Quantstamp Security Assessment Certificate

March 16th 2023 — Quantstamp Verified

Nayms

This audit report was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	Defi
Auditors	Andy Lin, Senior Auditing Engineer Valerian Callens, Senior Research Engineer Mostafa Yassin, Security Engineer Jeffrey Kam, Auditing Engineer
Timeline	2022-11-21 through 2022-12-05
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Technical document White paper
Documentation Quality	Medium
Test Quality	High

Source Code

Repository	Commit
nayms/contracts-v3	96adf68 initial audit

Total Issues	**30** (28 Resolved)
High Risk Issues	**1** (1 Resolved)
Medium Risk Issues	**3** (3 Resolved)
Low Risk Issues	**16** (16 Resolved)
Informational Risk Issues	**5** (3 Resolved)
Undetermined Risk Issues	**5** (5 Resolved)

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Fixed	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Nayms is a decentralized insurance marketplace built using the Diamond, Multi-facet Proxy architecture. The project uses three layers: facets, libraries, and storage. Facets are the entry point for the contract, libraries handle the main business logic, and all application data is stored in a single storage called AppStorage. The project is expected to undergo continuous development and may include unused storage slots or unclear logic, which could lead to unexpected behavior. This audit does not cover the Nayms token utility features. The audit scope includes `src/diamonds/nayms/*` and two ownership facets: `NaymsOwnershipFacet.sol` and `OwnershipFacet.sol` under `src/diamonds/shared/facets/`. We found several issues during the audit, and we recommend the team fix them.

During the audit, we identified several issues and recommended that the team implement tighter validations. The project's complexity, including multiple layers of calls and mapping storages, leads to higher risk without extra validations. We also suggest addressing any unclear specifications and improving documentation to prevent similar or new issues in the future.

Fix-review Update: the team has fixed or acknowledged all issues. The team provided atomic commits on the fixes for each issue, and the review for the fixes was straightforward!

ID	Description	Severity	Status
QSP-1	Draining Entity Asset by `payDividendFromEntity()`	High	Fixed
QSP-2	Overflow Blocking Offer Execution	Medium	Fixed
QSP-3	Policy Asset Can Be Different than the Entity	Medium	Fixed
QSP-4	System Admin Privileges Can Be Lost	Medium	Fixed
QSP-5	Risk of Calling `initialize()` Twice	Low	Fixed
QSP-6	Missing Validations	Low	Mitigated
QSP-7	Fulfilled Order Injecting Into Sorted List	Low	Fixed
QSP-8	`_getWithdrawableDividendAndDeductionMath()` Returning Unexpected `_dividendDeduction` Amount	Low	Fixed
QSP-9	Assigning Duplicated Stakeholders	Low	Fixed
QSP-10	Signature Replay Attack	Low	Fixed
QSP-11	`ReentrancyGuard` Colliding with Storage Slot	Low	Fixed
QSP-12	Risk of Insolvent Balance with Non-Standard Erc20 Token Integration	Low	Fixed
QSP-13	Invalid Policy Configurations	Low	Fixed
QSP-14	Incorrect Checks in `_assertValidOffer()`	Low	Fixed
QSP-15	Out-of-Gas Errors when Adding External Tokens to a Protocol	Low	Mitigated
QSP-16	Production Readiness Concern	Low	Fixed
QSP-17	Risk of Reentrancy	Low	Fixed
QSP-18	Accidental Role Revocation in `NaymsOwnershipFacet`	Low	Mitigated
QSP-19	Wrong Values in Emitted Event	Low	Fixed
QSP-20	Risk of `AppStorage` Mis-Ordering and Code Readability Concern	Low	Fixed
QSP-21	Not Supporting EIP-165	Informational	Fixed
QSP-22	Unlocked Pragma	Informational	Fixed
QSP-23	Dividend Denominations Can only Be Increased	Informational	Acknowledged
QSP-24	Missing Interface Extension in Contracts	Informational	Fixed
QSP-25	GTC Orders Risks	Informational	Acknowledged
QSP-26	Unclear Spec or Mimatched Documentation	Undetermined	Fixed
QSP-27	Unused Functions Lacking Validations	Undetermined	Fixed
QSP-28	Off-Chain Component Getting Non-Unique `guid`	Undetermined	Fixed
QSP-29	Updating an Entity Might Cause Inconsistencies	Undetermined	Fixed
QSP-30	Token Transfer Restrictions in `InternalTransfer()` May Be Bypassed in `_InternalTransfer()`	Undetermined	Fixed

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

DISCLAIMER:
If the final commit hash provided by the client contains features that are not within the scope of the audit or an associated fix review, those features are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.9.0

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither ./src/diamonds/nayms`

Findings

QSP-1 Draining Entity Asset by `payDividendFromEntity()`

Severity: *High Risk*

Status: Fixed

File(s) affected: `TokenizedVaultFacet.sol`, `LibTokenizedVault.sol`

Description: The `TokenizedVaultFacet.payDividendFromEntity()` can be called by anyone. The `LibTokenizedVault._payDividend()` function called under the hood would have the `_from` input as the `entityId`. Inside the `_payDividend()` function, it calls `_internalTransfer(_from, dividendBankId, _dividendTokenId, _amount)` to collect funds from the `_from` input (in the case here, the `entity`) to the virtual `dividendBankId`. The users can call this function to transform the assets of their entity to become dividends. Since anyone can call this, all entity members will be incentivized to immediately move all assets as dividends. The situation worsens as the `entity.asset` is supposed to be the same as the `simplePolicy.asset`. The attacker can immediately transform the funds for the insurance into a dividend.

Exploit Scenario:

1. The system manager starts a token sale of the entity with the call to the `EntityFacet.startTokenSale()` function. There is an offer selling the newly minted entity tokens for the entity asset tokens.

2. Alice bought all entity tokens with the entity asset.
3. Alice immediately calls the `payDividendFromEntity()` function that distributes all entity assets as dividends.
4. Since Alice is the only holder of the entity token, she gets all the dividends of the entity asset.
5. Alice now owns all the entity tokens and the original funds used to buy those tokens.
6. Even worse, since the `entity.asset` is the same as the `simplePolicy.asset`, when another entity pays the premium, Alice can call the `payDividendFromEntity()` to transfer those premiums as the dividend.

Recommendation:

1. Consider having a privileged role in calling the `TokenizedVaultFacet.payDividendFromEntity()` function. Otherwise, please embed the condition logic (e.g., when to pay dividends and how much) into the contract.
2. Consider adding validation that at least `entity.utilizedCapacity` (sum of the `simplePolicy.limit`) should be left on the `entity` after paying the dividend.

Update: The team fixed the issue in the commit `bc77e3d`:

1. Only the entity admin can pay the dividend after the change. New validation is added in the `TokenizedVaultFacet.payDividendFromEntity()` function.
2. To prevent transferring the `entity.utilizedCapacity`, the change syncs the amount of `utilizedCapacity` to the `AppStorage.lockedAmount`. The `LibTokenizedVault._internalTransfer()` function checks the locked tokens. The change provides a broader guard aside from the original suggestion to protect only the dividend scenario. After the change, the `simplePolicy.limit * entity.collateralRatio` will be locked on the simple policy creation. Note that there is a business logic change where the original code adds the whole `simplePolicy.limit` to the `utilizedCapacity`, and now only the `entity.collateralRatio` portion is added and locked.

QSP-2 Overflow Blocking Offer Execution

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `LibMarket.sol`

Description: The `LibMarket._matchToExistingOffers()` function can overflow on the following lines:

```
result.remainingSellAmount -= currentSellAmount
result.remainingBuyAmount -= currentBuyAmount;
```

It can overflow because the calculation logic on L199-214 for the `currentSellAmount` and the `currentBuyAmount` only takes care of one `result.remainingSellAmount` side or the `result.remainingBuyAmount` side. The other side is a derived amount, which can be larger than what is left in the remaining. Once the function overflows, it will revert and block the offer execution, although the trade should be valid.

From our analysis, overflow will only happen when the `buyExternalToken` is true. In the case of `buyExternalToken == false`, if the `currentSellAmount` is larger than the `result.remainingSellAmount`, it means it is trying to match a maker's offer that is worse than the taker's offer since the taker needs to sell more to get the same buying amount. However, in the opposite case, the overflow of the `remaingingBuyToken` means the maker's offer is better than the taker expects.

Exploit Scenario: A make offer that is "too good" can fail:

1. When buying 1 ETH with 2 pToken, the maker offers to sell 2 ETH with 2 pToken.
2. `buyExternalToken` is true in this case.
3. Since the maker offer is selling 2 ETH with 2 pToken, the `currentSellAmount` is 2 (pToken), and the `currentBuyAmount` is also 2 (ETH).
4. `result.remainingBuyAmount -= currentBuyAmount` will overflow as the `remainingBuyAmount` is 1 while the `currentBuyAmount` is 2.

Recommendation: When the `currentBuyAmount > result.remainingBuyAmount` sets `result.remainingBuyAmount` to zero.

Update: The team fixed the issue as recommended in the commit `046c639`.

QSP-3 Policy Asset Can Be Different than the Entity

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `LibEntity.sol`

Description: While creating a new `simplePolicy`, the function `_createSimplePolicy()` does not check that the policy `SimplePolicy.asset` variable is the same as `Entity.assetId`.
This means a policy can have a different asset than the entity.
It is also possible to update an entity's asset to become different than the rest of its policies through the function `_updateEntity()`.

Recommendation: Add a `require` statement to check that the asset for the policy is the same as the entity. Also, when updating an entity, either prevent asset changing or check for policies. If there are active policies, then asset change should not be possible.

Update: The team fixed the issue in the commit `4890c8d`. The asset is validated in the `LibEntity._validateSimplePolicyCreation()` function. Also, the `LibEntity._updateEntity()` function overrides the `s.entities[_entityId].assetId` to be the original one.

QSP-4 System Admin Privileges Can Be Lost

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `ACLFacet.sol`, `LibACL.sol`, `NaymsOwnershipFacet.sol`

Description: The highest privileges on the system are owned by the users in the group `GROUP_SYSTEM_ADMINS`, which are the users with the role `ROLE_SYSTEM_ADMIN`.
It is possible to revoke the role `ROLE_SYSTEM_ADMIN` from a user with the following methods:

1. `ACLFacet.assignRole()`: assign the admin a new role.

2. `ACLFacet.unassignRole()`: unassign the admin the role
3. `ACLFacet.updateRoleGroup()`: assign new groups to the `ROLE_SYSTEM_ADMIN`. However, only members of that group can call that function. Aside from that, additional roles can be added to the group `GROUP_SYSTEM_ADMINS` if a user with the role `ROLE_SYSTEM_ADMIN` calls the function `ACLFacet.updateRoleGroup()`.
4. `NaymsOwnershipFacet.transferOwnership()`: the contract owner can transfer ownership but cannot only revoke ownership. However, it can transfer ownership to the `0x0` address, effectively renouncing his contract ownership.

As the current number of users in the system having the role `ROLE_SYSTEM_ADMIN` is not tracked, it is possible to remain without any user having system admin privileges.

Recommendation:

1. Consider ensuring that the group `GROUP_SYSTEM_ADMINS` cannot remain without users in the `ACLFacet.assignRole()` and `ACLFacet.unassignRole()` functions.
2. Consider adding restrictions in the `ACLFacet.updateRoleGroup()` function that `ROLE_SYSTEM_ADMIN` and `GROUP_SYSTEM_ADMINS` cannot be changed.
3. Consider if other roles should be having the following privilege. Add restriction if they should not:
    1. assigned to the group `GROUP_SYSTEM_ADMINS`.
    2. allowed to assign roles to the group `GROUP_SYSTEM_ADMINS`.
4. Disallow `NaymsOwnershipFacet.transferOwnership()` with zero address.

Update: The team fixed the issue in the commit `9789f88`:

1. In the `ACLFacet` contract, both `assignRole()` and `unassignRole()` disallow changing role for the `LibConstants.SYSTEM_IDENTIFIER` context or the contract owner.
2. New validation added in the `ACLFacet.updateRoleGroup()` function to disallow updating the `GROUP_SYSTEM_ADMINS` role group.
3. Added validation in the `NaymsOwnershipFacet.transferOwnership()` function that the `_newOwner` cannot be zero address.

QSP-5 Risk of Calling `initialize()` Twice

Severity: *Low Risk*

Status: Fixed

File(s) affected: `InitDiamond.sol`

Description: The `InitDiamond.initialize()` might be triggered multiple times that can potentially reset the roles and configurations. If the contract owner calls the `diamondCut()` twice with the same `_init` and `_data` parameters, there is a risk that the system will accidentally be overridden.

Recommendation: Consider adding a control that the function `initialize()` can only be called once.

Update: The team fixed the issue as recommended in the commit `de25ad6`. A new storage variable `AppStorage.diamondInitialized` is used as the flag to revert the transaction of the `InitDiamond.initialize()` function.

QSP-6 Missing Validations

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `EntityFacet.sol`, `UserFacet.sol`, `LibACL.sol`, `LibAdmin.sol`, `LibEntity.sol`, `LibSimplePolicy.sol`, `LibMarket.sol`, `LibObject.sol`, `NaymsOwnershipFacet.sol`, `TokenizedVaultIOFacet.sol`, `LibFeeRouter.sol`, `LibObject.sol`

Description: We recommend adding validations to the Nayms system for several reasons. First, the system has multiple layers of function calls between facets, libraries, and storage. While libraries are often designed to be more generalized, facet functions may have more specific responsibilities within a smaller scope. In such cases, it is recommended to add validations to ensure the feature is only doing what the facet function is intended to support and nothing more. Second, Solidity provides empty default values for all variable types, which can create risks when getting struct data from a mapping. It will always provide empty struct data regardless of whether the key-value pair has been set. Finally, adding validations can help prevent accidental wrong inputs that could mutate the system.

1. `EntityFacet.enableEntityTokenization()`: should validate that only an "entity" can tokenize with this function (e.g., checking `s.existingEntities[_entityId] == true`). The current implementation allows tokenizing any object with this function which violates the function naming.
2. `LibEntity._createEntity()`: the fact that the `_entityAdmin` exists and that the `_entityID` does not already exist is not checked (e.g., checking `s.existingEntities[_entityId] == true`). Also, if given empty inputs, it can assign value to an empty key for the `s.existingEntities[]`, `s.entities[]`, `s.objectParent[]`, and `s.roles[]` mappings. The pattern is dangerous. For instance, it can get an unexpected entity with `s.entities[LibObject._getParent(_policyId)]` when the `_policyId` does not exist.
3. `LibEntity._updateEntity()`: the function should validate that `_entityId` is not empty. Otherwise, it can assign value to an empty key for the `s.entities[]` mapping. The pattern is dangerous. For instance, it can get an unexpected entity with `s.entities[LibObject._getParent(_policyId)]` when the `_policyId` does not exist.
4. `LibEntity._updateAllowSimplePolicy()`: the function should validate that `_entityId` is not empty.
5. `LibEntity._startTokenSale()`: the function should validate that `s.existingEntities[_entityId] == true` and that the `entity` has enabled tokenization (e.g., by calling `LibObject._isObjectTokenizable()`).
6. `LibEntity._createSimplePolicy()`:
    1. the function should validate that `_entityId` and `_policyId` are not empty bytes.
    2. the function should validate the fields of the `_stakeholders`. First, the `_stakeholders.roles.length` should be the same as the `_stakeholders.entityIds.length` (or the length of the `signatures`). Second, the `_stakeholders.entityIds[i]` should be the same as the `_entityId`. Third, depending on the business requirement, all the roles of the `_stakeholders.roles[i]` should probably be the same. Without the validations, it can set the policy role to another entity, and different stakeholders might have different roles.
    3. if the length of the parameter `_stakeholders` is too high, it can lead to an out-of-gas error in the for-loop.
7. `LibEntity.validateEntity()`: there is no check making sure that the `utilizedCapacity` of the entity is less than or equal to its `maxCapacity`.
8. `LibEntity. _validateSimplePolicyCreation()`: there is no min duration for the policy, no non-zero check for the `claimsPaid` and the `premiumPaids`, no min/max check for the `sponsorCommissionBasisPoints` and no check that the boolean attribute `canceled` is `false`.

9. `UserFacet.setEntity()`: the `_entityId` should be linked to an existing entity. Also, the function should check whether the user has already been attached to an entity. Otherwise, it can have unexpected outcomes, such as the user not being able to cancel orders for the previous entity anymore.
10. `LibSimplePolicy._payPremium()`: the function can add validation checking that the entity for `_payerEntityId` exists and the policy for `_policyId` exists. Though unlikely, the dummy `simplePolicy` returned as the default value from the mapping might pass through the function and have an unexpected effect.
11. `LibACL._assignRole()`: the `_objectId`, `_contextId`, and `_roleId` inputs should not be empty strings.
12. `LibACL._updateRoleAssigner()`: the `_role` and the `_assignerGroup` should not be empty. Specifically the `_role` input. It can set a value to the `s.canAssign[]` mapping for the empty key.
13. `LibACL._updateRoleGroup()`: the `_role` and `_group` inputs should not be empty strings. Otherwise, it can set the values for empty keys in the `s.groups` mapping. The pattern is dangerous because it can potentially break the `LibACL._canAssign()` function. The `assignerGroup` with the current implementation can be empty bytes when `_roleId` does not exist in the `s.canAssign[]` mapping. The `_isInGroup()` check might pass and falsely return true.
14. `LibAdmin._addSupportedExternalToken()`: the `_tokenAddress` input should not be empty.
15. `LibAdmin`: there is no non-zero address check for the parameter `_tokenAddress` in the functions `_addSupportedExternalToken()` and `_setDiscountToken()`.
16. `LibAdmin`: there is no min/max check for the parameters used in the functions `_setPoolFee()`, `_setTargetNaymsAllocation()`, `_setMaxDiscount()` and `_setEquilibriumLevel()`.
17. `TokenizedVaultIOFacet.externalDeposit()`: there is no early non-zero check for the parameter `_amount`.
18. `TokenizedVaultIOFacet.externalWithdrawFromEntity()`: there is no early non-zero check for the parameter `_amount`.
19. `LibFeeRouter._updatePolicyCommissionsBasisPoints()`: there is no min/max check for the parameters used (e.g., `bp.premiumCommissionNaymsLtdBP + bp.premiumCommissionNDFBP + bp.premiumCommissionSTMBP <= LibConstants.BP_FACTOR`).
20. `LibFeeRouter._updateTradingCommissionsBasisPoints()`: there is no min/max check for the parameter `bp.tradingCommissionTotalBP` (e.g., `bp.tradingCommissionTotalBP <= LibConstants.BP_FACTOR`).
21. `LibObject._enableObjectTokenization()`: there is no non-zero check for the length of the parameter `_symbol`.

Recommendation: Add validations as pointed out in the description section.

Update: The following is the fixed status for each suggestion in the description section. Most fixes are in the commit `454cb52`:

1. **fixed**: the team added the validation inside `LibObject._enableObjectTokenization()` instead.
2. **mitigated**: `_entityID` existence check is added. However, the validation for `_entityAdmin` is not in place.
3. **fixed**.
4. **fixed**: This function is removed in between the commits `96adf683..122a973`.
5. **fixed**.
6. **mitigated:** One sub-point fixed, one mitigated, and one acknowledged.
    1. **fixed**
    2. **mitigated**: from offline discussions, each stakeholder can have different roles. **Note:** the integrity of the `_stakeholders.entityIds[i] == _entityId` is not checked. A stakeholder of another entity can sign the policy without validation. However, according to the team, this seems to be the intended behavior.
    3. **ack**: the team replied with the following statement: "We currently do not bound the number of stakeholders a policy can have since the number is expected to be 4 or less."
7. **fixed**.
8. **mitigated**: validations for the `claimsPaid`, `premiumsPaid`, and `cancelled` are added. The min duration check for the policy and the min/max check for the `sponsorCommissionBasisPoints` are still missing.
9. **fixed**: the team added a validation that the entity must exist. The team also explained that whoever controls the entity should be able to cancel any orders the users have placed on the marketplace. Even if the entity is transferred to another user, the new user can cancel a market order that the previous entity owner has placed.
10. **fixed:** the team added existence check for `_payerEntityId` and `_policyId`.
11. **fixed**.
12. **fixed**.
13. **fixed**
14. **fixed**
15. **fixed**
16. **ack**: the team states that this is for token functionality, which the feature still needs to be set. The team will add proper validations once they start to develop token features.
17. **fixed**: the team added the check in the `LibTokenizedVaultIO._externalDeposit()` instead.
18. **fixed**: the team added the check in the `LibTokenizedVaultIO._externalWithdraw()` instead.
19. **fixed**
20. **unresolved**
21. **fixed**

QSP-7 Fulfilled Order Injecting Into Sorted List

Severity: *Low Risk*

Status: Fixed

File(s) affected: `LibMarket.sol`

Description: The logic handling dust order is inconsistent between the `_createOffer()`, `_takeOffer()`, and `_executeLimitOffer()` functions of the `LibMarket` contract. The dust condition in the `_createOffer()` function is `<= LibConstants.DUST`; but the condition in the `_takeOffer()` is `< LibConstants.DUST`. Also, in the `_executeLimitOffer()` function, the condition to inject into the sorted list is not an inverse of the dust condition used in the `_createOffer()` function. For instance, when the `remainingBuyAmount` and `remainingSellAmount` are non-zero and `<= LibConstants.DUST` (e.g., one) in the `_executeLimitOffer()` function, the offer will be flagged as

`LibConstants.OFFER_STATE_FULFILLED` on `_createOffer()#L247`. In contrast, the `_executeLimitOffer()` will continue to inject the already fulfilled order into the sorted list.

Recommendation:

1. Unify the condition between `_createOffer()` and `_takeOffer()` functions. Either go with `<=` or `<`.
2. Refactor the condition logic in `LibMarket._executeLimitOffer()#L420` from `if (result.remainingBuyAmount > 0 && result.remainingSellAmount > 0 && result.remainingSellAmount >= LibConstants.DUST)` to simply `s.offers[offerId].state == LibConstants.OFFER_STATE_ACTIVE`. So it does not have to duplicate the logic again.

Update: The team fixed the issue in the commit `20f96e1`:

1. The team goes with consistent `<`.
2. Change to use `s.offers[offerId].state == LibConstants.OFFER_STATE_ACTIVE` as the condition in the `LibMarket._executeLimitOffer()` function.

QSP-8 `_getWithdrawableDividendAndDeductionMath()` Returning Unexpected `_dividendDeduction` Amount

Severity: *Low Risk*

Status: Fixed

File(s) affected: `LibTokenizedVault.sol`

Description: The `LibTokenizedVault._getWithdrawableDividendAndDeductionMath()` can return a wrong `_dividendDeduction` value that adds one unexpectedly. Instead of checking against `_withdrawableDividend * _supply` on L276, it is supposed to check against `holderDividend * _supply`. Otherwise, whenever the `_withdrawnSoFar` is not zero, the `_dividendDeduction` would always be plus one.

Recommendation: Change the condition on L276 to `totalDividendTimesAmount > holderDividend * _supply`.

Update: The team simplified the `_getWithdrawableDividendAndDeductionMath()` function, and it no longer has the logic for `_dividendDeduction` in the commit `40240c6`.

QSP-9 Assigning Duplicated Stakeholders

Severity: *Low Risk*

Status: Fixed

File(s) affected: `LibEntities.sol`

Description: The `LibEntities._createSimplePolicy()` function assigns the stakeholders their roles. However, the stakeholders can be duplicated with the input. Thus, the for-loop might unexpectedly override the later role.

Exploit Scenario:

1. The system admin calls the function with Alice's signature twice. However, the `_stakeholders.roles[i]` are different.
2. Which role Alice is attached to depends on the order of the input.

Recommendation:

1. As a best practice, we recommend adding validations, ensuring no duplication within the signature recovered addresses. One gas-efficient way is to ask the client to sort the signatures according to the recovered addresses. Assuming that the input should be sorted, the validation can be as simple as `require(signer > prevSigner, ...)`.
2. Another issue is whether the roles attached to the stakeholders can differ. If they must be the same, then the impact of having duplicated stakeholders would simply be running the transaction with more gas. The team should clarify the business requirements and add validations if all roles should be identical.

Update: The team fixed the issue in the commit ` 2d9540by:

1. The team checked that there is no duplicated address as recommended.
2. The team clarifies that the roles can indeed differ.

QSP-10 Signature Replay Attack

Severity: *Low Risk*

Status: Fixed

File(s) affected: `LibEntities.sol`

Description: The `LibEntities._createSimplePolicy()` function recovers the ECDSA signatures with the message exposed to the risk of a replay attack. The message signs the `ECDSA.toEthSignedMessageHash(_policyId)`. The only variable is the `_policyId`. The signature can be replayed if the same `_policyId` is reused in multiple deployments or chains. Also, the system admin can maliciously collect the signatures but then use them to attach with different `entityId` or `role` as that information is not part of the signature.

Exploit Scenario: The team deploys on both the test and main net. Both networks create the policy with the same `_policyId`. The attacker can reuse the signature.

Recommendation:

1. Consider following the signature of EIP712 format, which includes the chain ID, contract address, and salt into the message to mitigate replay attacks. For instance, see the OpenZeppelin EIP712 code.
2. Also, we recommend adding `entityId` and `role` information to the signed message and checking the integrity against the `_stakeholders.entityIds[i]` and `_stakeholders.roles[i]`.

Update: The team fixed the issue as recommended in the commit `356e974`. Note that the role is still not part of the signature yet. The following is the original message from the team:

```
Recommendation implemented as suggested.
We should consider adding role information to the signed message. We currently have added the stakeholder entityId.
```

QSP-11 `ReentrancyGuard` Colliding with Storage Slot

Severity: *Low Risk*

Status: Fixed

File(s) affected: `EntityFacet.sol`, `MarketFacet.sol`

Description: Some facets inherit the `ReentrancyGuard` contract copied from the OpenZeppelin under `src/utils/`. However, the `ReentrancyGuard` implementation does not integrate with the diamond storage. The `ReentrancyGuard` will mutate a `_status` variable on the storage. The design of the `LibAppStorage.diamondStorage()` function (in the `AppStorage.sol`) uses slot zero instead of a randomized location. The `ReentrancyGuard` will override the data of the `AppStorage`.
The current contract will continue to work as expected because the first slot of the `AppStorage` is reserved for the `nonces` mapping. In EVM, the storage slot for mapping will not be used (see: blog). The EVM relies on the slot number rather than the value inside the slot to hash to store and get the mapping value. However, if the variable `uint256 totalSupply` is the first variable in the `AppStorage` struct, then it will be overridden by the `ReentrancyGuard` contract.
We set the severity as low because there is an immediate impact with the current codebase. However, this pattern is risky and might bring more considerable risks in the future.

Recommendation: We recommend doing the following:

1. Customize the `ReentrancyGuard` contract to use the diamond storage pattern instead.
2. It is safer to provide a namespace slot even for the `AppStorage`, similar to the `LibDiamond.DIAMOND_STORAGE_POSITION` instead of slot zero. Note that this will need to fix the `InitDiamond` contract too. The current implementation of the `InitDiamond` implicitly uses the slot zero `AppStorage internal s`.

Update: The team fixed the issue as recommended in the commit `d7f4a60`.

QSP-12 Risk of Insolvent Balance with Non-Standard Erc20 Token Integration

Severity: *Low Risk*

Status: Fixed

File(s) affected: `LibTokenizedVaultIO.sol`

Description: There are 2 concerns with the use of non-standard ERC20 tokens:

1. The external ERC20 token may use a non-standard decimal. For example, if token `A` uses a decimal value of `40`, then making a buy order with 1 token `A` will not even be possible since `10^40 > 2^128`, and `_assertAmounts()` make sure the amount is `<= type(uint128).max`.
2. The `LibTokenizedVaultIO._externalDeposit()` function relies on the input `_amount` to transfer, lock, and mint the internal tokens. However, the mechanism would fail if the ERC20 tokens take a fee or if the token is a rebaseable token. Integrating non-standard ERC20 tokens can have severe consequences, such as insolvent contracts, as it mints more than what is collected.

Note that the risk is primarily reduced as only approved tokens can be used in the Nayms system. Only the system admin can call the `AdminFacet.addSupportedExternalToken()` function to add tokens.

Exploit Scenario:

1. The contract integrates with a contract that takes a fee on transfer. Let's say the contract charges a fee of one per transfer.
2. The user calls `LibTokenizedVaultIO._externalDeposit()` with `_amount` equals to 10.
3. The contract will try to collect 10 tokens and mint 10 internal tokens. However, since the token contract takes a fee on transfer, only 9 tokens are collected.
4. The contract is insolvent now.

Recommendation: It might be impossible to support all kinds of tokens; as a result, we first recommend the system admin to be careful on adding new tokens. Second, we recommend that `LibTokenizedVaultIO._externalDeposit()` only mints what has been collected. In other words, the function should mint the difference between the balances before and after the `LibERC20.transferFrom()` call.

Update: The team first fixed the issue in the commit `ff160b5`. However, we found a few issues on the fix, and they provided the updates in the commits `b984b6662` and `7221ca152`. As a whole, the issue is fixed as the following:

1. The team adds a check that it only supports tokens with decimals not larger than 18 in the `LibAdmin._addSupportedExternalToken()` function.
2. The fix mints the balance difference in the `LibTokenizedVaultIO._externalDeposit()` function.

QSP-13 Invalid Policy Configurations

Severity: *Low Risk*

Status: Fixed

File(s) affected: `LibEntity.sol`

Description: While creating an entity, the code requires one of two things (see: `LibEntity.validateEntity()`):

- `_entity.simplePolicyEnabled` to be false
- `_entity.maxCapacity` should be greater than 0

However, the function `_updateAllowSimplePolicy()` can then be used to enable the `simplePolicyEnabled` for an entity with a `maxCapacity` of zero. Meanwhile, it is unclear whether it is acceptable to flag it back from `true` to `false` when the `_entity.maxCapacity` is non-zero.

Recommendation:

1. Before enabling the `simplePolicy` option, ensure that the entity has a `maxCapacity` greater than 0.
2. Clarify whether it is okay to flag `simplePolicyEnabled` from `true` to `false`.

Update: The function `updateAllowSimplePolicy()` is removed in the commit `80b245a`.

QSP-14 Incorrect Checks in `_assertValidOffer()`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `LibMarket.sol`

Description: In the function `_assertValidOffer()`:

- the first require statement checks that an entity identified by `_entityId` already exists, but the error message is `must belong to entity to make an offer`;
- the local boolean variable `sellTokenIsEntity` gets the value `true` if the token ID matches an existing entity. But it does not check that the token ID matches the entity ID. The same issue is for the local boolean `buyTokenIsEntity`.

Recommendation: Consider updating the code to align the variable names, the implemented checks and the associated error messages.

Update: The team adjusted the error message and variables to clarify the intention in the commit `92336d5`.

QSP-15 Out-of-Gas Errors when Adding External Tokens to a Protocol

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `LibAdmin.sol`

Description: The state variable `supportedExternalTokens` is used to record the external tokens supported by the protocol. A new token can be added with the function `_addSupportedExternalToken()`. As the whole list is browsed with a for loop, it can be possible that, once a given size is reached, the function reverts with an out-of-gas error. In this case, it will not be possible anymore to add new supported external tokens.

Recommendation: Consider setting an upper limit for the number of supported external tokens. Also, the `LibAdmin._addSupportedExternalToken()` function can be optimized by replacing the loop with the following. Note that line `s.externalTokenSupported[_tokenAddress] = true` should be moved inside the `if` block.

```
bool alreadyAdded = s.externalTokenSupported[_tokenAddress]
if (!alreadyAdded) {
    s.externalTokenSupported[_tokenAddress] = true;
    LibObject._createObject(LibHelpers._getIdForAddress(_tokenAddress));
    s.supportedExternalTokens.push(_tokenAddress);

    emit SupportedTokenAdded(_tokenAddress);
}
```

Update: The team optimize the code in the commit `a1d6ddb`. However, no cap is set for the number of external tokens.

QSP-16 Production Readiness Concern

Severity: *Low Risk*

Status: Fixed

File(s) affected: `LibEntity.sol`, `LibTokenizedVault.sol`, `TokenizedVaultFacet.sol`

Description: A few Todos or commented-out codes exist in the codebase, which leads to the concern about whether the codebase is production ready:

1. `LibEntity._validateSimplePolicyCreation()`:
    1. a todo on L42 `todo: ensure that the capital raised is >= max capacity`.
    2. Commented out code on L45: `require(entity.collateralRatio > 0 && entity.maxCapacity > 0, "currency disabled")`.
2. `LibTokenizedVault._withdrawDividend()`: the commented-out code `// require(withdrawableDividend > 0, "_withdrawDividend: no dividend")` on L179.
3. `TokenizedVaultFacet.internalTransfer()`: the commented-out code `// require(LibTokenizedVault._internalBalanceOf(senderId, tokenId) >= amount, "internalTransfer: insufficient balance");` on L51.

Recommendation: Please clarify whether the Todos and the commented-out codes are necessary to keep or fix before launching.

Update: The team removed TODOs and the commented code in the commit `c8b4f0b`.

QSP-17 Risk of Reentrancy

Severity: *Low Risk*

Status: Fixed

File(s) affected: `TokenizedVaultIOFacet.sol`, `LibTokenizedVaultIO.sol`

Description: The `TokenizedVaultIOFacet` contract interacts with external contracts (ERC20 tokens) without the `nonReentrant` guard. There are two main functions:

1. `TokenizedVaultIOFacet.externalDeposit()`: this function calls `LibTokenizedVaultIO._externalDeposit()`. However, the `LibTokenizedVaultIO._externalDeposit()` does not follow the checks-effects-interactions pattern. In other words, there is no protection against reentrancy in this function.
2. `TokenizedVaultIOFacet.externalWithdrawFromEntity()`: the library function `LibTokenizedVaultIO._externalWithdraw` follows the checks-effects-interactions pattern, so the reentrancy risk is mitigated here.

We did not find an obvious exploit path despite not having protection for the `TokenizedVaultIOFacet.externalDeposit()`. However, we recommend to add protections to avoid unexpected attack and mitigate risk for future code changes.

Recommendation: Consider adding the `nonReentrant` modifier to both functions to give complete protection. Alternatively, modify the `LibTokenizedVaultIO._externalDeposit()` function to follow the checks-effects-interactions pattern.

Update: The team added reentrancy guard and follows the checks-effects-interactions pattern in the commit `02ef589`.

QSP-18 Accidental Role Revocation in `NaymsOwnershipFacet`

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `NaymsOwnershipFacet.sol`

Description: The `NaymsOwnershipFacet.transferOwnership()` function can accidentally revoke the role when the `_newOwner` is the same as the current owner. The function first calls `LibACL._assignRole()` and then calls `LibACL._unassignRole()`. If the `_newOwner` is the same as the current owner, the first call `LibACL._assignRole()` will override the same thing, and the role will be immediately revoked by the call of `LibACL._unassignRole()`.

Exploit Scenario: The contract owner accidentally calls `transferOwnership()` setting the `_newOwner` to its address. The owner's ACL will no longer exist.

Recommendation: Change the order of the lines. Do the `LibACL._unassignRole()` before `LibACL._assignRole()`. Alternatively, add a validation that `_newOwner` is not the same as the old owner.

Update: The team removed the code to the un-assign role. The change removes the original issue. However, the fix changes the behavior as the old contract owners will be kept with the role. This reason for keeping the role is due to the fix for QSP-4. Due to the nuance change, we flag this as mitigated.

QSP-19 Wrong Values in Emitted Event

Severity: *Low Risk*

Status: Fixed

File(s) affected: `LibFeeRouter.sol`

Description: In `_payPremiumCommissions()`, the code emits the event `PremiumCommissionsPaid`, but the amount passed to the event is the total premium paid, not the premium commission paid.

Recommendation: Replace the emitted event in `_payPremiumCommissions()` with

```
uint256 premiumCommissionPaid = commissionNaymsLtd + commissionNDF + commissionSTM;

emit PremiumCommissionsPaid(_policyId, policyEntityId, premiumCommissionPaid);
```

Update: The team fixed the issue as recommended in the commit `3dbec68`.

QSP-20 Risk of `AppStorage` Mis-Ordering and Code Readability Concern

Severity: *Low Risk*

Status: Fixed

File(s) affected: `AppStorage.sol`

Description: The contract `AppStorage` lists the main variables used by the protocol. However, the description of some variables can sometimes be ambiguous or incomplete. In particular:

- the mechanism of dividends is explained in the middle of the contract;
- the difference between the state variables `lpAddress` and `pool` is unclear;
- the purpose of some variables is not described (`groupsConfig`, `canAssignConfig`);
- the measuring unit of some variables is not described (`numOwnedTokens`, `equilibriumLevel`, `actualDiscount`, `maxDiscount`, `actualNaymsAllocation`, `targetNaymsAllocation`);
- the fact that a given variable is currently used or not by the protocol is not always clear (`nonces`, commented variables related to `FEE BANK`);

As a result, it can be challenging for a reader to obtain further details about a given variable. Also, the pattern of having commented storage variables in the middle of a proxy contract is dangerous. If the storage gets uncommented during the upgrade, it will break the storage ordering and likely break the whole system.

Recommendation: Consider improving the readability of the contract by addressing at least the items listed above, and eventually adding a clear description of each state variable. Meanwhile, please DO NOT uncomment the storage slots declared in the middle once deployed. The change will break the storage order for the diamond proxy. New storages can be only added at the end of the struct (see: diamond upgrades).

Update: The team removed unused storage and the commented-out code in the commit `13391a8`. Those confusing variables are all removed.

QSP-21 Not Supporting EIP-165

Severity: *Informational*

Status: Fixed

File(s) affected: `Nayms.sol`

Description: The diamond contract `Nayms` never sets the `ds.supportedInterfaces`. The `DiamondLoupeFacet.supportsInterface()` function will always return `false`. The diamond spec does not necessarily require implementing EIP-165. However, having a non-functioning `supportsInterface()` function is confusing.

Recommendation: Consider setting the `ds.supportedInterfaces` inside the `constructor()` (see: reference implementation) or in the `InitDiamond.initialize()` function (see: reference implementation).

Update: The team fixed in the commit `f1cfd55` as recommended.

QSP-22 Unlocked Pragma

Severity: *Informational*

Status: Fixed

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity >=0.8.13;`. The version statement implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend locking the file onto a specific Solidity version. Also, `0.8.13`, at the time being, might be slightly outdated and we suggest moving it to a newer version.

Update: The team locked the version to `0.8.17` in the commit `122a973`.

QSP-23 Dividend Denominations Can only Be Increased

Severity: *Informational*

Status: Acknowledged

File(s) affected: `LibAdmin.sol`

Description: The function `_updateMaxDividendDenominations()` allows for updating the `maxDividendDenominations` variable. However, it only allows an increase in that value. If such value were inputted wrong, to begin with, it would only be allowed to increase from there.

Recommendation: In the `InitDiamond.initialize()` function, the `s.maxDividendDenominations` is set to one. This mitigates the risk, but it is worth considering if this is necessary and allows for lower values.

Update: The team will not remove supported tokens. The following is a more detailed response:

```
"We are aware that the number of dividend tokens supported can currently only be increased and this is a security measure we put in place as we research a full proof way of safely removing supported tokens. We currently
don't have a need to remove supported tokens.

The concern with removing a supported token is that if it still exists in our system, then there may be unwanted side effects if it's removed. The current thought is if we no longer have a balance in our system of a
particular supported token, then it can be safely removed from the supported token list."
```

QSP-24 Missing Interface Extension in Contracts

Severity: *Informational*

Status: Fixed

File(s) affected: `As Listed In the Description Section`

Description: It is generally recommended for Solidity contracts to inherit interfaces in order to define an abstract behavior for the extending contract. This helps to ensure that the contract implements the required functions and adheres to the specified behavior. Several interfaces are declared but not explicitly extended by the associated contract:

- interface `INayms` and contract `Nayms`;
- interface `IAdminFacet` and contract `AdminFacet`;
- interface `IEntityFacet` and contract `EntityFacet`;
- interface `IMarketFacet` and contract `MarketFacet`;
- interface `ISimplePolicyFacet` and contract `SimplePolicyFacet`;
- interface `ISystemFacet` and contract `SystemFacet`;
- interface `ITokenizedVaultFacet` and contract `TokenizedVaultFacet`;
- interface `IOTokenizedVaultIOFacet` and contract `TokenizedVaultIOFacet`;
- interface `IUserFacet` and contract `UserFacet`;

Recommendation: Consider using explicit interface extensions for the contracts listed above.

Update: Most interfaces are inherited in the implementations in the commit `bde9573`. The only exception is `INayms`, as it breaks the diamond architecture.

QSP-25 GTC Orders Risks

Severity: *Informational*

Status: Acknowledged

File(s) affected: `LibMarket.sol`, `MarketFacet.sol`

Description: In traditional financial exchanges, a submitted order has an expiration date (also known as a good 'til day order) after which the order will be considered invalid if it still has not been matched. Such a mechanism is not implemented as we only have a good 'til cancel (GTC) order system implemented.
However, there are risks associated with GTC orders, specifically the risk of "[executing] orders at inopportune moments, such as the brief rally in prices or temporary volatility. The consequent fallback in prices could leave traders with losses." (see: source)

Recommendation: Consider implementing the expiry date feature for orders. Nonetheless, the best offer design and the sorted list design might be impacted. Otherwise, acknowledge the issue and document the risk associated with having only GTC orders in the market.

Update: The team acknowledged the issue with the following statement:

```
We are aware of the GTC Orders Risks, but our platform is not expecting a huge number of active offers. We expect users to manage their offers proactively.
```

QSP-26 Unclear Spec or Mimatched Documentation

Severity: *Undetermined*

Status: Fixed

File(s) affected: `LibObject.sol`, `LibACL.sol`

Description: We noticed some places where the spec is unclear or the code mismatches with the documentation:

1. `LibACL._canAssign()`: the function plays an essential role in the library `LibACL`. However, it is hard to understand what should be its exact behavior due to incomplete documentation. For instance, the parameters need to be described. Also, it needs to be clarified why the function `_isParentInGroup()` is used to check if the

`assignerId` has a role in the context of the system.

2. `LibObject._createObject()`: the code only checks if the object already exists, despite a comment indicating that it should check for associated parent accounts (`// check if the id has been used (has a parent account associated with it) and revert if it has`).

Recommendation:

1. Consider improving the documentation of the functions with a clear description of the parameters and clear assignment rules regarding roles, groups, system context, and parents.
2. Consider aligning the code and the documentation.

Update: The team improved the documentation and the code comment in the commit `a368acb`.

QSP-27 Unused Functions Lacking Validations

Severity: *Undetermined*

Status: Fixed

File(s) affected: `LibAdmin.sol`

Description: We recommend adding validations to the unused functions that update storage variables. These functions may be intended for future features, but their inputs are currently not validated. It is unclear what kind of validations should be in place as it depends on the business requirements. Future code changes and audits should consider this issue and ensure appropriate validations are in place.

1. `LibAdmin._setPoolFee()`
2. `LibAdmin._setCoefficient()`
3. `LibAdmin._setDiscountToken()`
4. `LibAdmin._setTargetNaymsAllocation()`
5. `LibAdmin._setMaxDiscount()`
6. `LibAdmin._setEquilibriumLevel()`

Recommendation: Consider clarifying the requirements and adding input validation for these functions.

Update: The team removed the unused functions in commit `13391a8`.

QSP-28 Off-Chain Component Getting Non-Unique `guid`

Severity: *Undetermined*

Status: Fixed

File(s) affected: `TokenizedVaultFacet.sol`, `LibTokenizedVault.sol`

Description: The `TokenizedVaultFacet.payDividendFromEntity()` and `LibTokenizedVault._payDividend()` functions get a `guid` input and emit an event with the `guid` data. The `guid` is for the off-chain component. However, since the contract never uses the value aside from emitting it in the event, the `guid` can be non-unique and goes against the assumption of a "Globally unique identifier of a dividend distribution", as stated in the code document of the `TokenizedVaultFacet.payDividendFromEntity()` function.
Note that the `TokenizedVaultFacet.payDividendFromEntity()` function does not have authorization. So anyone can trigger the `guid` collision.

Recommendation: Consider adding a uniqueness check on the contract (e.g., have `guid` mappings).

Update: The commit `87dcf80` creates an object of the `_guid` to ensure uniqueness.

QSP-29 Updating an Entity Might Cause Inconsistencies

Severity: *Undetermined*

Status: Fixed

File(s) affected: `LibEntity.sol`

Description: The function `_updateEntity()` allows for updating the entity info, including the `collateralRatio`, which means that the new capital needed to underwrite policies could change.
It is unclear to us whether this will have an impact on the protocol or not.

Recommendation: Check and clarify if updating the `collateralRatio` can cause inconsistencies within the protocol. Usually, there should be some validations on the updating data.

Update: The commit `bc77e3d` adds validations in the `_updateEntity()` to ensure the `collateralRatio` would not break other parts.

QSP-30 Token Transfer Restrictions in `InternalTransfer()` May Be Bypassed in `_InternalTransfer()`

Severity: *Undetermined*

Status: Fixed

File(s) affected: `TokenizedVaultFacet.sol`

Description: In the function `internalTransfer()`, a require statement makes sure that the ID of the token transferred is not `LibConstants.STM_IDENTIFIER`. If the test passes, the internal function `_internalTransfer()` is called. However, there is no token restriction check in that second function, which can also be called from other functions. As a result, it may be possible to transfer tokens `LibConstants.STM_IDENTIFIER` via other paths. The severity level is "Undetermined" because the expected behavior is not documented.

Recommendation: Consider describing if the restriction should apply to the system or only to the function `internalTransfer()`. If the first option applies, consider moving the require statement from `TokenizedVaultFacet.internalTransfer()` to `LibTokenizedVault._internalTransfer()`.

Update: The team removed the validation in the commit `f2a06ab` as it is not used now.

Automated Analyses

Slither

The version 0.9.0 of Slither was used to analyze the `./src/diamonds/nayms` directory and found 44 results, most of which were either duplicates or false positives. The tool reported an issue with the old pragma version.

Adherence to Specification

1. There is a typo in the Policies page: `A policy must be approved my a Nayms System Manager` -> `...be approved by ....`

Code Documentation

1. In the `LibTokenizedVaultIO` and `LibTokenizedVaultIO` contracts, rephrase the code document regarding ERC1155. It might be clearer to state that the internal token mechanism is inspired but not following the ERC1155 standard.
2. **(fixed)** Clarify the future use case for the `LibTokenizedVault._payDividend()` function and add a comment describing it. The block `if (_internalTokenSupply(_to) == 0) {}` does not really have a clear business use case with the current code base now.
3. **(fixed)** `ACLFacet`:
    1. In function `assignRole()`, mismatch between type and description for the parameter `_roleId`.
    2. In function `canAssign()`, no description for the parameter `_assignerId`;
4. **(fixed)** `AdminFacet`: incorrect description for the function `isSupportedExternalToken()`;
5. **(fixed)** `TokenizedVaultFacet`:
    1. no documentation for the function `withdrawAllDividends()`.
    2. an uncommon call is made in the function `payDividendFromEntity()` where the value `entityId` is used for both parameters `_from` and `_to` when calling the function `_payDividend()`. An inline comment describing that behavior would improve the readability of the code.
6. **(fixed)** `LibACL`: in the event `RoleUpdate`, an incorrect verb is used for the parameter `roleId`. The word "assigned" could be used instead.
7. **(fixed)** `LibMarket`: the formula returned by the function `_isOfferPricedLtOrEq()` could be documented for more readability.
8. **(fixed)** `InitDiamond`: mismatch for the commission rates between the code (`0.4%`) and the technical documentation (`0.3%`): https://nayms.gitbook.io/nayms-docs/key-concepts/fees. **Update:** the team changed to 0.3%.
9. **(fixed)** `LibEntity`: the fact that the `entityId` is used as a parameter twice in the internal call to `_internalMint()` from the function `_startTokenSale()` can be misleading and could be explained with an inline comment.
10. **(fixed)** In `AppStorage.sol`, the comment for variable `dividendDenominationAtIndex` on line `36` should be `// entity ID => (index of dividend denomination => token id)` instead.
11. **(fixed)** In `LibEntity.sol#L190`, the revert message for `require(_entity.maxCapacity == 0, "only calls have max capacity");` should be `"only cells have max capacity"`.
12. **(fixed)** The revert message on `LibMarket._assertValidOffer()#L377` should be `one must be platform token` instead of `must be one platform token`. **Update:** the revert message is updated as `must be one participation token and one external token`.
13. `LibMarket._matchToExistingOffers()`: judging from the code comment and the logic, it seems like the function assumes one of the buy tokens or sell tokens should be an external token. The current codebase checks the assumption by `_assertValidOffer()` before calling this function. However, please add a code document to the function for future code changes to keep this assumption in mind when using this library function.

Adherence to Best Practices

1. **(fixed)** Replace `Nayms.fallback()#L27-32` with `ds = LibDiamond.diamondStorage()` instead.
2. **(fixed)** Check if the team can benefit from adding an index to the following events:
    1. `LibEntity.EntityCreated`: `entityId`.
    2. `LibEntity.EntityUpdated`: `entityId`.
3. **(fixed)** In the `LibEntity._createEntity()` function, instead of `delete _entity.utilizedCapacity`, it is better to validate that `_entity.utilizedCapacity == 0` instead. The change simplifies the function by reducing unnecessary side effects and magical logic.
4. **(fixed)** Consider removing the unused function `LibObject._isObjectTokenizable()`. **Update:** no longer unused after adding validations for another issue.
5. **(fixed)** In the `LibACL._canAssign()` function, the `assignerGroup` can be empty bytes if the `_roleId` does not exists in the mapping of `s.canAssign[_roleId]`. It is easier to read and has less risk of unexpected behavior if the function returns false directly when the `assignerGroup` is empty bytes.
6. In the `LibACL._isInGroup()` function, the `objectRoleInContext` can be empty bytes getting from the `s.roles` mapping. It is easier to read and has less risk of unexpected behavior if the function returns false directly when the `objectRoleInContext` is empty bytes. Note that the `objectRoleInContext` appears twice in the function, and the same recommendation applies to both places.
7. **(fixed)** Few components are imported but not used in:
    - `LibFeeRouter`: `TokenAmount`;
    - `AdminFacet`: `LibObject`;
    - `NaymsTokenFacet`: `LibAppStorage`, `AppStorage`;
    - `SystemFacet`: `LibAdmin`, `LibConstants`, `LibACL`;
    - `TokenizedVauldIOFacet`: `LibHelpers`, `LibTokenizedVault`;
    - `UserFacet`: `AppStorage`;
    - `LibACL`: `Modifiers`;
    - `LibConstants`: `LibHelpers`;

. `LibMarket`: `LibAdmin`;

. `LibTokenizedVaultIO`: `AppStorage`, `LibAppStorage`, `LibObject`;

. `InitDiamond`: `LibObject`;

. `Nayms`: `AppStorage`;

8. **(fixed)** `TokenizedVaultFacet.internalTransfer()`: there is a commented require statement. Without any explanation, as can be the case in other functions, it is difficult to assess if the comment is intentional or not. **Update:** the commented statement is removed.
9. **(fixed)** `LibAdmin`: event `BalanceUpdate` is declared but not used.
10. **(mitigated)**: `LibEntity._validateSimplePolicyCreation()`: two todo comments can be found. It suggests that the code is not finalized. **Update:** there is still one todo comment.
11. `LibEntity`: in the function `validateEntity()`, the value of the constant `BP_FACTOR` is hard coded in a comment as well as in an error message of a require statement. As a result, mismatches could appear if the value of `BP_FACTOR` is modified in the contract `LibConstants`. The same issue can be found in the functions `_payTradingCommissions()` and `_updateTradingCommissionsBasisPoints()` of the contract `LibFeeRouter`.
12. **(fixed)** `LibFeeRouter._payTradingCommissions()`: incorrect error message in the require statement. `<10000bp` should be replaced with `<=10000bp`
13. **(fixed)** `LibObject._enableObjectTokenization()`: incorrect error message in the require statements: the condition checks that `symbol` has less than 16 characters strictly.
14. **(fixed)** `LibTokenizedVault`: events `EntityDeposit` and `EntityWithdraw` are declared but not used.
15. **(fixed)** `LibTokenizedVault`: in the function `_internalTransfer()`, incorrect function name `_internalTransferFrom` used in the require statements, as well as in the emitted events.
16. **(fixed)** `LibTokenizedVault`: the local variable `dividendDenominationId` can be removed from the function `_withdrawAllDividends()`, and `dividendDenominations[i]` could be directly used instead.
17. **(fixed)** `LibTokenizedVaultIO`: events `NaymsVaultTokenTransfer` and `ExternalDeposit` are declared but not used.
18. **(fixed)** `AdminFacet`: the type returned by the function `getPoolFee()` should be `uint24` instead of `uint256` to avoid an implicit cast `uint24 -> uint256`. **Update:** the function is removed.
19. **(fixed)** `LibMarket`: using the order of layout recommended in the style guide of Solidity (https://docs.soliditylang.org/en/v0.8.17/style-guide.html#order-of-layout). For example, structs should be placed before events in a contract.
20. **(fixed)** `MarketFacet.cancelOffer()`: mismatch between checked condition and error message in the require statement. If the checked condition is correct, the message should be replaced with `only member of entity can cancel`.
21. `LibHelpers._getIdForObjectAtIndex()` is only used in the test. This should be removed from the library and put in mock or test contracts instead.
22. **(fixed)** On `LibMarket._createOffer()#L236`, it can be simplified by replacing `MarketInfo memory marketInfo = s.offers[lastOfferId]` with just `MarketInfo memory marketInfo;` (without the `s.offers[]` part). It only needs an empty struct data here.
23. [fix-review] Reuse the `Modifiers.assertEntityAdmin()` modifier for the `TokenizedVaultFacet.payDividendFromEntity()` function.
24. [fix-review] In the `LibEntity._updateEntity()` function, remove the magic mutation to override the `s.entities[_entityId].assetId` to the `originalAssetId` and ignore the asset value from the input `_entity`. Using a validation here instead makes the code easier to maintain without side effects.

Test Results

Test Suite Results

Run `forge test`.

```
No files changed, compilation skipped

Running 5 tests for test/T02User.t.sol:T02UserTest
 ✓[32m[PASS] ✓[0m testGetAddressFromExternalTokenId() (gas: 10893)
 ✓[32m[PASS] ✓[0m testGetBalanceOfTokensForSale() (gas: 571195)
 ✓[32m[PASS] ✓[0m testGetSetEntity() (gas: 188268)
 ✓[32m[PASS] ✓[0m testGetUserIdFromAddress() (gas: 10897)
 ✓[32m[PASS] ✓[0m testSetEntityFailsIfNotSysAdmin() (gas: 19867)
Test result:  ✓[32mok ✓[0m. 5 passed; 0 failed; finished in 13.56ms

Running 3 tests for test/T03NaymsOwnership.t.sol:T03NaymsOwnershipTest
 ✓[32m[PASS] ✓[0m testTransferOwernship() (gas: 63517)
 ✓[32m[PASS] ✓[0m testTransferOwernshipFailsIfNotContractOwner() (gas: 16342)
 ✓[32m[PASS] ✓[0m testTransferOwernshipWithRoleGroupsNotSetPropertly() (gas: 61651)
Test result:  ✓[32mok ✓[0m. 3 passed; 0 failed; finished in 8.39ms

Running 2 tests for test/T01LibERC20.t.sol:T01LibERC20
 ✓[32m[PASS] ✓[0m testTransfer() (gas: 82353)
 ✓[32m[PASS] ✓[0m testTransferFrom() (gas: 114241)
Test result:  ✓[32mok ✓[0m. 2 passed; 0 failed; finished in 14.59ms

Running 9 tests for test/T03SystemFacet.t.sol:T03SystemFacetTest
 ✓[32m[PASS] ✓[0m testD03CreateEntity() (gas: 16589)
 ✓[32m[PASS] ✓[0m testGetObjectMeta() (gas: 235677)
 ✓[32m[PASS] ✓[0m testIsObject() (gas: 232875)
 ✓[32m[PASS] ✓[0m testMultipleCreateEntity() (gas: 427380)
 ✓[32m[PASS] ✓[0m testNonManagerCreateEntity() (gas: 22954)
 ✓[32m[PASS] ✓[0m testSingleCreateEntity() (gas: 227430)
 ✓[32m[PASS] ✓[0m testStringToBytes32() (gas: 13406)
 ✓[32m[PASS] ✓[0m testUnsupportedExternalTokenWhenCreatingEntity() (gas: 24062)
 ✓[32m[PASS] ✓[0m testZeroCollateralRatioWhenCreatingEntity() (gas: 24230)
Test result:  ✓[32mok ✓[0m. 9 passed; 0 failed; finished in 13.37ms

Running 23 tests for test/T02ACL.t.sol:T02ACLTest
 ✓[32m[PASS] ✓[0m testAssignInvalidRole() (gas: 29648)
 ✓[32m[PASS] ✓[0m testAssignersCanAssignRole() (gas: 91964)
 ✓[32m[PASS] ✓[0m testAssignersCanUnassignRole() (gas: 102172)
 ✓[32m[PASS] ✓[0m testDeployerAssignAnyRoleToAnotherObjectInNewContext() (gas: 61619)
 ✓[32m[PASS] ✓[0m testDeployerAssignRoleToAnotherObject() (gas: 55993)
 ✓[32m[PASS] ✓[0m testDeployerAssignRoleToThemself() (gas: 36727)
 ✓[32m[PASS] ✓[0m testDeployerIsInGroup() (gas: 16348)
 ✓[32m[PASS] ✓[0m testDeployerUnassignRoleOnAnotherObject() (gas: 51396)
 ✓[32m[PASS] ✓[0m testDeployerUnassignRoleOnThemself() (gas: 26789)
 ✓[32m[PASS] ✓[0m testGetRoleInContext() (gas: 128465)
 ✓[32m[PASS] ✓[0m testHavingRoleInSystemContextConfersRoleInAllContexts() (gas: 98321)
 ✓[32m[PASS] ✓[0m testInvalidObjectIdWhenAssignRole() (gas: 14762)
 ✓[32m[PASS] ✓[0m testIsParentInGroup() (gas: 217599)
 ✓[32m[PASS] ✓[0m testNonAssignersCanAssignRoleIfTheirParentHasAssignerRoleInSystemContext() (gas: 255325)
 ✓[32m[PASS] ✓[0m testNonAssignersCanUnassignRoleIfTheirParentAsAssignerRoleInSystemContext() (gas: 276067)
 ✓[32m[PASS] ✓[0m testNonAssignersCannotAssignRole() (gas: 73387)
 ✓[32m[PASS] ✓[0m testNonAssignersCannotUnassignRole() (gas: 139150)
 ✓[32m[PASS] ✓[0m testRoleAssignmentEmitsAnEvent() (gas: 56135)
 ✓[32m[PASS] ✓[0m testRoleUnassignmentEmitsAnEvent() (gas: 85635)
 ✓[32m[PASS] ✓[0m testUpdateRoleAssigner() (gas: 98502)
 ✓[32m[PASS] ✓[0m testUpdateRoleAssignerFailIfNotAdmin() (gas: 21203)
 ✓[32m[PASS] ✓[0m testUpdateRoleGroup() (gas: 115136)
 ✓[32m[PASS] ✓[0m testUpdateRoleGroupFailIfNotAdmin() (gas: 21329)
Test result:  ✓[32mok ✓[0m. 23 passed; 0 failed; finished in 73.65ms

Running 16 tests for test/T04Entity.t.sol:T04EntityTest
 ✓[32m[PASS] ✓[0m testCancellSimplePolicy() (gas: 1068666)
 ✓[32m[PASS] ✓[0m testCheckAndUpdateSimplePolicyState() (gas: 1091192)
 ✓[32m[PASS] ✓[0m testCreateSimplePolicyAlreadyExists() (gas: 1078300)
 ✓[32m[PASS] ✓[0m testCreateSimplePolicyEmitsEvent() (gas: 1070276)
 ✓[32m[PASS] ✓[0m testCreateSimplePolicyEntitiesAreAssignedRolesOnPolicy() (gas: 1116064)
```

```
 ✓[32m[PASS] ✓[0m testCreateSimplePolicyFundsAreLockedInitially() (gas: 1072675)
 ✓[32m[PASS] ✓[0m testCreateSimplePolicySignersAreNotEntityAdminsOfStakeholderEntities() (gas: 1000433)
 ✓[32m[PASS] ✓[0m testCreateSimplePolicyUpdatesEntityUtilizedCapacity() (gas: 1576595)
 ✓[32m[PASS] ✓[0m testCreateSimplePolicyValidation() (gas: 1408742)
 ✓[32m[PASS] ✓[0m testEnableEntityTokenization() (gas: 241072)
 ✓[32m[PASS] ✓[0m testPayPremiumCommissions() (gas: 2450629)
 ✓[32m[PASS] ✓[0m testSimplePolicyPremiumsCommissionsClaims() (gas: 1766198)
 ✓[32m[PASS] ✓[0m testTokenSale() (gas: 715603)
 ✓[32m[PASS] ✓[0m testUpdateAllowSimplePolicy() (gas: 211364)
 ✓[32m[PASS] ✓[0m testUpdateCell() (gas: 239721)
 ✓[32m[PASS] ✓[0m testUpdateEntity() (gas: 169122)
Test result:  ✓[32mok ✓[0m. 16 passed; 0 failed; finished in 59.77ms

Running 9 tests for test/T01LibHelpers.t.sol:T01LibHelpers
 ✓[32m[PASS] ✓[0m testAddressToBytes32Fuzz(address) (runs: 256, μ: 789, ~: 789)
 ✓[32m[PASS] ✓[0m testBytes32ToBytesFuzz(bytes32) (runs: 256, μ: 9564, ~: 9564)
 ✓[32m[PASS] ✓[0m testBytes32ToStringFuzz(bytes32) (runs: 256, μ: 1351, ~: 1351)
 ✓[32m[PASS] ✓[0m testBytesToBytes32Fuzz(bytes) (runs: 256, μ: 860, ~: 857)
 ✓[32m[PASS] ✓[0m testGetAddressFromIdFuzz(bytes32) (runs: 256, μ: 388, ~: 388)
 ✓[32m[PASS] ✓[0m testGetIdForAddressFuzz(address) (runs: 256, μ: 456, ~: 456)
 ✓[32m[PASS] ✓[0m testGetIdForObjectAtIndexFuzz(uint256) (runs: 256, μ: 671, ~: 671)
 ✓[32m[PASS] ✓[0m testGetSenderId() (gas: 386)
 ✓[32m[PASS] ✓[0m testStringToBytes32Fuzz(string) (runs: 256, μ: 975, ~: 977)
Test result:  ✓[32mok ✓[0m. 9 passed; 0 failed; finished in 293.77ms

Running 31 tests for test/T02Admin.t.sol:T02AdminTest
 ✓[32m[PASS] ✓[0m testAddSupportedExternalToken() (gas: 105572)
 ✓[32m[PASS] ✓[0m testAddSupportedExternalTokenFailIfNotAdmin() (gas: 20384)
 ✓[32m[PASS] ✓[0m testAddSupportedExternalTokenIfAlreadyAdded() (gas: 109191)
 ✓[32m[PASS] ✓[0m testFuzzSetCoefficient(uint256) (runs: 256, μ: 42281, ~: 43392)
 ✓[32m[PASS] ✓[0m testFuzzSetDiscountToken(address) (runs: 256, μ: 22512, ~: 22512)
 ✓[32m[PASS] ✓[0m testFuzzSetEquilibriumLevel(uint256) (runs: 256, μ: 22175, ~: 22269)
 ✓[32m[PASS] ✓[0m testFuzzSetMaxDiscount(uint256) (runs: 256, μ: 22104, ~: 22236)
 ✓[32m[PASS] ✓[0m testFuzzSetPoolFee(uint24) (runs: 256, μ: 22429, ~: 22429)
 ✓[32m[PASS] ✓[0m testFuzzSetTargetNaymsAllocation(uint256) (runs: 256, μ: 22128, ~: 22278)
 ✓[32m[PASS] ✓[0m testGetActualNaymsAllocation() (gas: 12808)
 ✓[32m[PASS] ✓[0m testGetMaxDividendDenominationsDefaultValue() (gas: 12912)
 ✓[32m[PASS] ✓[0m testGetSystemId() (gas: 11062)
 ✓[32m[PASS] ✓[0m testIsSupportedToken() (gas: 97124)
 ✓[32m[PASS] ✓[0m testSetCoefficient() (gas: 50100)
 ✓[32m[PASS] ✓[0m testSetCoefficientFailIfNotAdmin() (gas: 20262)
 ✓[32m[PASS] ✓[0m testSetCoefficientFailIfValueTooHigh() (gas: 18780)
 ✓[32m[PASS] ✓[0m testSetDiscountToken() (gas: 33804)
 ✓[32m[PASS] ✓[0m testSetDiscountTokenFailIfNotAdmin() (gas: 20383)
 ✓[32m[PASS] ✓[0m testSetEquilibriumLevel() (gas: 33066)
 ✓[32m[PASS] ✓[0m testSetEquilibriumLevelFailIfNotAdmin() (gas: 20359)
 ✓[32m[PASS] ✓[0m testSetMaxDiscount() (gas: 33116)
 ✓[32m[PASS] ✓[0m testSetMaxDiscountFailIfNotAdmin() (gas: 20338)
 ✓[32m[PASS] ✓[0m testSetMaxDividendDenominations() (gas: 33465)
 ✓[32m[PASS] ✓[0m testSetMaxDividendDenominationsFailIfLowerThanBefore() (gas: 32245)
 ✓[32m[PASS] ✓[0m testSetMaxDividendDenominationsFailIfNotAdmin() (gas: 20304)
 ✓[32m[PASS] ✓[0m testSetPoolFee() (gas: 33253)
 ✓[32m[PASS] ✓[0m testSetPoolFeeFailIfNotAdmin() (gas: 20414)
 ✓[32m[PASS] ✓[0m testSetPremiumCommissionsBasisPoints() (gas: 50181)
 ✓[32m[PASS] ✓[0m testSetTargetNaymSAllocationFailIfNotAdmin() (gas: 20292)
 ✓[32m[PASS] ✓[0m testSetTargetNaymsAllocation() (gas: 33028)
 ✓[32m[PASS] ✓[0m testSetTradingCommissionsBasisPoints() (gas: 59951)
Test result:  ✓[32mok ✓[0m. 31 passed; 0 failed; finished in 431.90ms

Running 12 tests for test/T04Market.t.sol:T04MarketTest
 ✓[32m[PASS] ✓[0m testBestOffersWithCancel() (gas: 2724948)
 ✓[32m[PASS] ✓[0m testCancelOffer() (gas: 1041756)
 ✓[32m[PASS] ✓[0m testCommissionsPayed() (gas: 2398460)
 ✓[32m[PASS] ✓[0m testFuzzMatchingOffers(uint256,uint256) (runs: 256, μ: 1374236, ~: 1379600)
 ✓[32m[PASS] ✓[0m testFuzzMatchingSellOffer(uint256,uint256) (runs: 256, μ: 1536560, ~: 1532588)
 ✓[32m[PASS] ✓[0m testGetBestOfferId() (gas: 2847051)
 ✓[32m[PASS] ✓[0m testLibFeeRouter() (gas: 2301111)
 ✓[32m[PASS] ✓[0m testMatchMakerPriceWithTakerBuyAmount() (gas: 1735098)
 ✓[32m[PASS] ✓[0m testMatchingExternalTokenOnSellSide() (gas: 1675552)
 ✓[32m[PASS] ✓[0m testOfferValidation() (gas: 2044835)
 ✓[32m[PASS] ✓[0m testStartTokenSale() (gas: 1060625)
 ✓[32m[PASS] ✓[0m testUserCannotTransferFundsLockedInAnOffer() (gas: 1665174)
Test result:  ✓[32mok ✓[0m. 12 passed; 0 failed; finished in 1.47s

Running 4 tests for test/T01Deployment.t.sol:T01DeploymentTest
 ✓[32m[PASS] ✓[0m testDiamondLoupeFunctionality() (gas: 1429154)
 ✓[32m[PASS] ✓[0m testFork() (gas: 11009)
 ✓[32m[PASS] ✓[0m testInitDiamond() (gas: 3455151)
 ✓[32m[PASS] ✓[0m testOwnerOfDiamond() (gas: 12860)
Test result:  ✓[32mok ✓[0m. 4 passed; 0 failed; finished in 1.65s

Running 8 tests for test/T02LibHelpers.t.sol:T02LibHelpers
 ✓[32m[PASS] ✓[0m testAddressToBytes32Fuzz(address) (runs: 256, μ: 856, ~: 856)
 ✓[32m[PASS] ✓[0m testBytes32ToBytes(bytes32) (runs: 256, μ: 9541, ~: 9541)
 ✓[32m[PASS] ✓[0m testBytes32ToStringFuzz(bytes32) (runs: 256, μ: 9571, ~: 9571)
 ✓[32m[PASS] ✓[0m testBytesToBytes32(bytes) (runs: 256, μ: 866, ~: 863)
 ✓[32m[PASS] ✓[0m testGetAddressFromIdFuzz(bytes32) (runs: 256, μ: 455, ~: 455)
 ✓[32m[PASS] ✓[0m testGetIdForAddressFuzz(address) (runs: 256, μ: 545, ~: 545)
 ✓[32m[PASS] ✓[0m testIdForObjectAtIndexFuzz(uint256) (runs: 256, μ: 714, ~: 714)
 ✓[32m[PASS] ✓[0m testStringToBytes32Fuzz(string) (runs: 256, μ: 975, ~: 977)
Test result:  ✓[32mok ✓[0m. 8 passed; 0 failed; finished in 3.85s

Running 14 tests for test/T03TokenizedVault.t.sol:T03TokenizedVaultTest
 ✓[32m[PASS] ✓[0m testBasisPoints() (gas: 14243)
 ✓[32m[PASS] ✓[0m testDepositAndBurn() (gas: 1741112)
 ✓[32m[PASS] ✓[0m testFuzzSingleExternalDeposit(bytes32,bytes32,address,address,uint256) (runs: 256, μ: 975691, ~: 975723)
 ✓[32m[PASS] ✓[0m testFuzzTwoEntityDepositDividendWithdraw(uint256,uint256,uint256,uint256,uint256) (runs: 256, μ: 1109758, ~: 1160381)
 ✓[32m[PASS] ✓[0m testFuzzWithdrawableDividends(uint256,uint256,uint256) (runs: 256, μ: 1779146, ~: 1783167)
 ✓[32m[PASS] ✓[0m testMultipleDepositDividend() (gas: 1246666)
 ✓[32m[PASS] ✓[0m testMultipleDepositDividendWithdraw2() (gas: 1774035)
 ✓[32m[PASS] ✓[0m testMultipleDepositDividendWithdrawWithTwoDividendTokens() (gas: 3612305)
 ✓[32m[PASS] ✓[0m testPayDividendsWithNonZeroParticipationTokenSupply() (gas: 1275145)
 ✓[32m[PASS] ✓[0m testPayDividendsWithZeroParticipationTokenSupply() (gas: 316891)
 ✓[32m[PASS] ✓[0m testSingleExternalDeposit() (gas: 828191)
 ✓[32m[PASS] ✓[0m testSingleExternalWithdraw() (gas: 883846)
 ✓[32m[PASS] ✓[0m testSingleInternalTransferFromEntity() (gas: 301072)
 ✓[32m[PASS] ✓[0m testWithdrawableDividenWhenPurchasedAfterDistribution() (gas: 1783020)
Test result:  ✓[32mok ✓[0m. 14 passed; 0 failed; finished in 3.86s
```

Code Coverage

Run `forge coverage`. The coverage is high for most of the relevant contracts (`src/diamonds/nayms`).

File	% Lines	% Statements	% Branches	% Funcs
script/CreateEntity. s.sol	0.00% (0/10)	0.00% (0/15)	0.00% (0/2)	0.00% (0/1)
script/ UpdateCommissions.sol	0.00% (0/20)	0.00% (0/24)	100.00% (0/0)	0.00% (0/3)
script/deployment/DeployERC20. s.sol	0.00% (0/6)	0.00% (0/7)	100.00% (0/0)	0.00% (0/3)
script/deployment/GenerateInterfaces. s.sol	0.00% (0/24)	0.00% (0/29)	0.00% (0/2)	0.00% (0/1)
script/deployment/SmartDeploy. s.sol	0.00% (0/3)	0.00% (0/3)	100.00% (0/0)	0.00% (0/1)
script/utils/ DeploymentHelpers.sol	0.00% (0/509)	0.00% (0/606)	0.00% (0/94)	0.00% (0/20)
script/utils/ LibGeneratedNaymsFacetHelpers.sol	0.00% (0/139)	0.00% (0/171)	0.00% (0/44)	0.00% (0/5)
script/utils/ LibWriteJson.sol	0.00% (0/3)	0.00% (0/3)	100.00% (0/0)	0.00% (0/3)
src/diamonds/nayms/ AppStorage.sol	0.00% (0/1)	0.00% (0/1)	100.00% (0/0)	100.00% (1/1)
src/diamonds/nayms/ InitDiamond.sol	**100.00% (53/53)**	**100.00% (54/54)**	**100.00% (0/0)**	**100.00% (1/1)**
src/diamonds/nayms/ Nayms.sol	100.00% (5/5)	100.00% (6/6)	100.00% (0/0)	100.00% (1/1)

File	% Lines	% Statements	% Branches	% Funcs
src/diamonds/nayms/facets/ ACLFacet.sol	100.00% (15/15)	100.00% (18/18)	100.00% (4/4)	100.00% (10/10)
src/diamonds/nayms/facets/ AdminFacet.sol	100.00% (29/29)	100.00% (37/37)	100.00% (0/0)	100.00% (21/21)
src/diamonds/nayms/facets/ EntityFacet.sol	100.00% (6/6)	100.00% (6/6)	100.00% (0/0)	100.00% (6/6)
src/diamonds/nayms/facets/ MarketFacet.sol	100.00% (12/12)	100.00% (13/13)	100.00% (4/4)	100.00% (8/8)
src/diamonds/nayms/facets/ NaymsTokenFacet.sol	100.00% (2/2)	100.00% (2/2)	100.00% (0/0)	100.00% (2/2)
src/diamonds/nayms/facets/ SimplePolicyFacet.sol	100.00% (9/9)	100.00% (12/12)	100.00% (0/0)	100.00% (6/6)
src/diamonds/nayms/facets/ SystemFacet.sol	100.00% (4/4)	100.00% (4/4)	100.00% (0/0)	100.00% (4/4)
src/diamonds/nayms/facets/ TokenizedVaultFacet.sol	100.00% (15/15)	100.00% (19/19)	50.00% (2/4)	100.00% (8/8)
src/diamonds/nayms/facets/ TokenizedVaultIOFacet.sol	100.00% (5/5)	100.00% (6/6)	100.00% (4/4)	100.00% (2/2)
src/diamonds/nayms/facets/ UserFacet.sol	100.00% (5/5)	100.00% (5/5)	100.00% (0/0)	100.00% (5/5)
src/diamonds/nayms/libs/ LibACL.sol	100.00% (36/36)	100.00% (44/44)	100.00% (10/10)	100.00% (10/10)
src/diamonds/nayms/libs/ LibAdmin.sol	90.20% (46/51)	89.23% (58/65)	100.00% (8/8)	78.57% (11/14)
src/diamonds/nayms/libs/ LibEntity.sol	97.01% (65/67)	96.43% (81/84)	100.00% (44/44)	88.89% (8/9)
src/diamonds/nayms/libs/ LibFeeRouter.sol	100.00% (52/52)	100.00% (67/67)	66.67% (4/6)	100.00% (7/7)
src/diamonds/nayms/libs/ LibHelpers.sol	50.00% (6/12)	46.15% (6/13)	50.00% (1/2)	55.56% (5/9)
src/diamonds/nayms/libs/ LibMarket.sol	95.60% (152/159)	94.94% (169/178)	78.79% (52/66)	94.44% (17/18)
src/diamonds/nayms/libs/ LibNaymsToken.sol	0.00% (0/4)	0.00% (0/6)	100.00% (0/0)	0.00% (0/2)
src/diamonds/nayms/libs/ LibObject.sol	52.78% (19/36)	53.06% (26/49)	58.33% (7/12)	50.00% (6/12)
src/diamonds/nayms/libs/ LibSimplePolicy.sol	100.00% (38/38)	100.00% (45/45)	100.00% (16/16)	100.00% (6/6)
src/diamonds/nayms/libs/ LibTokenizedVault.sol	81.82% (72/88)	83.64% (92/110)	43.75% (14/32)	90.91% (10/11)
src/diamonds/nayms/libs/ LibTokenizedVaultIO.sol	100.00% (6/6)	100.00% (8/8)	100.00% (0/0)	100.00% (2/2)
src/diamonds/shared/facets/ DiamondCutFacet.sol	100.00% (15/15)	100.00% (18/18)	100.00% (6/6)	100.00% (1/1)
src/diamonds/shared/facets/ DiamondLoupeFacet.sol	94.37% (67/71)	93.94% (93/99)	83.33% (15/18)	60.00% (3/5)
src/diamonds/shared/facets/ NaymsOwnershipFacet.sol	100.00% (8/8)	100.00% (11/11)	75.00% (3/4)	100.00% (1/1)
src/diamonds/shared/facets/ OwnershipFacet.sol	100.00% (3/3)	100.00% (3/3)	100.00% (0/0)	100.00% (2/2)
src/diamonds/shared/libs/ LibDiamond.sol	21.62% (24/111)	21.54% (28/130)	18.52% (10/54)	55.56% (5/9)
src/diamonds/shared/libs/ LibMeta.sol	0.00% (0/5)	0.00% (0/5)	0.00% (0/2)	0.00% (0/1)
src/erc20/ LibERC20.sol	100.00% (15/15)	100.00% (16/16)	100.00% (12/12)	100.00% (3/3)
src/utils/ Create3Deployer.sol	0.00% (0/2)	0.00% (0/2)	100.00% (0/0)	0.00% (0/2)
test/defaults/ D00GlobalDefaults.sol	0.00% (0/4)	0.00% (0/4)	100.00% (0/0)	0.00% (0/1)
test/defaults/ D01Deployment.sol	0.00% (0/8)	0.00% (0/10)	100.00% (0/0)	0.00% (0/1)
test/defaults/ D02TestSetup.sol	0.00% (0/7)	0.00% (0/7)	100.00% (0/0)	0.00% (0/1)
test/defaults/ D03ProtocolDefaults.sol	0.00% (0/30)	0.00% (0/33)	100.00% (0/0)	0.00% (0/4)
test/fixtures/ LibERC20Fixture.sol	100.00% (2/2)	100.00% (2/2)	100.00% (0/0)	100.00% (2/2)
test/fixtures/ LibFeeRouterFixture.sol	100.00% (5/5)	100.00% (5/5)	100.00% (0/0)	100.00% (5/5)
test/fixtures/ SimplePolicyFixture.sol	100.00% (3/3)	100.00% (4/4)	100.00% (0/0)	100.00% (2/2)
test/fixtures/ TradingCommissionsFixture.sol	0.00% (0/2)	0.00% (0/3)	100.00% (0/0)	0.00% (0/1)
test/utils/ DSTestPlusF.sol	0.00% (0/7)	0.00% (0/8)	0.00% (0/2)	0.00% (0/2)
test/utils/ DummyToken.sol	**100.00% (17/17)**	**100.00% (17/17)**	**100.00% (10/10)**	**80.00% (4/5)**
Total	46.64% (811/1739)	46.72% (975/2087)	48.92% (226/462)	71.15% (185/260)

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
a2df14329ec0527becda9a64fb3ce910872f2923dcb00eb0ecdf756b0c76c187 ./diamonds/nayms/Nayms.sol
2f83195fb362168bbd030c7c9e8097d1d1486c40798dcb06cc2f0493184a26f1 ./diamonds/nayms/INayms.sol
d2bbcf04c41de4544dd4ec9daaffcb5814b0e99ef6d34f6ba1bc9a66ae4db3e1 ./diamonds/nayms/InitDiamond.sol
0f81d3b14e4a926cd05b71062cf373c7462d4bb134e033565698c07cba6cdf41 ./diamonds/nayms/AppStorage.sol
4214e96989c49d93c53c46b9266caebc66c478a20477ab1c312da8b9b845ee6f ./diamonds/nayms/Modifiers.sol
4432806fb4340981c1b10940ed40bee059c786a1885f932eb3509ab3f414fd35 ./diamonds/nayms/interfaces/ITokenizedVaultIOFacet.sol
105a72e70cdffcae280407a3855f9c9ef96dc05d32ffe8dce21640a094287d59 ./diamonds/nayms/interfaces/ITokenizedVaultFacet.sol
7a57cd40bd90f5f6bebcdde016b0a24e3c680ac4c2563029e59c9e476bca9c08 ./diamonds/nayms/interfaces/IUserFacet.sol
90a0cc6c9f862e0bbe5049440fa3588ea3dbfa17b997c09f8cb31efdd4f93c40 ./diamonds/nayms/interfaces/IAdminFacet.sol
65efa49746fd438b37ad52eb982d8c1298aeea547aa9b83d08779d0ee9acd59e ./diamonds/nayms/interfaces/IMarketFacet.sol
8383533bee20baaf92e9e7b85d1c1eae3f4b2a5023d51823968650d2393a6ee3 ./diamonds/nayms/interfaces/ISystemFacet.sol
1efeaac27561781feb7d3a9612d176b915ad6ab5e849bf1b65ca8f92c8658a3d ./diamonds/nayms/interfaces/IACLFacet.sol
ee9183b82812e466f19f8dd68d09252d36bbdb6a130e4c29e5458ef4653e4c0d ./diamonds/nayms/interfaces/IEntityFacet.sol
91047541909c9d9b36b3d0df5da0812f7548f6f8411e65b222ced45420c6ac5d ./diamonds/nayms/interfaces/ISimplePolicyFacet.sol
c430ccab6c8671ef10f57c87ed6bae7a8560539fb343dfe687c7e9cd999b0795 ./diamonds/nayms/interfaces/FreeStructs.sol
abb03376f5f9c770ea92ae7a5bece44a0536978d6f106b1b26fc4b0730aba18f ./diamonds/nayms/interfaces/INaymsTokenFacet.sol
189c3e523fc9a472dcd0a319babd1b8463fde8d9977c8bd8adff67851e62d8a5 ./diamonds/nayms/libs/LibObject.sol
22f8f4704575b282d6a1eecbb92f184b9a3467ae6f09352e70680618551ac429 ./diamonds/nayms/libs/LibHelpers.sol
263860645a048d29e39e4a30ed853213f30533db9debaf16b99f3c6ea934410b ./diamonds/nayms/libs/LibNaymsToken.sol
bac638cb999f22e7900270e9d50cb2c2ee351c7d2bbae33af27a5c60648976ef ./diamonds/nayms/libs/LibMarket.sol
1a0b7719abc1c5e3440c29027f09e73157f8558d9d451c5795a0167e56b894e2 ./diamonds/nayms/libs/LibConstants.sol
fb6de70654feb7a208ccf27aebe9b9ce7bd3d311c86a765fc5d9cc65cb3f0b69 ./diamonds/nayms/libs/LibTokenizedVault.sol
ab7136d31a52346ce3447b8ea6c555be1fcf79577920cdd30f99d110a219fe89 ./diamonds/nayms/libs/LibEntity.sol
3917168adb7b8e6f09f0d46990a493d339189f3ca3ce6c35a8d6b74cef42a2fc ./diamonds/nayms/libs/LibFeeRouter.sol
769f52988da22a3f0483fbd8689ab0d38c40737064944c11a7491ecb9cf1557d ./diamonds/nayms/libs/LibTokenizedVaultIO.sol
9e3d6169fd1ecc2e264f6d98e2c59c15b89d958b386946688fc8b7816ff1990e ./diamonds/nayms/libs/LibSimplePolicy.sol
ab0445e5a36d558c63adda1ad8f39bed55daa488e7a39919ee55420784ab499d ./diamonds/nayms/libs/LibAdmin.sol
45a1bca90841e885adae3da9a89f1a8b05c5fb5ba04f969e38a2dec793f85147 ./diamonds/nayms/libs/LibACL.sol
cfb2daa8459ad3274c96869ab843ef79b4e633a4f7a574f6182aa38b6cbf623f ./diamonds/nayms/facets/MarketFacet.sol
c7083d32be0066ac1459507857bc882a5a376982dae8568831b8814281d18410 ./diamonds/nayms/facets/UserFacet.sol
588d10ca79a224ba960d5646e71cc39f2add8205f4a470ababc86f86e1365208 ./diamonds/nayms/facets/SimplePolicyFacet.sol
601498f9ac71a26973d1c267bcb2ad0c24f3a35d3046c6f1d3275840383968cd ./diamonds/nayms/facets/ACLFacet.sol
8760d156c06e26c1160a027e8b51f837c8f1bee72dd9721233a4c51f330d9e9c ./diamonds/nayms/facets/TokenizedVaultIOFacet.sol
e02666472fceb327ff2360b3d28993df1abe4a971a4a1bc5576f1ddd16a81e94 ./diamonds/nayms/facets/AdminFacet.sol
13a949591791f890dd355c0904cc7c21332797eaf26b80b5245831f2d24cc12e ./diamonds/nayms/facets/TokenizedVaultFacet.sol
4d7ea7adbc9a512d37e9b72bfb829976c72737e804ef50e7a4875339278afc4f ./diamonds/nayms/facets/NaymsTokenFacet.sol
d99a2edcc9c5e0b8ce798fa74ca78d6e13a9d798a893b34e5234a6e8ad8b78ba ./diamonds/nayms/facets/EntityFacet.sol
6459e1f93663bf48454592993a18863d6203fa6e3260de3cd7ff1ac82a017296 ./diamonds/nayms/facets/SystemFacet.sol
4fe348e8803b6dd0e0ff97d7a6fd98535a0ad59fc7d21fa1fa0d0016eae01b2b ./diamonds/shared/interfaces/IDiamondCut.sol
20444bf9624b6275c3eaa4ba4050e31d8cd612246566a4522da39fc63e600315 ./diamonds/shared/interfaces/IDiamondLoupe.sol
cb3832167e999f91592f4e36c517f42baaebffaba05af5bf5cd537fc0e01d034 ./diamonds/shared/interfaces/IERC165.sol
28538237e1649741c64c23c6dce3b73ca4a58833f24492957ced1620b1f4ea00 ./diamonds/shared/interfaces/IERC173.sol
20dee0c6cb48ef2e445cd7f9f59e4f2cff45ed1fc5ccbbf36a2d796dbc594f06 ./diamonds/shared/libs/LibMeta.sol
7aa7fcc59629c9764f03f35e89a453beadd476fce3e5889a26250d03cd5f6135 ./diamonds/shared/libs/LibDiamond.sol
41a5779dd8796b5fee9eec35da42bdd8a89a26357453ec18cde849bb4cac74d9 ./diamonds/shared/facets/DiamondCutFacet.sol
64741a1b086072771c8c5794d2f992e2b57cbdf89e437e826b5729ebfb4530e2 ./diamonds/shared/facets/OwnershipFacet.sol
2a7ca2b04b6be3ed89139d1b7b6a9fa5bba8f95da2df1fbcc3213bce5ad8787f ./diamonds/shared/facets/NaymsOwnershipFacet.sol
2d282693abeb7d1e75820ef5b2c3fa2b99d36f4af525befa5431c042067a0de7 ./diamonds/shared/facets/DiamondLoupeFacet.sol
```

Tests

```
9dc02e0bc99a91272851b85ad28865f53fbad092e089f4c7d4ea10de4d79b63a ./test/T01SmartDeploymentV1.t.sol
27cad077648c98fa74329c8855ab85fddea443f35cf496b463ef297fd2c4489f ./test/T01LibHelpers.t.sol
edc3388d7afacf6e1e271e66dda282cd29efaa9892ddb9a0b7acb4ffe66996f1 ./test/T03SystemFacet.t.sol
e63790d049c3b31d46a21836c0c0499e5ab296fd8447e5b0271eb430e014653b ./test/T03TokenizedVault.t.sol
479fa7811e09180536ab158e1ef3df5756a353c933489c330dc93999d3781006 ./test/T02User.t.sol
3307d61744a8b726b2a81d4ea1dbfa274a8c823352e5798b5f675649432ebd27 ./test/T03NaymsOwnership.t.sol
```

71a78a4fd01c75f2d9ff2723442a2384e4a8158eef41ff4022734076820b86dc ./test/T04Entity.t.sol

ca63a6d6f40c2227622cbbbc63100300662ead0dbde773382ef4314da54d73bf ./test/T02ACL.t.sol

1f61cbd60665a84576334f929d41a28b1e548610f38d910010837b50bc0e7595 ./test/T01Deployment.t.sol

173648cf4c139f693ba1507f816bad57f7966d43fad7b8c172c740c97d97dad6 ./test/T02Admin.t.sol

3874e14f662796f99bb1c1d60237c1c5c6ebd59418998b150e50b183944e18d6 ./test/T02LibHelpers.t.sol

5b067ebcd7d2badf82e58f493da9870eb6f77bf7a24d2d087cca81e9e2077195 ./test/T04Market.t.sol

b7461e236b2e2ec6efdcb301457a9b0b8d6da42b4b97eca979c1c96f53af5756 ./test/T01LibERC20.t.sol

8c65b92a91a8db27f92d1462c117e9e63168c5ff99b4a1abdf71885152e9dfdf ./test/fixtures/TradingCommissionsFixture.sol

44b512acd13a32fd0d2dccf6e4f2da66b26cf9b8cbd0942ea3881ec828461cb3 ./test/fixtures/LibERC20Fixture.sol

92aa9b95f4e97ec1d89f3135c2d8bd32599d50948a9bfbed7b05035d9e913a72 ./test/fixtures/InitDiamondFixture.sol

724d11e2ffbbe4a59bb6faab8cb0a01a92aa831fbfd41ec9ad74b76f201e9dbc ./test/fixtures/LibFeeRouterFixture.sol

60b3b79e0ff201249dca5fa3700d167e524d14e0d6d9d50168e6794ecdac9e1c ./test/fixtures/SimplePolicyFixture.sol

14be1ee665159ea55582560203950e550c38d1462de38e752312895f439fe72d ./test/defaults/D02TestSetup.sol

7fc4c6f4115544c9fea07890c1d46ecb27ade7632ed6ba7cb2b8c3c2319a67d1 ./test/defaults/D01Deployment.sol

206b8c3eead10a90c18e65064e8752e934197ade0dedeee789f49d8b9d08a8d0 ./test/defaults/D03ProtocolDefaults.sol

74307b61d980ff63605179d175f7f7f3180e478381815d7c4ef4e2fff29f2b95 ./test/defaults/D00GlobalDefaults.sol

72e4dbdbc932c5eee7f8eb3f49639bfac624e23510a732e48b6a995bbffddbe0 ./test/utils/DummyToken.sol

f01167cbf5a24dddd7d308f7695d27a623963e0855d49f33dc03bc6b4183b3a1 ./test/utils/DSTestPlusF.sol

520f6a100436aff48ed138345802f57f2ec79c840be0493507d08b0a722723f1 ./test/utils/users/MockAccounts.sol

Changelog

- 2022-12-02 - Initial report
- 2022-12-22 - Fix-review report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Aave, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.