Lydia Coin - Audit

Security Assessment

CertiK Assessed on Oct 10th, 2025

CertiK Assessed on Oct 10th, 2025

Lydia Coin - Audit

The security assessment was prepared by CertiK.

Executive Summary

TYPES	ECOSYSTEM	METHODS
ERC-20	Base Blockchain	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE
Solidity	Preliminary comments published on 09/09/2025 Final report published on 10/11/2025

Vulnerability Summary

20	16	0	4	0
Total Findings	Resolved	Partially Resolved	Acknowledged	Declined

2	Centralization	2 Acknowledged	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
1	Major	1 Resolved	Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
6	Medium	5 Resolved, 1 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
2	Minor	2 Resolved	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.
9	Informational	8 Resolved, 1 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 | LYDIA COIN - AUDIT

CODEBASE | LYDIA COIN - AUDIT

Repository

USAD :

- Proxy: 0x61f3Bf9f875368Fb919cB7C3Bbf15f1a29784d1F
- Implementation: 0x7162659EDcCabce2977B9A234479213821a49989

TOKEN :

- Proxy: 0x5614665d745e7cc1424bdefefc774f57fc549c6c
- Implementation: 0x7be88859404db3d0308af840da463a09f19fb1e2

DISTRIBUTION :

- Proxy: 0x6Ceb561c0303cE0B23Fc3066867DD2a6E3539A7B
- Implementation: 0xc9120c7398C2eE4e33Ad56e369F192f93e4C9634

VAULT :

- Proxy: 0x5614665d745E7cc1424BDEfEfc774f57FC549c6C
- Implementation: 0x7bE88859404db3D0308Af840Da463a09F19FB1E2

Commit

040e29826f104cdf00a26cef7a76bec0d9570666

4b8d1fd65bf3464e661f2c4100f3472737c63581

580cc01ae3cb3a4e2131ddb60f83a66a9302a499

AUDIT SCOPE | LYDIA COIN - AUDIT

CertiKProject/certik-audit-projects
DISTRIBUTION/TokenDistribution.sol
TOKEN/LydiaToken.sol
USAD/LydiaStablecoin.sol
VAULT/Vault.sol

APPROACH & METHODS | LYDIA COIN - AUDIT

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

The review process emphasized the following areas:

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

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

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

FINDINGS | LYDIA COIN - AUDIT

20	0	2	1	6	2	9
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared for Lydia Coin to identify potential vulnerabilities and security issues within the reviewed codebase. During the course of the audit, a total of 20 issues were identified. Leveraging a combination of Formal Verification, Manual Review & Static Analysis the following findings were uncovered:

ID	Title	Category	Severity	Status
LCA-06	**Initial Token Distribution**	**Centralization**	**Centralization**	**Acknowledged**
LCA-21	**Centralization Related Risks**	**Centralization**	**Centralization**	**Acknowledged**
LCA-09	Unrestricted `requestPauseUnpause` Overwrites Active Proposals	Denial of Service, Design Issue	Major	Resolved
LCA-07	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Medium	Resolved
LCA-08	Blacklisting Inconsistencies	Logical Issue, Inconsistency	Medium	Resolved
LCA-10	Voting Mechanism Issues	Design Issue	Medium	Resolved
LCA-22	Removed Signers' Prior Approvals Still Count Toward Quorum	Logical Issue	Medium	Acknowledged
LCA-29	Inconsistent State Of Pause Request	Logical Issue	Medium	Resolved
LCA-33	Cancellation And Expiration Of A Pause Request Do Not Reset `hasApprovedPause`	Inconsistency, Logical Issue	Medium	Resolved
LCA-12	Missing Zero Address Validation	Volatile Code	Minor	Resolved

ID	Title	Category	Severity	Status
LCA-13	Missing Check Of Duplicate Address And Empty Array	Volatile Code	Minor	Resolved
LCA-05	Information On Upgrade Handling	Coding Issue	Informational	Resolved
LCA-14	Inconsistent Solidity Versions	Language Version	Informational	Resolved
LCA-15	Contracts With Todos	Coding Issue	Informational	Resolved
LCA-16	Commented Code	Coding Style	Informational	Resolved
LCA-17	Inherited Contracts Not Initialized In Initializer	Logical Issue	Informational	Resolved
LCA-23	Unlocked Compiler Version	Coding Issue	Informational	Resolved
LCA-24	Redundant Event Emit In `burn` And `mint` Functions	Coding Issue	Informational	Resolved
LCA-25	Inconsistent Naming	Coding Style	Informational	Acknowledged
LCA-26	LydiaStablecoin Should Be ILydia, Which Should Be Implemented Only Once	Volatile Code	Informational	Resolved

LCA-06 | Initial Token Distribution

Category	Severity	Location	Status
Centralization	**Centralization**	**USAD/LydiaStablecoin.sol (StableCoin): 106~107**	**Acknowledged**

Description

All of the `LydiaStablecoin` tokens are sent to the `treasury` address at initialization. This is a centralization risk because the owner of the `treasury` address can distribute tokens without obtaining the consensus of the community. Any compromise to these addresses may allow a hacker to steal and sell tokens on the market, resulting in severe damage to the project.

Recommendation

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

==========For Preliminary Report Only==========

In order for CertiK to update the status of this finding during the remediation phase, please kindly provide the URL to the published token distribution plan and the multi-signature wallet address that holds the undistributed tokens. We will verify the information and update the report. Thank you.

Link to the token distribution plan: https://www…

Multi-sig wallet address: 0x…

Signer 1: 0x…

Signer 2: 0x…

Signer 3: 0x…

Alleviation

[CertiK, 09/23/2025]: The team acknowledged the issue and decided not to implement the recommended change in the current engagement.

LCA-21 | Centralization Related Risks

Category	Severity	Location	Status
Centralization	Centralization	DISTRIBUTION/TokenDistribution.sol (StableCoin): 76, 97, 124, 141, 154, 161, 169; TOKEN/LydiaToken.sol (StableCoin): 72, 85, 94, 118; USAD/LydiaStablecoin.sol (StableCoin): 135, 161, 183, 192, 202, 211, 226, 244, 291; VAULT/Vault.sol (StableCoin): 148, 156, 170, 198, 213, 238, 289, 319, 339, 355, 385, 403	Acknowledged

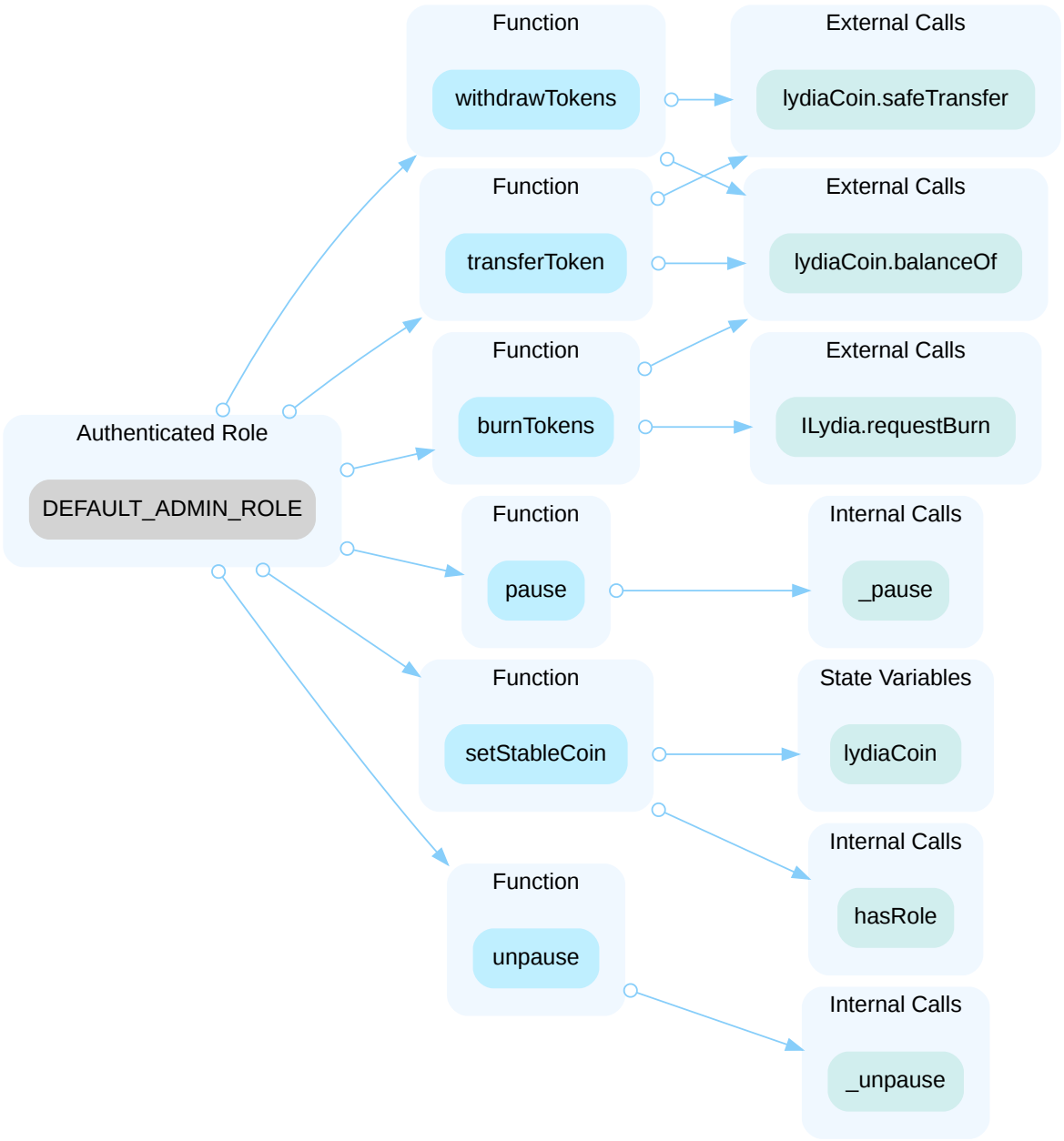
Description

LydiaTokenDistribution

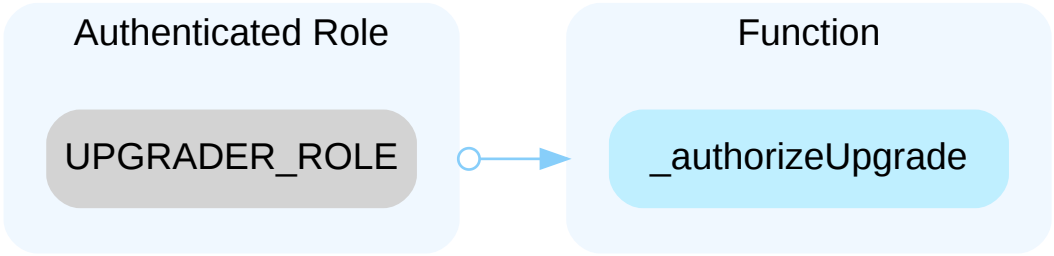
In the contract `LydiaTokenDistribution`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and do the following:

- Transfer all the funds to any address using `transferToken`
- Redeem all the funds using `burnTokens()`
- Steal all the funds using `wirthdrawTokens()`
- Change the underlying stable coin using `setStablecoin()`
- Change roles UPGRADER_ROLE using `grantRole()` and `revokeRole()` and assign them to themselves or another malicious entity.
- Or revoke ownership of UPGRADER_ROLE, making upgrade impossible.

.

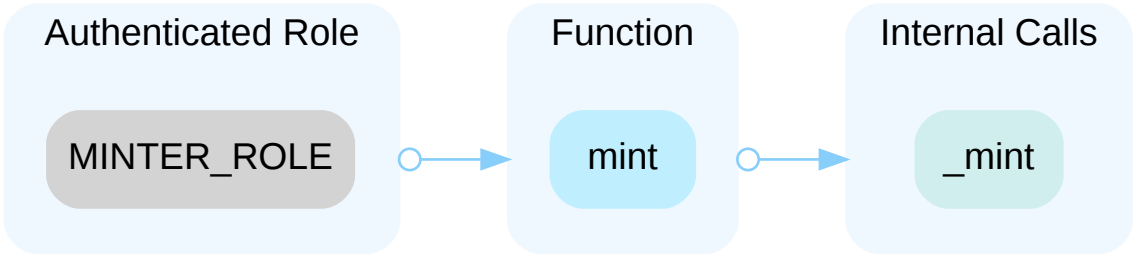


In the contract `LydiaTokenDistribution`, the role `UPGRADER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `UPGRADER_ROLE` account may allow the hacker to take advantage of this authority and exert control over upgrade operations.

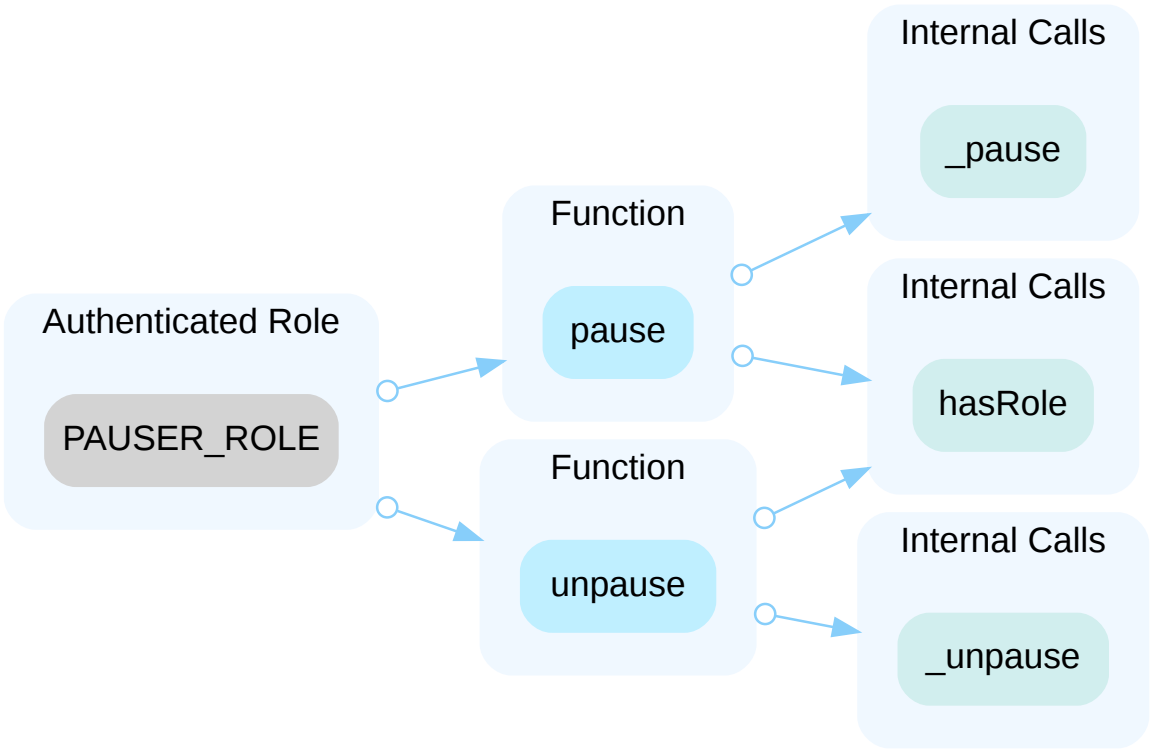


LydiaToken

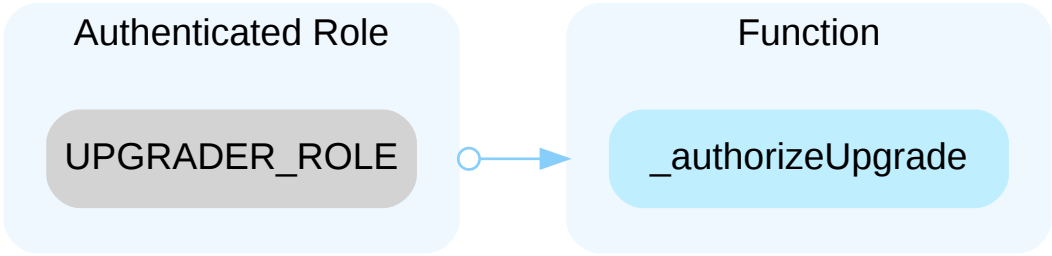
In the contract `LydiaToken` , the role `MINTER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `MINTER_ROLE` account may allow the hacker to take advantage of this authority and mint tokens to an address they control.



In the contract `LydiaToken` , the role `PAUSER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `PAUSER_ROLE` account may allow the hacker to take advantage of this authority and execute unauthorized pausing of the contract's functions.



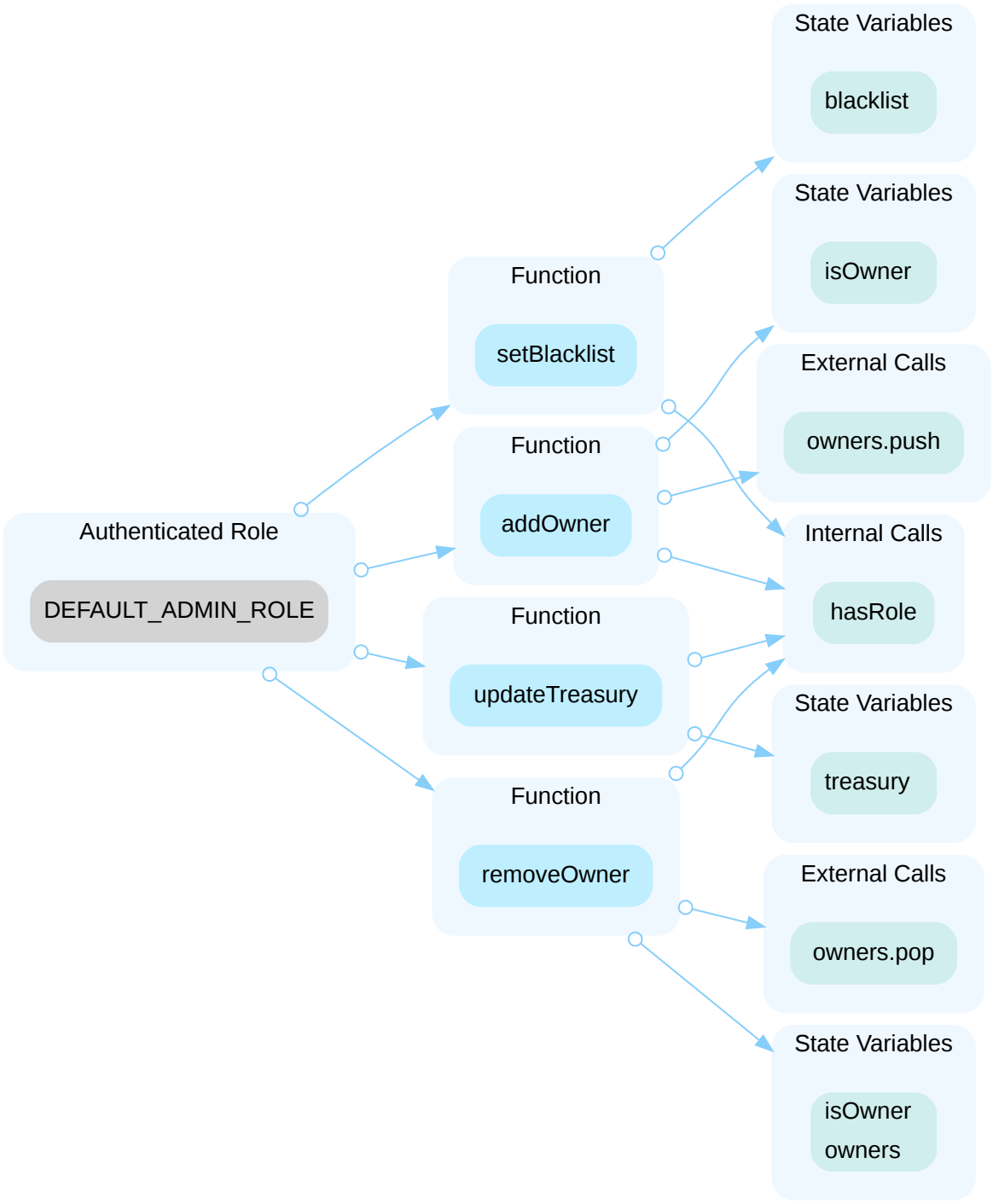
In the contract `LydiaToken` , the role `UPGRADER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `UPGRADER_ROLE` account may allow the hacker to take advantage of this authority and upgrade the contract to a malicious implementation.



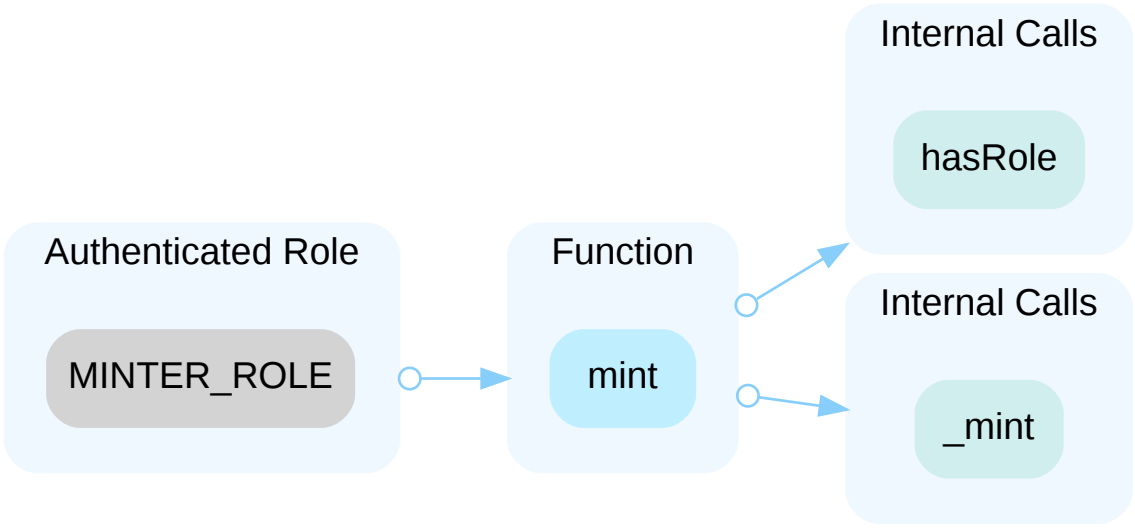
LydiaStablecoin

In the contract `LydiaStablecoin`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and do the following:

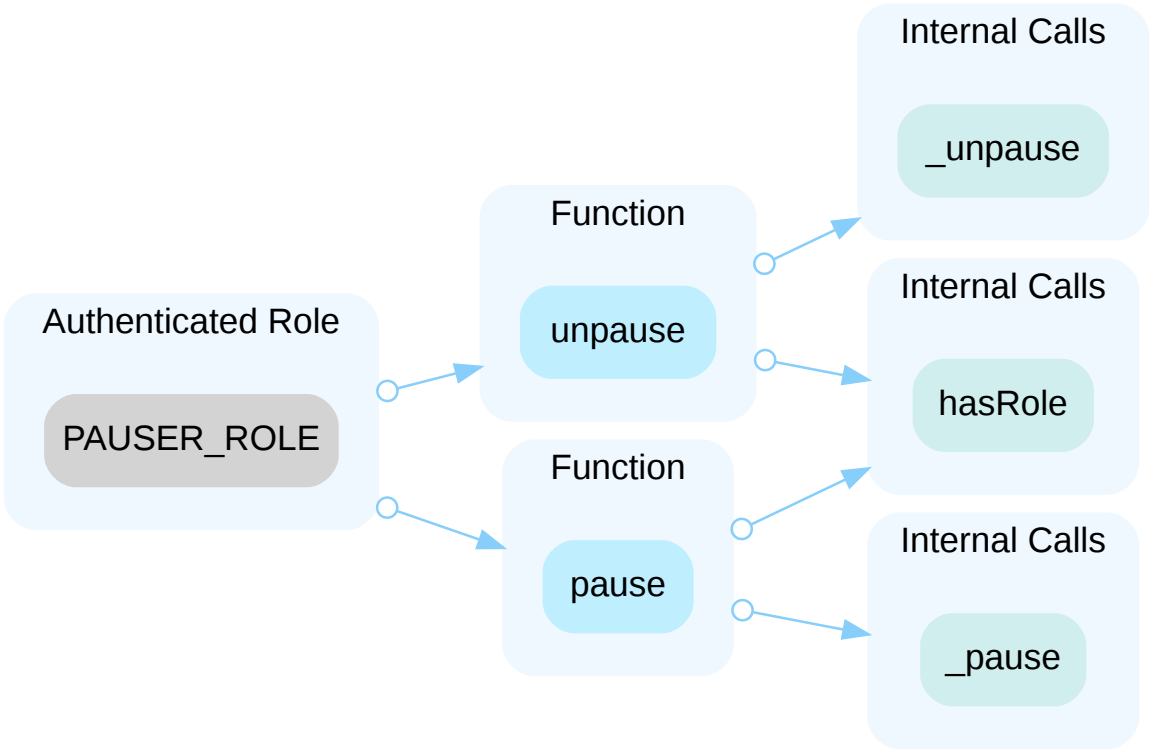
- Blacklist legitimate accounts, thus freezing their orders, and unblacklist compromised ones, allowing them to manipulate (legitimate or illegitimate) funds.
- Give control of the treasury to a illegitimate user.
- Remove legitimate owners (aka burners) and add illegitimate ones so that burning is entirely controlled by malicious entities.
- Change roles (i.e. MINTER_ROLE, PAUSER_ROLE and UPGRADER_ROLE) using `grantRole()` and `revokeeRole()` and assign them to themselves or another malicious entity.
- Or revoke ownership of UPGRADER_ROLE, making upgrade impossible.



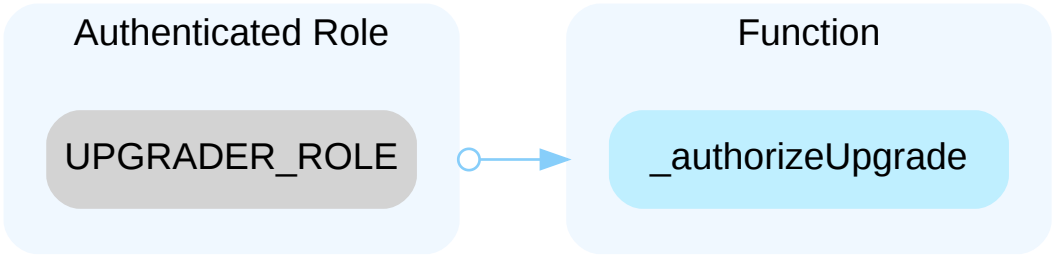
In the contract `LydiaStablecoin`, the role `MINTER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `MINTER_ROLE` account may allow the hacker to take advantage of this authority and mint new tokens, diluting their values.



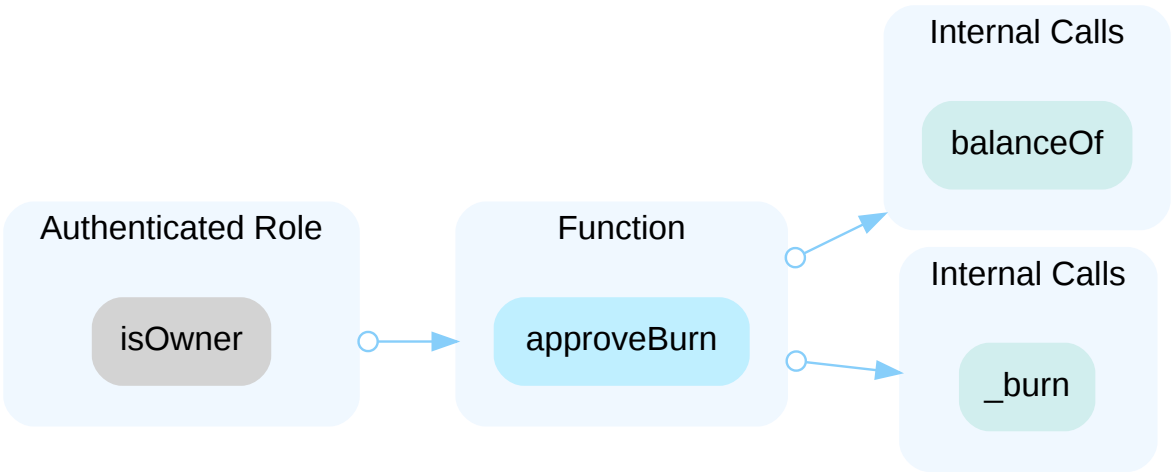
In the contract `LydiaStablecoin`, the role `PAUSER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `PAUSER_ROLE` account may allow the hacker to take advantage of this authority and pause certain functionalities of the contract, like minting for `treasury` or transferring.



In the contract `LydiaStablecoin`, the role `UPGRADER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `UPGRADER_ROLE` account may allow the hacker to take advantage of this authority and upgrade the logic contract toward a malicious implementation.



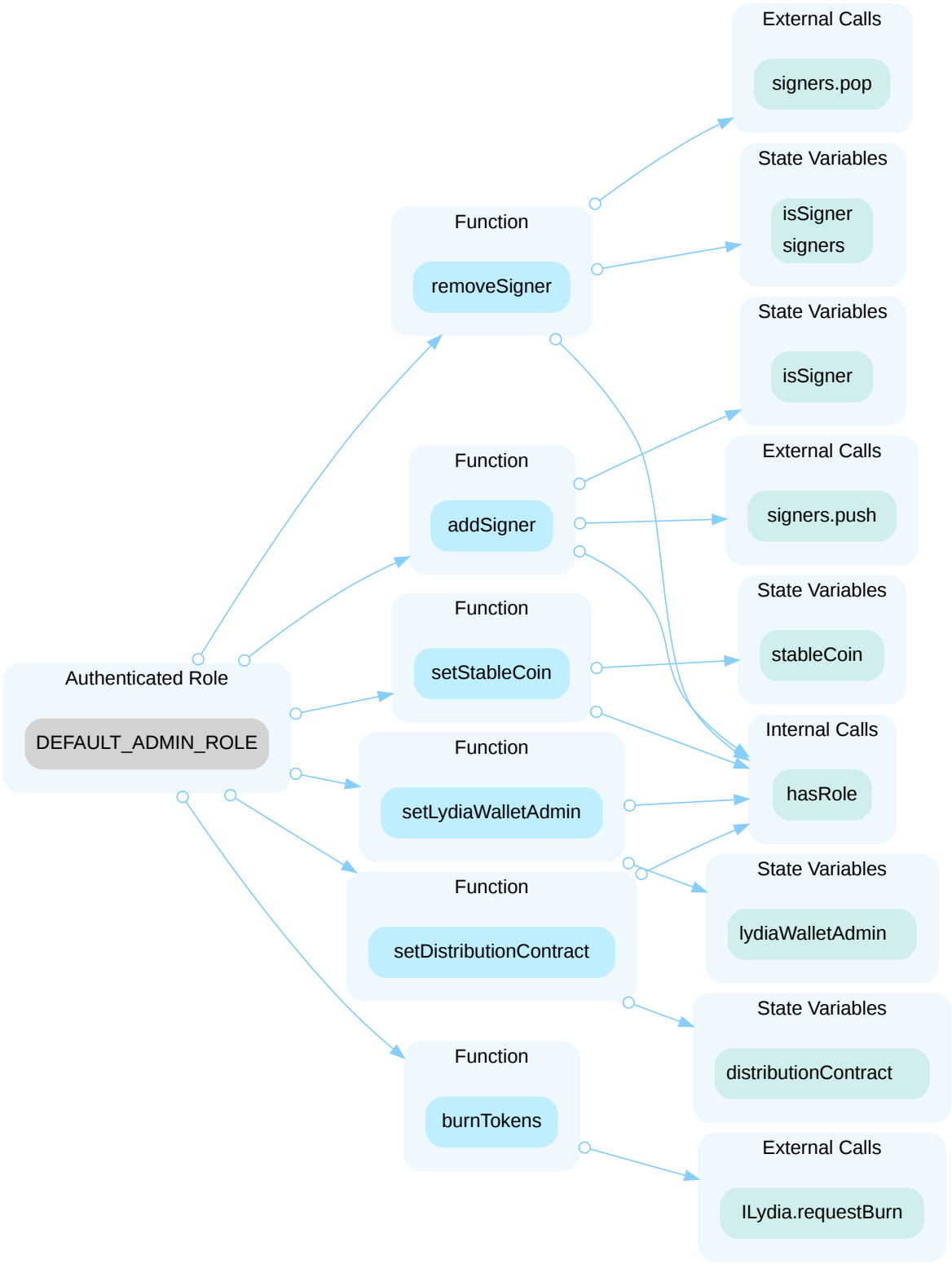
In the contract `LydiaStablecoin`, the role `isOwner` has authority over the functions shown in the diagram below. Any compromise to the `isOwner` account may allow the hacker to take advantage of this authority and approve burn requests.



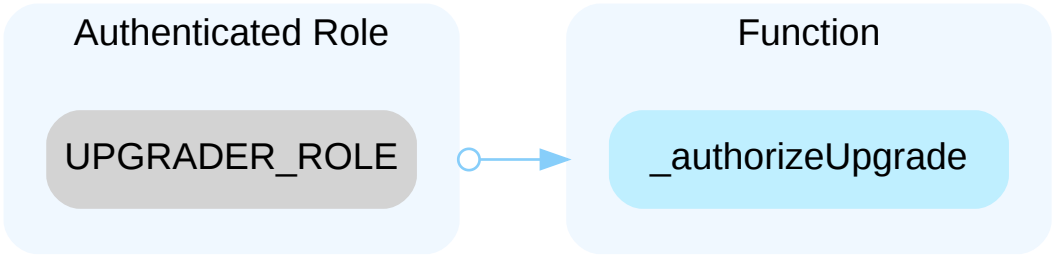
MultiSignatureTreasuryVault

In the contract `MultiSignatureTreasuryVault`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and utilize the associated privileges as follows:

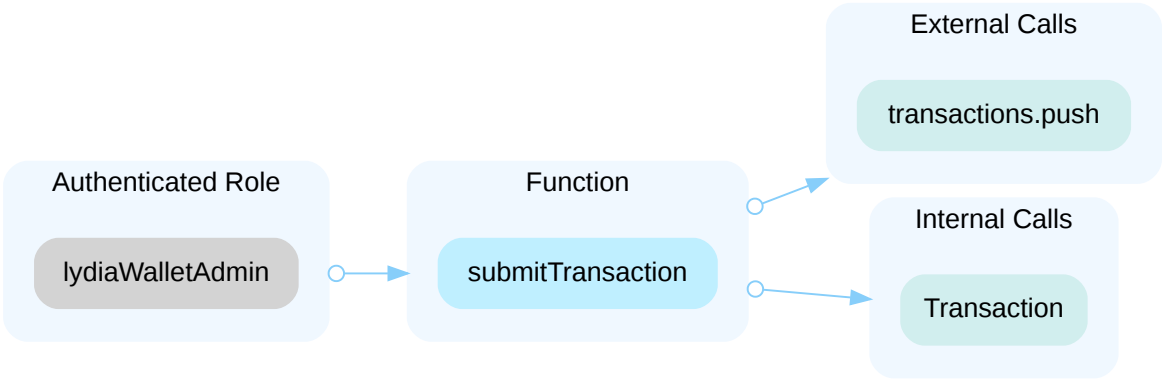
- Transfer the `lydiaWalletAdmin` role to themselves or another malicious entity.
- Add illegitimate signers (possibly only themselves by using multiple addresses), which could approve illegitimate transactions; or pause/unpause the contract.
- Remove legitimate signers to have total control over approved transactions and pause/unpause events.
- Change roles UPGRADER_ROLE using `grantRole()` and `revokeRole()` and assign it to themselves or another malicious entity.
- Or revoke ownership of UPGRADER_ROLE, making upgrade impossible.



In the contract `MultiSignatureTreasuryVault`, the role `UPGRADER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `UPGRADER_ROLE` account may allow the hacker to take advantage of this authority and allow the hacker to take advantage of this authority and upgrade the logic contract toward a malicious implementation.



In the contract `MultiSignatureTreasuryVault`, the role `lydiaWalletAdmin` has authority over the functions shown in the diagram below. Any compromise to the `lydiaWalletAdmin` account may allow the hacker to take advantage of this authority and submit malicious transactions (e.g., to bloat memory with submissions).



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

[Lydia, 10/08/2025]: We acknowledge this finding but have chosen to retain the existing structure as per our functional flow and use case. All role-owner private keys will be securely stored, and key management procedures will follow strict internal security standards to mitigate centralization risk.

LCA-09 Unrestricted `requestPauseUnpause` Overwrites Active Proposals

Category	Severity	Location	Status
Denial of Service, Design Issue	Major	VAULT/Vault.sol (StableCoin): 327~328	Resolved

Description

The `requestPauseUnpause` function can be called by any signer at any time, regardless of whether a pause/unpause proposal is already in progress. Each call overwrites the current `pauseRequest` object and resets approvals. `requestPauseUnpause` resets the `PauseRequest` struct (approvals = 0, executed = false) but does not clear the `hasApprovedPause` mapping that records which signers have already approved.
Since `approvePauseUnpause()` requires `!hasApprovedPause[msg.sender]`, any signer who approved the prior (now overwritten) request remains stuck as “already approved” for the new request and cannot approve again.
Because `pauseRequest.approvals` was reset to 0 while the main mechanism to clear `hasApprovedPause` (the loop in executePauseUnpause) is only reached after accumulating `requiredApprovals`, the new request might never reach the threshold if enough approvers have been neutralized.

This can potentially lead to creating a deadlock.

Recommendation

We recommend restricting `requestPauseUnpause` so that a new request cannot overwrite an active one.

Alleviation

[CertiK, 09/24/2025] The client applied the suggested change, thus resolving the finding in commit 040e29826f104cdf00a26cef7a76bec0d9570666.

LCA-07 | Unchecked ERC-20 `transfer()` / `transferFrom()` Call

Category	Severity	Location	Status
Volatile Code	Medium	VAULT/Vault.sol (StableCoin): 265~281	Resolved

Description

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

```
278         IERC20(transaction.token).transfer(transaction.to, transaction.amount);
```

Scenario

1. Prepare a non-compliant ERC20 token (attacker-controlled or an existing token with false-return semantics). Its transfer(address,uint256) returns false without reverting when called by the vault.
2. Ensure the vault holds a balance of this token (anyone can send ERC20 tokens to the vault address; no privileged role is needed for depositing tokens).
3. The signers follow their normal flow and approve the transaction for this token (submit/approve steps are routine and do not require attacker involvement). Once approvals >= requiredApprovals, executeTransaction(txId) is invoked internally.
4. During executeTransaction, transaction.executed is set to true, then transfer(...) is called. The token's transfer returns false but does not revert. Because the return value is not checked, the function does not revert and emits TransactionExecuted.
5. Post-conditions to verify the exploit:
    - transactions[txId].executed == true (permanently blocks re-execution of the same tx).
    - IERC20(token).balanceOf(address(this)) is unchanged (no tokens actually left the vault).
    - The recipient did not receive the tokens.

Recommendation

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

Alleviation

[CertiK, 09/23/2025]: The team heeded the advice and resolved the issue in commit

040e29826f104cdf00a26cef7a76bec0d9570666.

LCA-08 | Blacklisting Inconsistencies

Category	Severity	Location	Status
Logical Issue, Inconsistency	Medium	USAD/LydiaStablecoin.sol (StableCoin): 111	Resolved

Description

The contract `LydiaStablecoin` implements a blacklist mechanism restrict certain addresses from transferring their tokens. However, the blacklist is inconsistently enforced across the token's functionality:

- **Transfer vs. TransferFrom**: The `transfer` function checks for blacklist conditions, but `transferFrom()` does not. This allows blacklisted addresses to send tokens via allowances.
- **Owner Blacklist**: If an address is an `owner` of the contract, then if it is blacklisted, ownership privileges are retained until manually revoked. A blacklisted owner should automatically be prevented from continuing administrative operations.
- **Pending Burns**: If an address has requested a burn prior to being blacklisted, that request remains valid and can still be executed after the address is blacklisted.
- **Blacklisted addresses can receive tokens**: Even if a blacklisted address cannot send tokens directly, it can still receive newly minted tokens or tokens from other users.

Recommendation

We recommend:

1. Applying the blacklisting restriction consistently and automatically across all state-changing functions, including `transfer`, `transferFrom()`, `mint()`, `burn()`, and any other custom functions. These checks can be enforced with the `_update` function (without omitting `msg.sender` beside `from` and `to`).
2. When blacklisting the owner, explicitly revoke or restrict administrative privileges at the same time.
3. Invalidate or cancel pending operations for an address once it is blacklisted.
4. Enable decreasing allowances to blacklisted addresses, but prevent allowance increases.
5. Preventing the treasury from being blacklisted and preventing a blacklisted address from being set as the new treasury.

Alleviation

[CertiK, 09/23/2025]: The team removed the blacklisting mechanism in commit 040e29826f104cdf00a26cef7a76bec0d9570666.

LCA-10 | Voting Mechanism Issues

Category	Severity	Location	Status
Design Issue	Medium	VAULT/Vault.sol (StableCoin): 213~214	Resolved

Description

In `Vault`, the voting mechanism for transactions lacks lifecycle controls:

- **Spam risk**: Anyone can push unlimited useless proposals.
- **No cancellation**: Bad or obsolete proposals cannot be removed.
- **No deadline**: Proposals remain valid indefinitely and can be approved years later.

This could cause storage bloat, governance ambiguity, and stale execution risks.

Recommendation

We recommend:

- Enforcing deadlines for voting and execution.
- Adding a cancellation mechanism for bad or obsolete proposals.

Additionally, an anti-spam mechanism might be considered, for example, by requiring a proposal deposit, or enforcing proposer rate limits.

Alleviation

[CertiK, 09/23/2025] The client has added cancellation and expiry mechanisms in commit 040e29826f104cdf00a26cef7a76bec0d9570666, but several problems persist, for instance, there is no mechanism to remove transactions from `transactions`. Moreover, a compromised signer address could add as many transactions as they want. Thus, storage bloat is still enabled. We recommend that the client implement:

1. The proper removal of transactions in `executeTransaction()` and in `cancelTransaction()`
2. A function `clearExpiredTransaction()` which allows removing an unexecuted transaction after its deadline has been reached. Note that no such need exists for pause proposals, since there can only be one at the same time.

[CertiK, 10/01/2025]: The team heeded the advice and partially resolved the aforementioned risks in commit 4b8d1fd65bf3464e661f2c4100f3472737c63581, however, this leads to optimization consideration:

The `Transaction` struct defines two state flags: `executed` and `cancelled`. However, these flags are never updated during transaction processing anymore. Instead, transactions are removed entirely from storage using `delete transactions[txId]` when they are executed or cancelled.
The `executed` and `cancelled` fields are effectively unused.

We recommend removing the unused flags and modifiers to simplify the codebase and reduce the risk of misinterpretation.

[CertiK, 10/07/2025]: The team heeded the advice and resolved the issue in commit 580cc01ae3cb3a4e2131ddb60f83a66a9302a499.

LCA-22 | Removed Signers' Prior Approvals Still Count Toward Quorum

Category	Severity	Location	Status
Logical Issue	Medium	VAULT/Vault.sol (StableCoin): 403~404	Acknowledged

Description

`removeSigner()` only flips `isSigner[addr]` to `false` and removes the address from the signers array. It does not revoke or decrement any approvals that have already been cast for pending actions.
Specifically, it never clears `hasApproved[txId][signerToRemove]` or decrements `transactions[txId].approvals` for any pending transaction the signer had approved.

Approve paths rely on simple counters and do not re-check that counted approvals belong to current signers at execution time. Consequently, approvals from a removed signer still contribute to the quorum, allowing an action to execute with fewer approvals from the current signer set than `requiredApprovals` .

An analogous issue exists for pause/unpause requests: approvals are counted without an electorate snapshot, so removed signers' approvals remain counted and new signers can approve legacy requests, enabling outcome manipulation.

Recommendation

We recommend rewriting `removeSigner()` to ensure all approvals to pending transactions are deleted.

Alleviation

[Lydia, 10/08/2025]: We have retained the current logic since a proposal deadline mechanism is already implemented to prevent indefinite approvals. If a signer should no longer be allowed to approve, the signer can be safely removed from the list, and their approvals will no longer affect future proposals.

LCA-29 | Inconsistent State Of Pause Request

Category	Severity	Location	Status
Logical Issue	Medium	TreasuryVault.sol (Update1): 407, 512~513	Resolved

Description

In `TreasuryVault.requestPauseUnpause()`, when a pause request is created, `pauseRequest.approvals` is set to zero.

However, the same function considers that `pauseRequest.approvals == 0` should allow the creation of a new request:

```
406         require(
407             pauseRequest.executed || pauseRequest.cancelled || pauseRequest.
approvals == 0 ||
408             block.timestamp > pauseRequest.deadline,
409             "Active pause request exists"
410         );
```

and the function `isPauseRequestExpired()` considers that `pauseRequest.approvals == 0` means the request does not exist:

```
512         if (pauseRequest.approvals == 0) return false; // No request exists
```

This inconsistency allows any signer to overwrite a request when the approval is zero, potentially leading to front-running.

Scenario

1. A request to unpause the contract through `requestPauseUnpause()` has been created;
2. Bob wants to approve this request through `approvePauseUnpause()`;
3. Alice notices Bob wants to approve the function and front-run his transaction while calling `requestPauseUnpause()`;
4. Because the approval was zero, Alice has now replaced the request with her own, right before Bob approves this new request.

As a result, Bob has now approved a request different from the first one.

Recommendation

We recommend disallowing the overriding of any ongoing request with `pauseRequest.approvals == 0`.

Alleviation

[CertiK, 10/01/2025]: The team heeded the advice and partially resolved the issue in commit 4b8d1fd65bf3464e661f2c4100f3472737c63581.

The `isPauseRequestExpired()` function remains inconsistent, as it considers an executed, cancelled, or non-existent request as not expired.
We recommend the function revert if there is no pending transaction.

[CertiK, 10/07/2025] The team resolvedd the finding by making `isPauseRequestExpired()` revert as suggested, in commit 580cc01ae3cb3a4e2131ddb60f83a66a9302a499.

LCA-33 Cancellation And Expiration Of A Pause Request Do Not Reset `hasApprovedPause`

Category	Severity	Location	Status
Inconsistency, Logical Issue	Medium	TreasuryVault.sol (Update2): 480~481	Resolved

Description

`TreasuryVault.cancelPauseRequest()` allows the creator of the request of an admin user to cancel a pending request before expiration so a new proposal can be created.
However, this function does not reset the `hasApprovedPause` mapping, as `executePauseUnpause()` does. This can result in signers being unable to vote when a new proposal is created due to this flag.
The same issue exists when a new transaction is created over an expired one.

The voting power will be restored only if `executePauseUnpause()` is called, which risks being prevented if too many signers lose their voting ability.

Recommendation

We recommend adding a deletion of each signer's `hasApprovedPause` when a transaction is cancelled or expired.

Alleviation

[CertiK, 10/07/2025]: The team heeded the advice and resolved the issue in commit 580cc01ae3cb3a4e2131ddb60f83a66a9302a499.

LCA-12 | Missing Zero Address Validation

Category	Severity	Location	Status
Volatile Code	Minor	USAD/LydiaStablecoin.sol (StableCoin): 89, 90; VAULT/Vault.sol (StableCoin): 145	Resolved

Description

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

Recommendation

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

Alleviation

[CertiK, 09/23/2025]: The team heeded the advice and partially resolved the issue in commit 040e29826f104cdf00a26cef7a76bec0d9570666.
The `walletAdmin` parameter in `TreasuryVault.initialize()` can still be the zero address.

[CertiK, 10/01/2025]: The team heeded the advice and fully resolved the issue in commit 4b8d1fd65bf3464e661f2c4100f3472737c63581.

LCA-13 | Missing Check Of Duplicate Address And Empty Array

Category	Severity	Location	Status
Volatile Code	Minor	USAD/LydiaStablecoin.sol (StableCoin): 75~76; VAULT/Vault.sol (Stable Coin): 121	Resolved

Description

`Vault` and `LydiaStablecoin` contracts fail to validate the uniqueness of signer addresses during the set process. This lack of validation could allow for duplicate addresses within the signer array, potentially undermining the integrity of the multi-sig approval mechanism.

Additionally, no checks are in place to enforce that the input arrays are non-empty.

Recommendation

We recommend adding a check to prevent empty arrays and duplicate addresses in the provided `signers` and `owners` arrays.

Alleviation

[CertiK, 09/23/2025]: The team heeded the advice and partially resolved the issue in commit 040e29826f104cdf00a26cef7a76bec0d9570666.
Uniqueness is enforced, but the arrays provided can still be empty

[CertiK, 10/01/2025]: The team heeded the advice and fully resolved the issue in commit 4b8d1fd65bf3464e661f2c4100f3472737c63581.

LCA-05 | Information On Upgrade Handling

Category	Severity	Location	Status
Coding Issue	Informational		Resolved

Description

Contracts in the audit's scope are upgradeable. Please clarify whether the source code provided is for an upgrade of an existing deployment or whether this is the implementation code for a proxy being deployed for the first time. This information is necessary to determine whether storage collisions during upgrades should be considered.

Recommendation

If the currently audited codebase is an upgrade to an existing deployment, we recommend providing the address of the existing contract logic to assess the potential for storage collisions. Otherwise, please confirm this is the contract logic to be used with the first deployment of the project.

Alleviation

[Lydia Coin, 10/07/2025]: The contract would be freshly deployed. It is not upgraded over any existing contract.

LCA-14 | Inconsistent Solidity Versions

Category	Severity	Location	Status
Language Version	Informational	TOKEN/LydiaToken.sol (StableCoin): 2	Resolved

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: `^0.8.20`, `^0.8.22`, `^0.8.17`

```
4  pragma solidity ^0.8.20;
```

`^0.8.20` is used in node_modules/@openzeppelin/contracts/utils/math/SignedMath.sol file.

```
4  pragma solidity ^0.8.20;
```

```
4  pragma solidity ^0.8.22;
```

`^0.8.22` is used in node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol file.

```
4  pragma solidity ^0.8.22;
```

```
2  pragma solidity ^0.8.17;
```

`^0.8.17` is used in contracts/LydiaToken.sol file.

```
2  pragma solidity ^0.8.17;
```

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

[CertiK, 09/23/2025]: The team heeded the advice and resolved the issue in commit

040e29826f104cdf00a26cef7a76bec0d9570666.

LCA-15 | Contracts With Todos

Category	Severity	Location	Status
Coding Issue	Informational	VAULT/Vault.sol (StableCoin): 288	Resolved

Description

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

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

Recommendation

To mitigate this issue, it's important to:

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

Alleviation

[CertiK, 09/23/2025]: The team heeded the advice and resolved the issue in commit 040e29826f104cdf00a26cef7a76bec0d9570666.

LCA-16 | Commented Code

Category	Severity	Location	Status
Coding Style	Informational	DISTRIBUTION/TokenDistribution.sol (StableCoin): 30~31, 58~59, 65~66; VAULT/Vault.sol (StableCoin): 42~43, 66~67, 180~192, 295~299, 301~303	Resolved

Description

Commented code does not affect the functionality of the codebase and appears to be either remnants of test code or older functionality.

Recommendation

We recommend removing the redundant code to better prepare the code for production environments.

Alleviation

[CertiK, 09/23/2025]: The team heeded the advice and resolved the issue in commit 040e29826f104cdf00a26cef7a76bec0d9570666.

LCA-17 | Inherited Contracts Not Initialized In Initializer

Category	Severity	Location	Status
Logical Issue	Informational	DISTRIBUTION/TokenDistribution.sol (StableCoin): 26; TOKEN/LydiaToken.sol (StableCoin): 23; USAD/LydiaStablecoin.sol (StableCoin): 23; VAULT/Vault.sol (StableCoin): 25; FundReceiver.sol (Update1): 12; FundReceiver.sol (Update2): 13	Resolved

Description

The contracts extend `ReentrancyGuardUpgradeable`, but do not initialize it. Generally, the initializer function of a contract should always call all the initializer functions of the contracts that it extends.

Recommendation

We recommend initializing `ReentrancyGuardUpgradeable`.

Alleviation

[CertiK, 09/23/2025]: The team heeded the advice and resolved the issue in commit 040e29826f104cdf00a26cef7a76bec0d9570666.

LCA-23 | Unlocked Compiler Version

Category	Severity	Location	Status
Coding Issue	Informational	TOKEN/LydiaToken.sol (StableCoin): 2	Resolved

Description

- The contracts cited have an unlocked compiler version: ^0.8.17. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to ambiguity when debugging, as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.
- The contract LydiaToken cannot compile on solidity 0.8.1[789] since the `_update` function, which is explicitly overwritten, has been introduced by OpenZeppelin v4.9, which requires solidity ^0.8.20.

Recommendation

We recommend that the client lock the version of Solidity.

Alleviation

[CertiK, 09/23/2025]: The team heeded the advice and resolved the issue in commit 040e29826f104cdf00a26cef7a76bec0d9570666.

LCA-24 | Redundant Event Emit In `burn` And `mint` Functions

Category	Severity	Location	Status
Coding Issue	Informational	TOKEN/LydiaToken.sol (StableCoin): 65, 78; USAD/LydiaStablecoin.sol (StableCoin): 139, 176~177	Resolved

Description

LydiaToken.sol @TOKEN The contract explicitly overrides `_update`, so that it is compatible only with Solidity ^0.8.20. In turn, the `_burn` and `_mint` functions that `mint` and `burn` wrap make internal calls to `_update`, which already emit adequate events.

LydiaStablecoin.sol @USAD The same holds for `mint` and `approveBurn`.

Recommendation

We recommend that the client remove redundant, explicit `emit`s in `burn` and `mint`.

Alleviation

[CertiK, 09/23/2025]: The team heeded the advice and partially resolved the issue in the `LydiaStablecoin` contract (not in the other files) in commit 040e29826f104cdf00a26cef7a76bec0d9570666.
Note that the problem persists in all the contracts, `LydiaStablecoin` included, since other OpenZeppelin functions also emit events, such as `_transfer`.

[CertiK, 10/01/2025]: The team heeded the advice and fully resolved the issue in commit 4b8d1fd65bf3464e661f2c4100f3472737c63581.

LCA-25 | Inconsistent Naming

Category	Severity	Location	Status
Coding Style	Informational	USAD/LydiaStablecoin.sol (StableCoin): 162; VAULT/Vault.sol (StableCoin): 82	Acknowledged

Description

LydiaStablecoin.sol @USAD The only role of the `owners` seems to be able to approve `burn`s. Thus, for code clarity, the variable should be renamed `burners` and this change should be propagated to the appropriate events or structures.

Vault.sol The `onlySubmitter` modifier is associated with the state variable `lydiaWalletAdmin`, whereas other modifiers' names are transparent in other parts of the project. The `onlySubmitter` modifier should be renamed `OnlyLydiaWalletAdmin` or `OnlyWalletAdmin` for code consistency.

Recommendation

We recommend that the client apply the suggested name changes for code clarity and consistency.

Alleviation

[Lydia, 10/08/2025]: We acknowledge this naming inconsistency but chose to retain the current implementation for clarity within our system. Both signers and lydiaWalletAdmin can submit transactions; hence, the naming convention submitter has been intentionally kept generic to represent either role.

LCA-26 LydiaStablecoin Should Be ILydia, Which Should Be Implemented Only Once

Category	Severity	Location	Status
Volatile Code	Informational	USAD/LydiaStablecoin.sol (StableCoin): 16	Resolved

Description

- Contract `LydiaStablecoin` should *explicitly* enforce the `ILydia` interfaces of files `TokenDistribution.sol` and `Vault.sol`, for code clarity and consistency.
- Moreover, this interface should be implemented in only one file, with the adequate import.

Recommendation

We recommend that LydiaStablecoin enforce the ILydia interface, which should be implemented only once in the project.

Alleviation

[CertiK, 10/01/2025]: The team heeded the advice and resolved the issue in commit 4b8d1fd65bf3464e661f2c4100f3472737c63581.

OPTIMIZATIONS | LYDIA COIN - AUDIT

ID	Title	Category	Severity	Status
LCA-01	Cache Array Length	Coding Issue	Optimization	Resolved
LCA-02	Unused Error	Code Optimization	Optimization	Resolved
LCA-03	Lack Of `onlyRole` Modifier	Code Optimization	Optimization	Resolved
LCA-04	`nonReentrant` Modifier Can Be Placed And Executed Before Other Modifiers In Functions	Volatile Code, Code Optimization	Optimization	Resolved
LCA-18	Gas Optimizations	Gas Optimization	Optimization	Acknowledged
LCA-19	Custom Errors Can Be Used	Gas Optimization	Optimization	Acknowledged
LCA-20	Unneeded Check	Coding Issue, Gas Optimization	Optimization	Resolved

LCA-01 | Cache Array Length

Category	Severity	Location	Status
Coding Issue	Optimization	VAULT/Vault.sol (StableCoin): 375	Resolved

Description

The linked loop uses the length member of some storage array in its loop condition.

Recommendation

We recommend caching the lengths of storage arrays if they are used and not modified in for loops.

Alleviation

[CertiK, 09/23/2025]: The team heeded the advice and resolved the issue in commit 040e29826f104cdf00a26cef7a76bec0d9570666.

LCA-02 | Unused Error

Category	Severity	Location	Status
Code Optimization	Optimization	DISTRIBUTION/TokenDistribution.sol (StableCoin): 45~46	Resolved

Description

In `TokenDistribution`, the error `InvalidTokenType` is declared but never used.

```
45      error InvalidTokenType();
46      error InsufficientTokenBalance();
```

Recommendation

We recommend either using the error or removing it.

Alleviation

[CertiK, 09/23/2025]: The team heeded the advice and resolved the issue in commit 040e29826f104cdf00a26cef7a76bec0d9570666.

LCA-03 | Lack Of `onlyRole` Modifier

Category	Severity	Location	Status
Code Optimization	Optimization	DISTRIBUTION/TokenDistribution.sol (StableCoin): 142~143; TOKEN/LydiaToken.sol (StableCoin): 86~87, 95~96; USAD/LydiaStablecoin.sol (StableCoin): 136~137, 184~185, 193~194, 203~204, 212~213, 227~228, 245~246; VAULT/Vault.sol (StableCoin): 161, 175~176, 199~200, 386~387, 386~387	Resolved

Description

The `AccessControlUpgradeable` library provides the `onlyRole` modifier, which is optimized to efficiently restrict function access to accounts with a specific role.

Recommendation

We recommend using the `onlyRole` modifer to optimize code.

Alleviation

[CertiK, 09/23/2025]: The team resolved the issue only in the `LydiaStablecoin` contract in commit 040e29826f104cdf00a26cef7a76bec0d9570666. We recommend that the client search for the strings "hasRole" and "msg.sender" in their project to resolve the issue fully.

[CertiK, 10/01/2025]: The team heeded the advice and fully resolved the issue in commit 4b8d1fd65bf3464e661f2c4100f3472737c63581.

LCA-04 `nonReentrant` Modifier Can Be Placed And Executed Before Other Modifiers In Functions

Category	Severity	Location	Status
Volatile Code, Code Optimization	Optimization	DISTRIBUTION/TokenDistribution.sol (StableCoin): 76~79, 97~99, 124~126; TOKEN/LydiaToken.sol (StableCoin): 62, 72~75; VAULT/Vault.sol (StableCoin): 238~248, 289~292	Resolved

Description

- As a best practice, the `nonReentrant` modifier could be placed and executed before other modifiers in functions to prevent reentrancy attacks through other modifiers and make the code more efficient.
- In general, it is good practice to apply a consistent order for modifiers, e.g, to systematically put `onlyRole` modifiers at the end. This order currently often changes from one function to another.

Recommendation

Consider placing the `nonReentrant` modifier before the other modifiers in the following functions and applying a consistent order.

Alleviation

[CertiK, 09/23/2025]: The team partially resolved the issue in commit 040e29826f104cdf00a26cef7a76bec0d9570666.

[CertiK, 10/01/2025]: The team heeded the advice and fully resolved the issue in commit 4b8d1fd65bf3464e661f2c4100f3472737c63581.

LCA-18 | Gas Optimizations

Category	Severity	Location	Status
Gas Optimization	Optimization	TOKEN/LydiaToken.sol (StableCoin): 63, 76~77, 76~77	Acknowledged

Description

@TOKEN LydiaToken.sol

- `mint` and `burn` require > 0 amounts. It is often better practice do this checks off-chain: they do not compromise the token structure and systematic check will cost gas. You should clarify if this check is necessary.

Recommendation

We recommend that the client optimize gas consumption when security is not involved.

Alleviation

[Lydia, 10/08/2025]: We acknowledge this optimization suggestion but have intentionally retained on-chain validation checks to ensure data integrity and reliability. While minor gas savings could be achieved, they do not outweigh the importance of preserving strict runtime safety and validation on-chain.

LCA-19 | Custom Errors Can Be Used

Category	Severity	Location	Status
Gas Optimization	Optimization	TOKEN/LydiaToken.sol (StableCoin): 63, 86~87, 95~96	Acknowledged

Description

Custom errors have been introduced since Solidity version 0.8.4 and are less costly in gas than string error messages. Since the client is using ^0.8.17, they cannot use the pattern `require(condition, CustomError(foo))`. However, they can use the pattern `if !condition revert CustomError(foo)`, which will save gas during deployment.

This holds for all the places in the code where a string error message is used.

Recommendation

We recommend that the client systematically use custom errors in the project.

Alleviation

[Lydia, 09/23/2025]: We acknowledge the suggestion to use custom errors but decided to skip this optimization as it does not impact contract functionality or performance in any meaningful way. Given that the contract logic is relatively simple, the current error-handling approach is sufficient for maintainability and clarity.

LCA-20 | Unneeded Check

Category	Severity	Location	Status
Coding Issue, Gas Optimization	Optimization	VAULT/Vault.sol (StableCoin): 270~273	Resolved

Description

Vault.sol The internal function `executeTransaction()` is called only by function `approveTransaction()`, which performs the `transaction.approvals >= requiredApprovals`. Thus, `executeTransaction()` does not need perform the same check.

Recommendation

We recommend that the redundant check be removed.

Alleviation

[CertiK, 10/01/2025]: The team heeded the advice and resolved the issue in commit 4b8d1fd65bf3464e661f2c4100f3472737c63581.

APPENDIX | LYDIA COIN - AUDIT

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
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.
Denial of Service	Denial of Service findings indicate that an attacker may prevent the program from operating correctly or responding to legitimate requests.
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.

DISCLAIMER | CERTIK

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

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

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

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

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

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

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

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

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

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

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

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

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

Elevating Your Web3 Journey

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