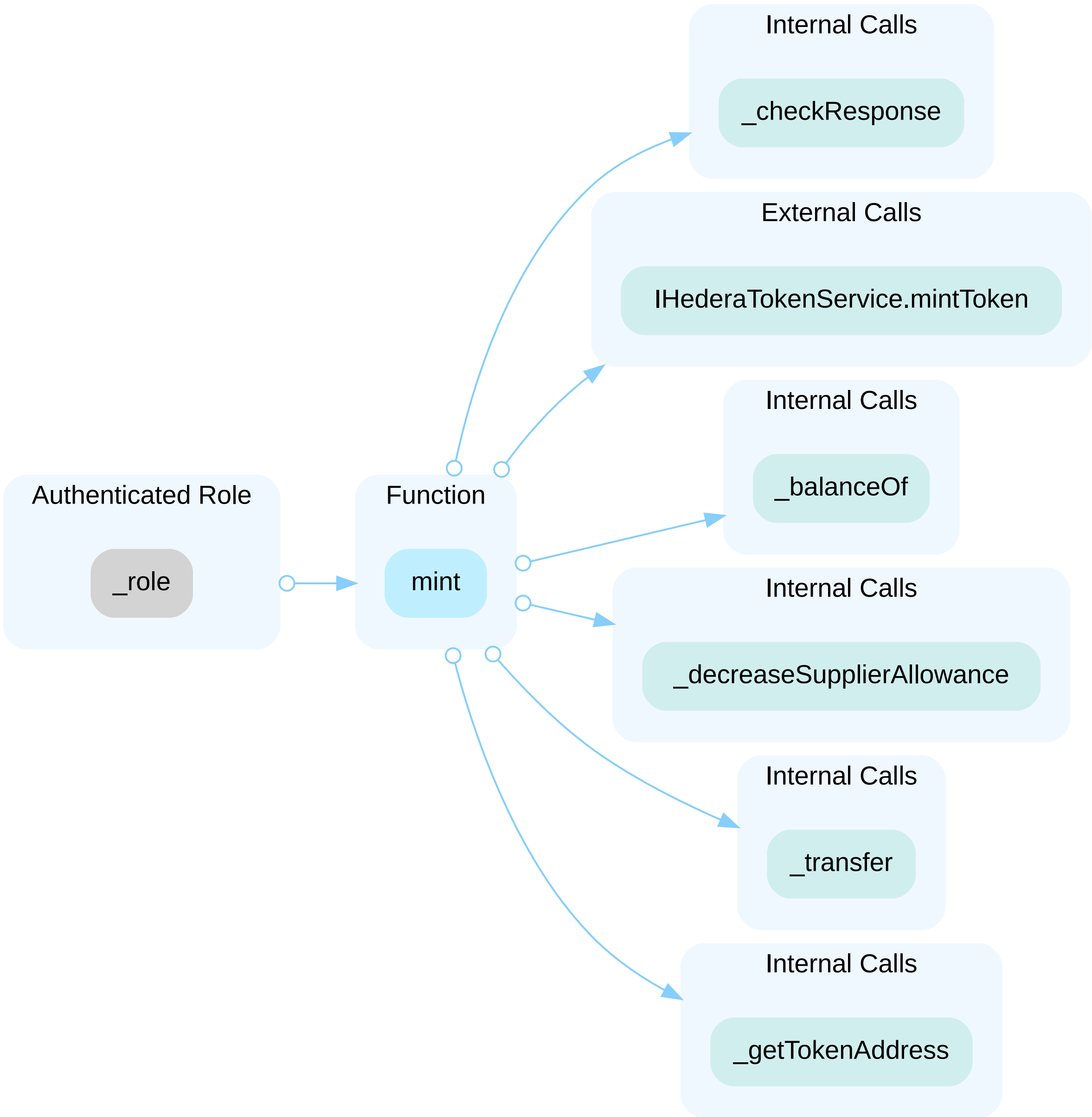
In the contract `Freezable` the role `FREEZE` has authority over the functions shown in the diagram below. Any compromise to the `FREEZE` account may allow the hacker to take advantage of this authority to freeze transfers of the token for the `account`.



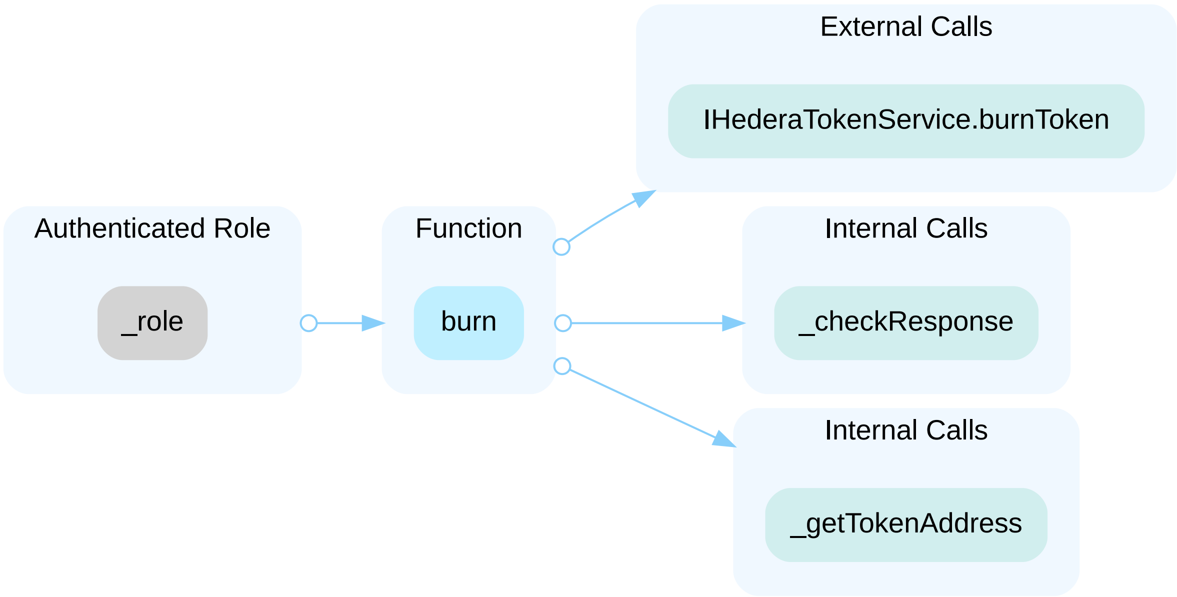
In the contract `Deletable` the role `DELETE` has authority over the functions shown in the diagram below. Any compromise to the `DELETE` account may allow the hacker to take advantage of this authority to delete the token.



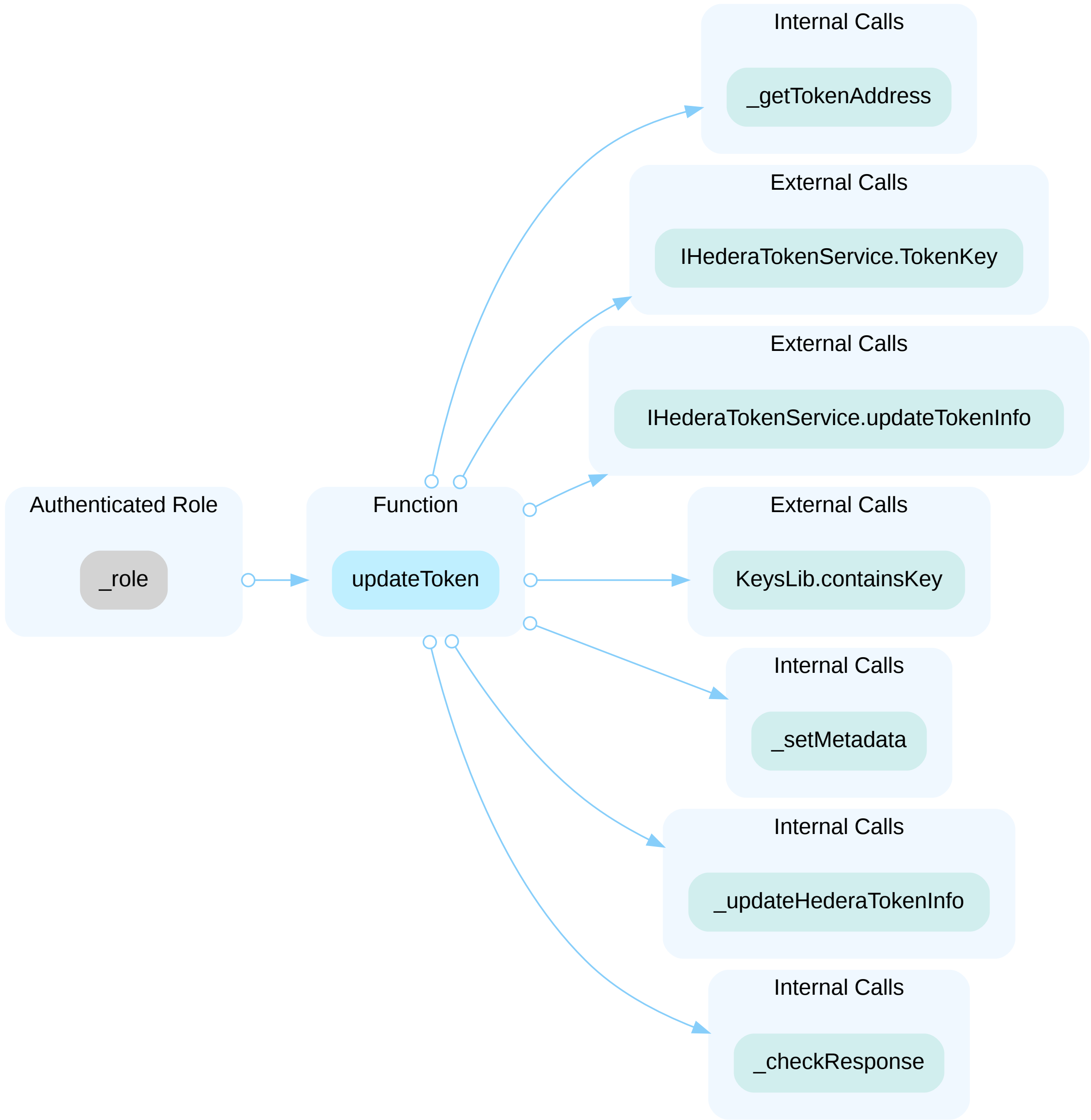
In the contract `CashIn` the role `CASHIN` has authority over the functions shown in the diagram below. Any compromise to the `CASHIN` account may allow the hacker to take advantage of this authority to create an `amount` of tokens and transfer them to an `account`, increasing the total supply.



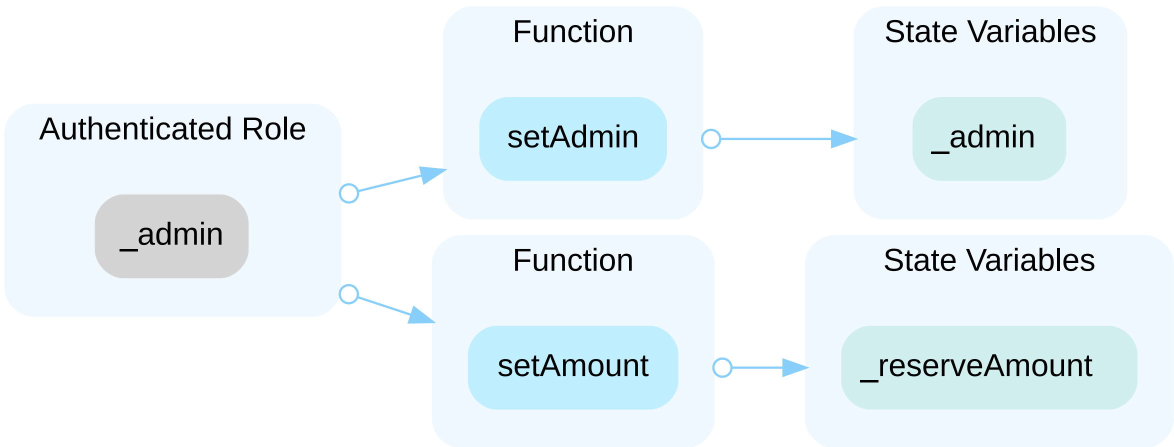
In the contract `Burnable` the role `BURN` has authority over the functions shown in the diagram below. Any compromise to the `BURN` account may allow the hacker to take advantage of this authority to burn an `amount` of tokens owned by the treasury account.



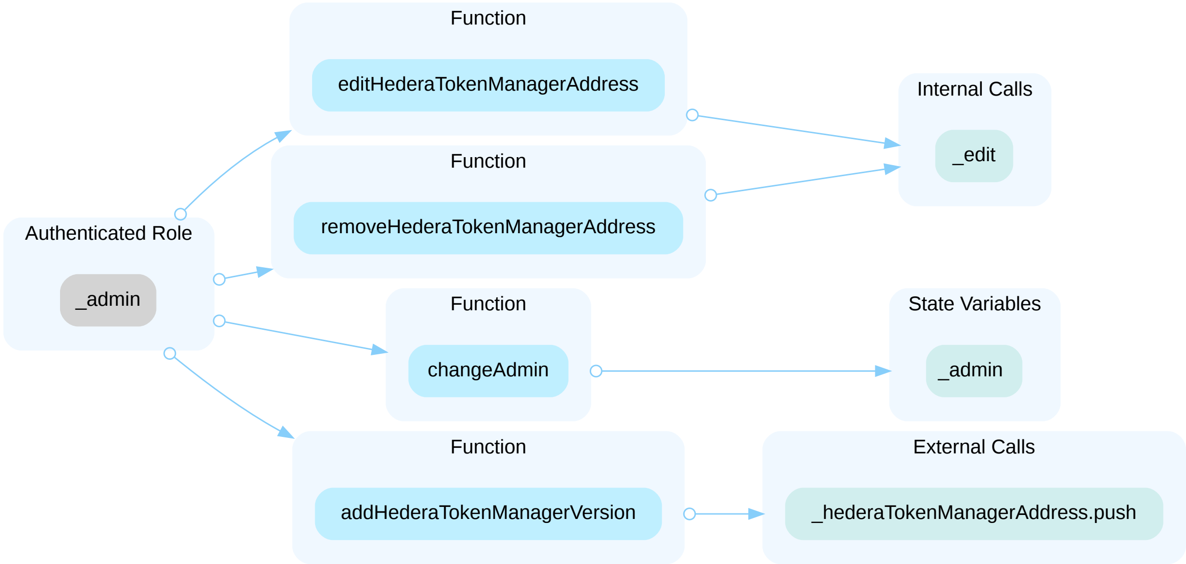
In the contract `HederaTokenManager` the role `ADMIN` has authority over the functions shown in the diagram below. Any compromise to the `ADMIN` account may allow the hacker to take advantage of this authority to update token configurations.



In the contract `HederaReserve` the role `_admin` has authority over the functions shown in the diagram below. Any compromise to the `_admin` account may allow the hacker to take advantage of this authority to set a new reserve amount or a new admin address.



In the contract `StableCoinFactory` the role `_admin` has authority over the functions shown in the diagram below. Any compromise to the `_admin` account may allow the hacker to take advantage of this authority to add/edit/remove stable coin contract addresses and change the admin address.



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

[Swirlds Labs]:

Regarding the centralization of the permissions to perform any operation task on the stable coin:

From the beggining of the project, we have all assumed this permissions centralization. As in the case of any DLT, to comprise a private key is a major risk which is well known. Using Hedera complex keys, what could fix this permission centralization issue, was never a goal of the project. Both we and the client don't agree to use any time-lock mechanism for token operational functionalities since most of them would need to be executed asap, rather than administrative operations like upgrading the contracts.

These issues are related to the centralization of the permissions to perform any operational task on the token or any administrative task on the stable coin on the basis that operations are not controlled by multi-signatures.

While the solution could be extended to leverage Hedera's native multi-sig capabilities, we assume that a token issuer will be using a key custody provider to manage the critical processes related to minting, upgrading, etc... via external workflows involving multiple parties

Token lock mechanisms for stable coins are inefficient, minting a token needs to be near instant in order to issue the minted token to the receiving user as efficiently as possible. That said, such locking mechanisms can be implemented off chain via key custodians too if required.

With regards stable coin administrative operations, the proxy admin can now to be delegated to an administrative account that could be controlled by a key custodian, or delegated to a time-lock contract (see remediation commit).

Owner changes are also now conditioned to an Owner2Steps mechanism (see remediation commit).

Finally, for Proof Of Reserve, the contracts are an example, the process of updating the reserve value would typically be delegated to an oracle, ensuring that the governance for reserve management is managed accordingly.

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

HTM-01 | INITIAL TOKEN DISTRIBUTION

Category	Severity	Location	Status
Centralization	Major	contracts/HederaTokenManager.sol: 84~90	Acknowledged

Description

The `initialize` function in the `HederaTokenManager` contract will initialize the stable coin from the proxy and create a fungible token with the specified properties via the `IHederaTokenService` .

All the `initialTotalSupply` of the fungible tokens are sent to the treasury. This could be a centralization risk as the anonymous deployer can distribute tokens without obtaining the consensus of the community. Any compromise to the deployer account that holds undistributed tokens may allow the attacker to steal and sell tokens on the market, resulting in severe damage to the project.

Recommendation

It's recommended 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 shall make enough efforts to restrict the access of the private key. A multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to the 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 project teams with a third-party KYC provider to create greater accountability.

Alleviation

[Swirlds Labs]:

Regarding the initial token distribution plan, the user who creates the stable coin could use the stable coin metadata to publish such plan, so anyone could access this metadata to know the initial distribution plan.

Note that the solution is for a stable coin, vesting schedules don't apply.

These issues are related to the centralization of the permissions to perform any operational task on the token or any administrative task on the stable coin on the basis that operations are not controlled by multi-signatures.

While the solution could be extended to leverage Hedera's native multi-sig capabilities, we assume that a token issuer will be using a key custody provider to manage the critical processes related to minting, upgrading, etc... via external workflows involving multiple parties

Token lock mechanisms for stable coins are inefficient, minting a token needs to be near instant in order to issue the minted token to the receiving user as efficiently as possible. That said, such locking mechanisms can be implemented off chain via key custodians too if required.

With regards stable coin administrative operations, the proxy admin can now to be delegated to an administrative account that could be controlled by a key custodian, or delegated to a time-lock contract (see remediation commit).

Owner changes are also now conditioned to an Owner2Steps mechanism (see remediation commit).

Finally, for Proof Of Reserve, the contracts are an example, the process of updating the reserve value would typically be delegated to an oracle, ensuring that the governance for reserve management is managed accordingly.

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

RCP-03 | SUM OF AMOUNTS WITH DIFFERENT DECIMALS IN FUNCTION `_checkReserveAmount()`

Category	Severity	Location	Status
Logical Issue	● Major	contracts/extensions/Reserve.sol: 107~108, 113~114	● Resolved

Description

The internal function `_checkReserveAmount()` is used by the modifiers `checkReserveIncrease` and `checkReserveDecrease` to check if the current reserve is enough for a certain amount of tokens.

Since the token decimals and the reserve decimals might not be equal. the `_checkReserveAmount()` will modify the input amount or the reserve amount to make them having the same decimals. That is, if token decimals is smaller, then the token input amount will be multiplied by the difference of the reserve decimals and the token decimals.

```
106        } else if (tokenDecimals < reserveDecimals) {
107            amount = amount * (10 ** (reserveDecimals - tokenDecimals));
108        }
```

In this case, if the input parameter `less` is `false`, the reserve amount `currentReserve` will be compared to `_totalSupply() + amount`. The right hand side of the inequality is the sum of total token supply and input token amount. However, here the total supply, `_totalSupply()`, is using token decimals, while the input amount, `amount`, is using the modified reserve decimals. The summation of two amounts of different decimals will lead to the return value of `currentReserve >= _totalSupply() + amount` is a lot easier to be `true`, since `_totalSupply()` is `reserveDecimals - tokenDecimals` times smaller when calculating in the reserve decimals.

```
110        if (less) {
111            return currentReserve >= amount;
112        } else {
113            return currentReserve >= _totalSupply() + amount;
114        }
115
```

Since the modifier `checkReserveIncrease` calls `_checkReserveAmount()` inside, and it is applied to the `CashIn.mint()` function. When the above described conditions are met, the `mint()` function will be processed unexpectedly, and breaks the stability backed by the reserve assets.

Scenario

Summary of the two conditions mentioned in the description section:

1. `tokenDecimals < reserveDecimals`

2. `less == false`

Proof of Concept

This proof of concept should not be used as a test in production directly, since the behavior of oracle is not simulated.

```
...
describe('HederaTokenManager Tests', function () {
    before(async function () {
...
        // Deploy Token using Client
        const result = await deployContractsWithSDK({
            name: TokenName,
            symbol: TokenSymbol,
            decimals: BigNumber.from(1),
            initialSupply: BigNumber.from(900),
            maxSupply: MAX_SUPPLY.toString(),
            memo: TokenMemo,
            account: operatorAccount,
            privateKey: operatorPriKey,
            publicKey: operatorPubKey,
            isED25519Type: operatorIsE25519,
            initialAmountDataFeed: BigNumber.from('1000').toString(),
        })
...
    })
...
    it('Mint should revert when reserve is not enough', async () => {
        const initialTotalSupply = await getTotalSupply(
            proxyAddress,
            operatorClient
        )

        await expect(
            Mint(
                proxyAddress,
                BigNumber.from(200),
                operatorClient,
                operatorAccount,
                operatorIsE25519
            )
        ).to.eventually.be.rejectedWith(Error)
    })
})
```

Recommendation

Recommend properly reviewing the design, carefully handling the calculation when the decimals are different, and adding enough test cases to cover the corner cases.

Alleviation

Fixed in commit 5dbe8450dfd835b4d34743e6644b3930f434c8fd.

SCF-02 | CONTRACT UPGRADE CENTRALIZATION RISK

Category	Severity	Location	Status
Centralization	**Major**	**contracts/StableCoinFactory.sol: 115~123, 144~154**	**Acknowledged**

Description

In the StableCoinFactory contract, the caller of the function `deployStableCoin()` will be the owner of the proxy admin of both reserveProxy and stableCoinProxy. The owner has the authority to update the implementation contract behind the HederaReserve and the HederaTokenManager contract.

Any compromise to the owner account may allow a hacker to take advantage of this authority and control the implementation contract which is pointed by proxy and therefore execute potential malicious functionality in the implementation contract.

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

[Swirlds Labs]:

Regarding the centralization of the permissions to perform any administrative task on the stable coin's or the PoR's proxy admin:

From the beggining of the project, we have all assumed this permissions centralization. As in the case of any DLT, to comprise a private key is a major risk which is well known. Using Hedera complex keys, what could fix this permission centralization issue, was never a goal of the project. In the case of stable coin administrative operations, we are going to add that, during the creation process, users can choose an account to be the owner of stable coin's proxy admin. For example, a time-lock contract can be set as the owner. Moreover, we are going to implemented an Owner2Steps mechanism to change not only the stable coin's proxy admin owner, but also the factory's proxy admin owner. Therefore, stable coin management operations have a more robust governance system. Finally, in the case of the PoR, the contracts are only an example for the users, so we didn't consider to include this governance mechanism that users can implement by themselves.

CKP-08 | INCORRECT RETURN VALUE IN `_checkReserveAmount()`

Category	Severity	Location	Status
Volatile Code	Medium	contracts/extensions/CashIn.sol: 24~25; contracts/extensions/Reserve.sol: 22~23, 93~94	Acknowledged

Description

In the contract `Reserve`, there is a modifier `checkReserveIncrease()` that can check if the current reserve is enough for a certain amount of tokens comparing with the sum of amount plus total supply, according to the comments. The modifier calls the internal function `_checkReserveAmount()`, such that if `_checkReserveAmount()` return `false`, the modifier will revert. Otherwise, it will pass.

In the contract `CashIn`, the `checkReserveIncrease()` modifier is applied to the `mint()` function to check if the reserve is enough to mint the input amount of tokens. However, if the `_reserveAddress` in the contract `Reserve` is `address(0)`, the internal function `_checkReserveAmount()` will always return `true`, and the `checkReserveIncrease()` modifier will pass. As a result, the `CashIn.mint()` function will not revert as expected.

```
    function _checkReserveAmount(uint256 amount, bool less) private view returns
(bool) {
        if (_reserveAddress == address(0)) return true;
        ...
    }
```

Recommendation

Recommend properly reviewing the design and making sure that invalid `_reserveAddress` values will be reverted.

Alleviation

[Swirlds Labs]:

In this case, zero address indicates that the stable coin doesn't have any Reserve contract, so no check is needed in order to cash in tokens.

Proof of reserve is optional and may not be included at the time of deployment. As a result, if there is no reserve address, there is nothing to check when minting

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

CON-10 | POTENTIAL FAILURE OF MINT

Category	Severity	Location	Status
Logical Issue	Medium	contracts/HederaTokenManager.sol: 186~190; contracts/extensions/CashIn.sol: 44; contracts/library/KeysLib.sol: 24~37	Resolved

Description

In the `updateToken()` function, in case the `hederaKeys` contains the `_SUPPLY_KEY_BIT`, the treasury will be set differently depending on the `hederaKeys[i].key.delegatableContractId` value.

```
if (KeysLib.containsKey(_SUPPLY_KEY_BIT, hederaKeys[i].keyType))
              newTreasury = hederaKeys[i].key.delegatableContractId ==
                  address(this)
                  ? address(this)
                  : msg.sender;
```

According to the logic in the `KeyLibs` contract, `delegatableContractId` will be set to `stableCoinProxyAddress` only in case the `publicKey` is empty. Then the treasury will be `stableCoinProxyAddress`.

```
if (publicKey.length == 0)
          key.delegatableContractId = stableCoinProxyAddress;
```

Looking back to the first code snippet, if the `delegatableContractId` is not `address(this)`, the treasury will be `msg.sender`, which will be an EOA account. In this case, the `mint()` function in the `CashIn` contract might face transfer failures, since the minted tokens were sent to the treasury, and the contract might not have enough balance.

```
 function mint(address account,int64 amount)...
{
       (int64 responseCode, , ) =
IHederaTokenService(_PRECOMPILED_ADDRESS).mintToken(currentTokenAddress, amount, new
bytes[](0));

        bool success = _checkResponse(responseCode);

        if (!((_balanceOf(address(this)) - balance) == uint256(uint64(amount))))
            revert('The smart contract is not the treasury account');
        _transfer(account, amount);//CertiK: potential failure
}
```

Recommendation

We recommend the team adding validations before calling the `_transfer()` function.

Alleviation

The team heeded our advice and resolved the issue in commit 5dbe8450dfd835b4d34743e6644b3930f434c8fd.

RKP-01 | ADMIN ROLE IS NOT STRICTLY CONTROLLED

Category	Severity	Location	Status
Logical Issue	Medium	contracts/extensions/Roles.sol	Acknowledged

Description

Unlike the OpenZeppelin `AccessControl` contract, there is no role admin concept in the `RoleData` struct.

Hence an address with the `ADMIN_ROLE` can grant the `ADMIN_ROLE` to other addresses, and the new addresses with the `ADMIN_ROLE` can grant the `ADMIN_ROLE` to more addresses or revoke the old `ADMIN_ROLE` addresses.

Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/tree/master/contracts/access

Recommendation

We would like to confirm with the client if the current implementation aligns with the project design.

Alleviation

[Swirlds Labs]:

We don't have role admin and this is agreed with the client.

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

SCF-01 | LOGICAL ISSUE ABOUT THE RESERVE FEED

Category	Severity	Location	Status
Logical Issue	● Medium	contracts/StableCoinFactory.sol: 130~133	● Resolved

Description

In the `deployStableCoin()` function, the `reserveInitialAmount` will be validated if it is less than the `tokenInitialSupply` . However, the `reserveInitialAmount` is retrieved from the `HederaReserve` rather than the `Reserve` .

```
        else if (reserveAddress != address(0)) {
          (, int256 reserveInitialAmount, , , ) = HederaReserve(
              reserveAddress
          ).latestRoundData();
```

According to the contract logic, the `latestRoundData()` from the `HederaReserve` is set by the admin. The reserve amount in the Reserve is retrieved from the Chainlink reserved feed. The interface that the reserve data feed must implement for the stable coin to be able to interact with, is defined by AggregatorV3Interface and used by Chainlink.

Recommendation

We recommend the client review the logic and fix the issue.

Alleviation

The team heeded our advice and resolved the issue in commit 5dbe8450dfd835b4d34743e6644b3930f434c8fd.

CKP-09 | THIRD-PARTY DEPENDENCY USAGE

Category	Severity	Location	Status
Design Issue	Minor	contracts/extensions/CashIn.sol: 36; contracts/extensions/Reserve.sol: 97~98, 123~124; contracts/extensions/TokenOwner.sol: 23, 25	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.

```
23      address internal constant _PRECOMPILED_ADDRESS = address(0x167);
```

```
23      (int64 responseCode, , ) = IHederaTokenService(_PRECOMPILED_ADDRESS)
24              .mintToken(currentTokenAddress, amount, new bytes[](0));
```

The contract `TokenOwner` interacts with third party contract with `IHederaTokenService` interface via `_PRECOMPILED_ADDRESS`. The implementation of `IHederaTokenService` is defined in another repo and is out of the audit scope: https://github.com/hashgraph/hedera-smart-contracts/blob/main/contracts/hts-precompile/HederaTokenService.sol

```
25      address private _tokenAddress;
```

The contract `TokenOwner` interacts with third party contract with `IERC20Upgradeable` interface via `_tokenAddress`.

The contract `Reserve` interacts with third party contract with `AggregatorV3Interface` via `Chainlink`.

```
97      uint8 reserveDecimals = AggregatorV3Interface(_reserveAddress)
98          .decimals();
```

```
122     if (_reserveAddress != address(0)) {
123         (, int256 answer, , , ) = AggregatorV3Interface(_reserveAddress)
124             .latestRoundData();
125         return answer;
126     }
```

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.

Alleviation

[Swirlds Labs]: We will pay attention to changes to any of these third parties contracts since these changes may fix errors or increase security issues.

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

CON-03 | UNSAFE INTEGER CAST

Category	Severity	Location	Status
Incorrect Calculation	Minor	contracts/HederaTokenManager.sol: 276; contracts/StableCoinFactory.sol: 370, 372, 373, 374; contracts/extensions/Burnable.sol: 23; contracts/extensions/CashIn.sol: 24, 30, 41; contracts/extensions/Rescatable.sol: 33, 70; contracts/extensions/Wipeable.sol: 31	Resolved

Description

Type casting refers to changing an variable of one data type into another. The code contains an unsafe cast between integer types, which may result in unexpected truncation or sign flipping of the value.

```
276              uint256(uint64(amount)),
```

Casted expression `amount` has estimated range [-9223372036854775808, 9223372036854775807] but target type `uint64` has range [0, 18446744073709551615].

```
370              revert LessThan(uint256(reserveInitialAmount), 0);
```

Casted expression `reserveInitialAmount` has estimated range [-57896044618658097711785492504343953926634992332820282019728792003956564819968, -1] but target type `uint256` has range [0, 115792089237316195423570985008687907853269984665640564039457584007913129639935].

```
372          uint256 initialReserve = uint256(reserveInitialAmount);
```

Casted expression `reserveInitialAmount` has estimated range [-57896044618658097711785492504343953926634992332820282019728792003956564819968, 57896044618658097711785492504343953926634992332820282019728792003956564819967] but target type `uint256` has range [0, 115792089237316195423570985008687907853269984665640564039457584007913129639935].

```
373          uint32 _tokenDecimals = uint32(tokenDecimals);
```

Casted expression `tokenDecimals` has estimated range [-2147483648, 2147483647] but target type `uint32` has range [0, 4294967295].

```
374          uint256 _tokenInitialSupply = uint256(uint64(tokenInitialSupply));
```

Casted expression `tokenInitialSupply` has estimated range [-9223372036854775808, 9223372036854775807] but target type `uint64` has range [0, 18446744073709551615].

```
23              uint256(uint64(amount)),
```

Casted expression `amount` has estimated range [-9223372036854775808, 9223372036854775807] but target type `uint64` has range [0, 18446744073709551615].

```
24          checkReserveIncrease(uint256(uint64(amount)))
```

Casted expression `amount` has estimated range [-9223372036854775808, 9223372036854775807] but target type `uint64` has range [0, 18446744073709551615].

```
30              _decreaseSupplierAllowance(msg.sender, uint256(uint64(amount)));
```

Casted expression `amount` has estimated range [-9223372036854775808, 9223372036854775807] but target type `uint64` has range [0, 18446744073709551615].

```
41          if (!((_balanceOf(address(this)) - balance) == uint256(uint64(amount))))
```

Casted expression `amount` has estimated range [-9223372036854775808, 9223372036854775807] but target type `uint64` has range [0, 18446744073709551615].

```
33              uint256(uint64(amount)),
```

Casted expression `amount` has estimated range [-9223372036854775808, 9223372036854775807] but target type `uint64` has range [0, 18446744073709551615].

```
70              uint256(uint64(amount)),
```

Casted expression `amount` has estimated range [0, 115792089237316195423570985008687907853269984665640564039457584007913129639935] but target type `uint64` has range [0, 18446744073709551615].

```
31              uint256(uint64(amount)),
```

Casted expression `amount` has estimated range [-9223372036854775808, 9223372036854775807] but target type `uint64` has range [0, 18446744073709551615].

Recommendation

It is recommended to check the bounds of integer values before casting. Alternatively, consider using the `SafeCast` library from OpenZeppelin to perform safe type casting and prevent undesired behavior.

Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/cf86fd9962701396457e50ab0d6cc78aa29a5ebc/contracts/utils/math/SafeCast.sol

Alleviation

[Swirlds Labs]:

Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/hashgraph/hedera-accelerator-stablecoin/commit/5dbe8450dfd835b4d34743e6644b3930f434c8fd

[CertiK]:

Thank you for the reply. We have checked the new commit and most of the locations in this finding have been resovled. However, we still find some locations missing the fixes:

Cash.sol 24,30,41 Rescatable.sol 70

[Swirlds Labs]:

Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/hashgraph/hedera-accelerator-stablecoin/commit/8a36504d6ba2f976bf9fa2a131bb14190a453de9

CON-04 | INAPPROPRIATE DATA TYPE FOR PARAMETERS AND FIELDS

Category	Severity	Location	Status
Volatile Code	Minor	contracts/HederaTokenManager.sol: 84~85; contracts/Interfaces/IHederaTokenManager.sol: 8~9; contracts/Interfaces/IStableCoinFactory.sol: 8~9; contracts/StableCoinFactory.sol: 92~93, 108~109, 135~136	Acknowledged

Description

There are several variables that are defined as integer types, but are used as unsigned types.

`StableCoinFactory.deployStableCoin()`

The function `deployStableCoin()` takes a parameter `TokenStruct calldata requestedToken`. The struct `TokenStruct` is defined in `IStableCoinFactory`, and there are four fields, `tokenMaxSupply`, `tokenInitialSupply`, `tokenDecimals` and `reserveInitialAmount`. These four fields are defined as integer types, while the ideal value ranges seem to belong to unsigned integer types.

```
struct TokenStruct {
    string tokenName;
    string tokenSymbol;
    bool freeze;
    bool supplyType;
    int64 tokenMaxSupply;
    int64 tokenInitialSupply;
    int32 tokenDecimals;
    address reserveAddress;
    int256 reserveInitialAmount;
...
```

There are no input validation checks when the values pass in. The type casting from integer to unsigned integer is performed in the most inner helper function `_validationReserveInitialAmount()`. Inside `_validationReserveInitialAmount()`, the input parameters `reserveInitialAmount`, `tokenDecimals` and `tokenInitialSupply` are explicitly casted to unsigned integer types.

```
uint256 initialReserve = uint256(reserveInitialAmount);
uint32 _tokenDecimals = uint32(tokenDecimals);
uint256 _tokenInitialSupply = uint256(uint64(tokenInitialSupply));
```

`HederaTokenManager.initialize()`

Referring to documentation(https://docs.hedera.com/hedera/sdks-and-apis/sdks/smart-contracts/hedera-service-solidity-libraries), the input parameter types of the function `createFungibleToken()` should be `IHederaTokenService.HederaToken memory`, `uint` and `uint`. However, the struct `InitializeStruct` defines its fields `initialTotalSupply` to be `int64` and `tokenDecimals` to be `int32`.

```
    (int64 responseCode, address createdTokenAddress) = IHederaTokenService(
        _PRECOMPILED_ADDRESS
    ).createFungibleToken{value: msg.value}(
        init.token,
        init.initialTotalSupply,
        init.tokenDecimals
    );
```

```
    struct InitializeStruct {
        IHederaTokenService.HederaToken token;
        int64 initialTotalSupply;
        int32 tokenDecimals;
...
```

Recommendation

Recommend reviewing the design, fixing the inappropriate data type definitions and properly testing the contracts to ensure compatibility for current core logics with current codebase and other dependencies.

Alleviation

[Swirlds Labs]:

In the case of the fields belonging to the struct TokenStruct, these fields have, as a target, to be the values of fields belonging to other structs declared in the IHederaTokenService contract, that have the same types, which is a Hedera contract that is not under our control.

In the second case, the createFungibleToken function into the IHederaTokenService has the following declaration:

function createFungibleToken(

```
IHederaTokenService.HederaToken memory token,

int64 initialTotalSupply,

int32 decimals)
```

So second and third parameters are integers with sign.

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

HRC-01 | DECIMALS TOO SMALL

Category	Severity	Location	Status
Logical Issue	Minor	contracts/HederaReserve.sol: 10	Acknowledged

Description

The token's decimal is set too small, which could result in much loss in circulation.

```
uint8 private constant _DECIMALS = 2;
```

Recommendation

Consider ensuring that the loss due to accuracy suffered users is within tolerable limits.

Alleviation

[Swirlds Labs]:

The HederaReserve contract is used to simulate the backing of the stable coin through fiat money, so we decided, according to the client, to use 2 decimals, like most fiat money.

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

HTM-03 | PULL-OVER-PUSH PATTERN

Category	Severity	Location	Status
Logical Issue	Minor	contracts/HederaTokenManager.sol: 242~243	Acknowledged

Description

In the contracts `HederaTokenManager` the `initialize()` function can receive HBARs. The HBARs are used to create fungible token through the `IHederaTokenService` precompiled contract. Then if there are any leftover HBARs, the extras will be sent back to the `init.originalSender` by calling the `_transferFundsBackToOriginalSender()` function.

The smart contract contains low-level call `.call()`. Since these calls bypass some of the automatic checks that Solidity provides, like function type checks, they can introduce vulnerabilities, logic errors, or unexpected behavior.

Recommendation

Recommend not refunding extra HBARs in the way of directly sending back to the `init.originalSender` address, referring to the Pull over Push Pattern.

Alleviation

[Swirlds Labs]:

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

RCP-04 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/extensions/Reserve.sol: 61	Resolved

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.

```
        _reserveAddress = newAddress;
```

- `newAddress` 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

The team acknowledged this issue and they stated this is by design:

" `updateReserveAddress()` can receive the zero address since this is the way to remove a reserve contract from a stable coin. The reserve address can be set to 0 if proof of reserve is not required."

SAC-01 | MISSING VALIDATIONS WHEN INCREASE AND DECREASE SUPPLIER ALLOWANCE

Category	Severity	Location	Status
Inconsistency, Logical Issue	Minor	contracts/extensions/SupplierAdmin.sol: 132~175	Resolved

Description

According to the code comments, the functions `increaseSupplierAllowance()` and `decreaseSupplierAllowance()` should validate that if the address account `supplier` isn't unlimited supplier's allowance. However, these two functions do not validate the accounts and directly add/substract the `_supplierAllowances` .

Recommendation

We recommend the team adding the neccessary validations.

Alleviation

The team heeded our advice and resolved the issue in commit 5dbe8450dfd835b4d34743e6644b3930f434c8fd.

CON-05 | INCONSISTENT SOLIDITY VERSIONS

Category	Severity	Location	Status
Language Version	● Informational	contracts/extensions/Interfaces/IWipeable.sol; contracts/hts-precompile/HederaResponseCodes.sol; contracts/hts-precompile/IHederaTokenService.sol	● Acknowledged

Description

The codebase contains multiple Solidity versions, which can lead to unexpected behavior, potential vulnerabilities, difficulties in maintaining the code, and inconsistencies in the execution of the smart contract. Using different versions may also result in increased complexity during code auditing, as different security features and bug fixes are present in different versions of the compiler.

Versions used: `0.8.16`, `>=0.4.9<0.9.0`

Other directives used: `ABIEncoderV2`

`0.8.16` is used in projects/hedera-accelerator-stablecoin/contracts/contracts/extensions/Interfaces/IWipeable.sol file.

```
2 pragma solidity 0.8.16;
```

`>=0.4.9<0.9.0` is used in projects/hedera-accelerator-stablecoin/contracts/contracts/hts-precompile/IHederaTokenService.sol file.

```
2 pragma solidity >=0.4.9 <0.9.0;
```

`ABIEncoderV2` is used in projects/hedera-accelerator-stablecoin/contracts/contracts/hts-precompile/IHederaTokenService.sol file.

```
3 pragma experimental ABIEncoderV2;
```

Recommendation

It is recommended to standardize on a single, up-to-date Solidity version throughout the codebase to ensure consistent behavior, benefit from the latest security features, and improve maintainability.

Alleviation

[Swirlds Labs]:

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

Different solidity version in imported libraries from Hedera source code.

CON-07 | REDUNDANT CODE COMPONENTS

Category	Severity	Location	Status
Volatile Code	Informational	contracts/HederaReserve.sol: 113~127; contracts/extensions/Reserve.sol: 33	Partially Resolved

Description

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation

Recommend removing the redundant statements for production environments and finishing the implementation when the project is about to be test the production stage or launch.

Alleviation

[Swirlds Labs]: Issue acknowledged. I won't make any changes for the current version.

The getRoundData of HederaReserve contract cannot be removed since HederaReserve implements Chainlink AggregatorV3Interface contract, so we need to implement this function because HederaReserve contract cannot be an abstract contract as it must be deployed.

[CertiK]:

The team heeded our advice and resolved the issue in commit 5dbe8450dfd835b4d34743e6644b3930f434c8fd.

The function `HederaReserve.getRoundData()` still implemented as `revert('Not implemented')`.

[Swirlds Labs]: Issue acknowledged. I won't make any changes for the current version.

The Chainlink AggregatorV3Interface contract is a mock contract that helps us to demonstrate how this will work using an oracle.

CON-08 | INCORRECT VARIABLE DATA TYPES

Category	Severity	Location	Status
Logical Issue	Informational	contracts/HederaReserve.sol: 13; contracts/StableCoinFactory.sol: 34; contracts/hts-precompile/IHederaTokenService.sol: 69	Acknowledged

Description

```
    int64 private constant _DFAULT_AUTO_RENEW_PERIOD = 90 days;
```

The state variable `_DFAULT_AUTO_RENEW_PERIOD` is an unit64, but it is declared as int64.

```
    int256 private _reserveAmount;
```

The state variable `_reserveAmount` should never be negative, but it is declared as int256 and allows negative number.

`HederaReserve.setAmount()`

Both the input parameter (`newValue`) and the state variable to be assigned (`_reserveAmount`) seem to belong to unsigned integer types.

```
function setAmount(int256 newValue) external isAdmin {
    emit AmountChanged(_reserveAmount, newValue);
    _reserveAmount = newValue;
}
```

```
    int64 autoRenewPeriod;
```

The state variable `autoRenewPeriod` should never be negative, but it is declared as int64 and allows negative number.

Recommendation

We recommend the team use the correct data type to declare state variables.

Alleviation

[Swirlds Labs]:

In the case of _DEFAULT_AUTO_RENEW_PERIOD constant, its target is to populate the autoRenewPeriod property, which has int64 type, of the Expiry struct of the IHederaTokenService contract wich is a Hedera contract that is not under our control, while the _reserveAmount state variable is used to be the answer return value of the Chainlink latestRoundData function, which also has int256 type.

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

CON-11 | INFORMATION ABOUT `generateKey()`

Category	Severity	Location	Status
Logical Issue	Informational	contracts/HederaTokenManager.sol: 193~202; contracts/library/KeysLib.sol: 24	Acknowledged

Description

In the function `generateKey()` , the two fields `inheritAccountKey` and `contractId` of the `KeyValue` are not used. According to the code comments, the `inheritAccountKey` means "if set to true, the key of the calling Hedera account will be inherited as the token key". The field `contractId` means "smart contract instance that is authorized as if it had signed with a key".

Furthermore, after the `hederaKeys` is generated, the value is passed to the `hederaToken` . It is not used anywhere in the audit contracts but used in the out-of-scope dependant repo.

```
hederaToken = _updateHederaTokenInfo(
            updatedToken,
            hederaKeys,
            currentTokenAddress
        );
int64 responseCode = IHederaTokenService(_PRECOMPILED_ADDRESS)
            .updateTokenInfo(currentTokenAddress, hederaToken);
```

The same case is applied to the Expiry.

Recommendation

We would like to confirm with the client whether this implementation aligns with the project design.

Alleviation

[Swirlds Labs]:

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

Both KeyValue and HederaToken structs are declared in the IHederaTokenService interface which is a Hedera contract not developed for the accelerator.

HTM-02 | UNUSED RETURN VARIABLE

Category	Severity	Location	Status
Logical Issue	Informational	contracts/HederaTokenManager.sol: 324	Resolved

Description

The function `_updateHederaTokenInfo()` declares a return variable `IHederaTokenService.HederaToken memory hederaTokenUpdated`. However, this pre-declared variable is never written or used. The local variable `hederaTokenInfo` is returned instead.

```
    function _updateHederaTokenInfo(
        UpdateTokenStruct calldata updatedToken,
        IHederaTokenService.TokenKey[] memory hederaKeys,
        address currentTokenAddress
    )
        private
        returns (IHederaTokenService.HederaToken memory hederaTokenUpdated)
    {
        ...

        return hederaTokenInfo;
    }
```

Recommendation

It is recommended to assign return variables or write explicit return statements to avoid implicitly returning default values. Also, if there are local variables duplicating named return variables, we recommend removing the local variables and use the return variables instead.

Alleviation

The team heeded our advice and resolved the issue in commit 5dbe8450dfd835b4d34743e6644b3930f434c8fd.

HTM-04 | INFORMATION ABOUT `_hederaTokenManagerAddress`

Category	Severity	Location	Status
Logical Issue	Informational	contracts/HederaTokenManager.sol	Acknowledged

Description

The state variable `_hederaTokenManagerAddress` in the contract `StableCoinFactory` is managed by the admin, who can add/edit/remove the `_hederaTokenManagerAddress`. However, the `_hederaTokenManagerAddress` is not used anywhere except in the view function `getHederaTokenManagerAddress()`.

Recommendation

We would like the team to elaborate more about the usage of the `_hederaTokenManagerAddress`.

Alleviation

[Swirlds Labs]:

`_hederaTokenManagerAddress` variable is used to store different versions, by its address, of HederaTokenManager contracts, so the user creating the stable coin can be reported about all versions that can be used to create the stable coin through this factory.

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

OPTIMIZATIONS | STABLECOIN STUDIO

ID	Title	Category	Severity	Status
CON-06	Unused State Variable	Coding Issue	Optimization	Acknowledged

CON-06 | UNUSED STATE VARIABLE

Category	Severity	Location	Status
Coding Issue	Optimization	contracts/StableCoinFactory.sol: 30; contracts/hts-precompile/HederaResponseCodes.sol: 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 29, 30, 31, 32, 33, 35, 36, 37, 38, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 52, 53, 55, 56, 58, 59, 60, 61, 62, 63, 64, 66, 67, 69, 70, 71, 72, 74, 75, 76, 77, 78, 80, 81, 82, 83, 84, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 132, 133, 134, 135, 136, 137, 138, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223~224, 225, 226~227, 228~230, 231, 232, 233, 234, 235, 236, 237, 238~240, 241, 242, 243~244, 245~246, 247~248, 249, 250~252, 253, 254~255, 256~258, 259, 260, 261, 262, 263, 264, 265, 266~267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299~300, 301~302, 303, 304~306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324~325, 326, 327, 328, 329	Acknowledged

Description

Some state variables are not used in the codebase. This can lead to incomplete functionality or potential vulnerabilities if these variables are expected to be utilized.

Recommendation

It is recommended to ensure that all necessary state variables are used, and remove redundant variables.

Alleviation

[Swirlds Labs]:

HederaResponseCodes contract is a Hedera contract but we don't need most of their response codes.

APPENDIX | STABLECOIN STUDIO

Finding Categories

Categories	Description
Language Version	Language Version findings indicate that the code uses certain compiler versions or language features with known security issues.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
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.

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.

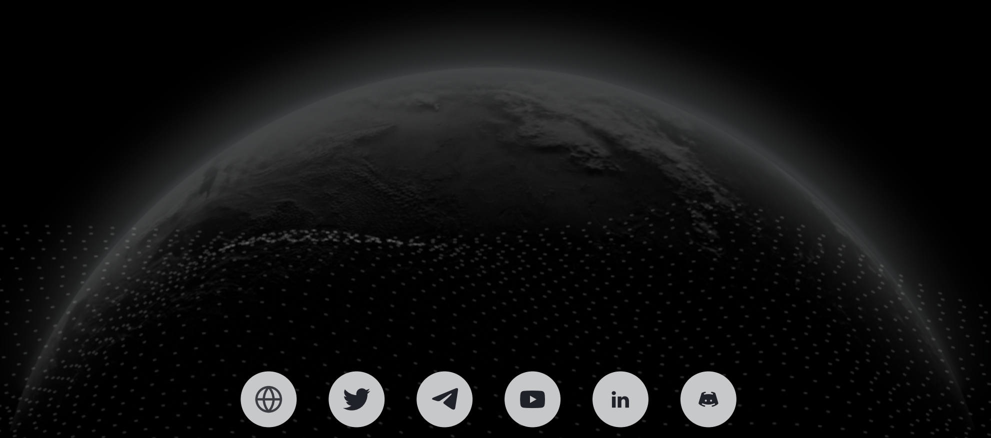
NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND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



Copyright © CertiK