OpenZeppelin | security

Forta Staking Vault Audit

March 11, 2024

Table of Contents

Summary

Type	DeFi
Timeline	From 2024-02-05 To 2024-02-09
Languages	Solidity

Total Issues	23 (15 resolved, 3 partially resolved)
Critical Severity Issues	1 (1 resolved)
High Severity Issues	1 (1 resolved)
Medium Severity Issues	3 (1 resolved, 1 partially resolved)
Low Severity Issues	9 (8 resolved)
Notes & Additional Information	9 (4 resolved, 2 partially resolved)

Scope

We audited the NethermindEth/forta-staking-vault repository at commit ce87cffbf813e27cc83157933760b51fa44a1885.

In scope were the following files:

```
src/
├── FortaStakingVault.sol
├── InactiveSharesDistributor.sol
├── RedemptionReceiver.sol
├── interfaces
│   ├── IFortaStaking.sol
│   └── IRewardsDistributor.sol
└── utils
    ├── FortaStakingUtils.sol
    └── OperatorFeeUtils.sol
```

System Overview

The Forta Staking Vault project introduces a suite of contracts designed to enhance the user experience of delegating, undelegating, redeeming, and claiming assets in Forta Protocol pools. By utilizing the Forta Staking Vault, users can deposit assets into the system and redeem them later without having to engage directly in the delegation process. The delegation and undelegation processes are managed by a newly introduced role, the operator, who determines to which pools and for how long will assets be staked. The Forta Staking Vault is an extension of the upgradeable version of the ERC-4626 contract from the OpenZeppelin contracts library.

The workflow is as follows:

- Users deposit FORT tokens into the `FortaStakingVault` contract and in return receive shares that represent those tokens.
- The operator delegates some or all of the FORT tokens deposited by users to one or more pools of the `FortaStaking` contract. When assets are delegated, they are labeled as "active shares" in the `FortaStaking` contract.
- The operator can initiate the undelegation process for some or all of the FORT tokens delegated to one or more pools. This action triggers a cooldown period that is recorded in the vault for each pool from which tokens will be undelegated, creates a new `InactiveSharesDistributor` contract, and sends the corresponding Forta staking shares to initiate the withdrawal and transforms those shares into "inactive shares". Until the cooldown period expires, these assets cannot be withdrawn from the `InactiveSharesDistributor` contract.
- Once the cooldown period expires, anyone can trigger the undelegation process for a specific pool that already has an `InactiveSharesDistributor` contract created, which transfers all the FORT tokens back to the vault for later redemption by users.
- Users can then redeem and claim their assets. Notably, users can also redeem their assets before the cooldown period has expired. In this case, they will initially redeem a proportion of the assets they hold from the available balance of FORT tokens in the vault and will be able to claim the remaining assets after the cooldown period expires for all the subjects where their assets were being staked. Finally, a fee can be charged when redeeming and claiming assets, which is then sent to a fee treasury set by the admin of the vault.

Aside from the `FortaStakingVault` contract, two other contracts have an important role in the system:

`RedemptionReceiver`

Each vault user eventually will have their own `RedemptionReceiver` contract. This contract is responsible for managing redemptions and claims for the user. It keeps track of all pools with active shares that are ready to be withdrawn from the `FortaStaking` contract, as well as all distributors holding inactive shares that are either undergoing the cooldown period or are ready for withdrawal.

`InactiveSharesDistributor`

This ERC-20 contract is responsible for distributing assets to both the vault and the users' `RedemptionReceiver` contracts. Whenever the operator initiates the undelegation process for a specific `FortaStaking` pool, a new instance of the `InactiveSharesDistributor` is deployed. This means that for each pool, there could be one or more `InactiveSharesDistributor` instances, each potentially holding assets to distribute.

Security Model and Trust Assumptions

Privileged Roles

There are two privileged roles in the system:

- `DEFAULT_ADMIN_ROLE`: In charge of setting the redemption fee and the treasury address as well as managing roles defined in the system, using OpenZeppelin's `AccessControlUpgradeable` contract. Initially set as the deployer of the vault.
- `OPERATOR_ROLE`: In charge of delegating and initiating the undelegation process of FORT tokens to different `FortaStaking` pools. Initially set as the deployer of the vault.

Based on our discussions with the Forta Team, the operator and admin roles will be initially managed by them and the governance council.

Critical Severity

C-01 Incomplete Implementation of ERC-4626 Base Contract Leading to Asset Management and Usability Issues

The vault contract's adaptation from the ERC-4626 standard demonstrates critical flaws in asset management and functionality due to incomplete overrides of essential functions. Initially, the primary concern centers around the `mint` function. When users invoke the function, the contract correctly emits shares. However, it fails to update the _totalAssets variable and the pool's asset holdings. This discrepancy leads to a significant issue: the vault's asset tracking becomes inaccurate, which in turn will result in incorrect calculations when users attempt to redeem their shares.

Similarly, the omission of an override for the `withdraw` function can lead to transactions reverting when users attempt to execute withdrawals. Although this issue has a minor direct impact compared to the asset management flaw, it contributes to a degraded user experience, potentially deterring user interaction with the contract.

Consider modifying the `mint` and `withdraw` functions from the `ERC-4626Upgradeable` contract within the `FortaStakingVault` contract to accurately reflect withdrawn and minted assets in the `_totalAssets` variable.

Update: *Resolved in pull request #22.*

High Severity

H-01 Attacker Can Stall Undelegations

The process of undelegating assets from a subject in the `FortaStakingVault` contract requires two steps:

Firstly, the `operator` triggers the undelegation process by calling the `initiateUndelegate` function, which deploys a distributor instance, an ERC-20 token. The

vault transfers its `FortaStaking` shares in the given subject to the distributor, and in return, receives minted distributor tokens that represent its position and are equivalent to the `FortaStaking` shares transferred to the distributor contract. The distributor also initiates the withdrawal of the assets in the `FortaStaking` contract and the waiting period is stored in the Vault for that particular subject.

Secondly, once the waiting period passes, users are permitted to call the `undelegate` function to undelegate all the staked tokens of a specific subject by the vault. However, a problem arises when calculating the amount of FORT tokens to be sent from the distributor back to the vault. Instead of using the amount returned by the `withdraw` function from the `FortaStaking` contract, the balance of FORT tokens held by the distributor will be used instead which can be manipulated since anyone can send FORT tokens to the distributor.

If this occurs, the `undelegate` function will revert when attempting to subtract the assets delegated to the given subject since the amount subtracted will be greater than the one tracked in the `_assetsPerSubject` variable. This implies that the funds deposited in the distributor will become inaccessible and any subsequent attempts to invoke the `undelegate` function for that particular subject will fail.

Nonetheless, this is not irreversible as the operator can allocate additional tokens to the subject, thereby increasing the `_assetsPerSubject` amount to prevent an underflow. However, it may take some time before the Forta team notices this problem and the attacker could front-run any future delegations to put the undelegation process in a stalled situation again. A step-by-step proof-of-concept for this scenario can be found in this secret gist.

Consider using the amount returned by the `withdraw` function from the `FortaStaking` contract instead of the balance of FORT held by the distributor.

Update: *Resolved in pull request #24.*

Medium Severity

M-01 Lack of Event Emissions

The following functions do not emit relevant events after executing sensitive actions or modifying storage variables:

- The `updateFeeBasisPoints` function should emit a `FeeBasisPointsUpdated` event. This event should also be emitted in the `initialize` function.
- The `updateFeeTreasury` function should emit a `FeeTreasuryUpdated` event. This event should also be emitted in the `initialize` function.
- The `delegate` function should emit a `StakeDelegated` event.
- The `redeem` function should emit a `StakeRedeemed` event.
- The `deposit` function should emit a `StakeDeposited` event.
- The `claimRedeem` function should emit a `StakeClaimed` event.
- The `undelegate` function should emit a `StakeUndelegated` event.
- The `initiateUndelegate` function should emit an `UndelegateInitiated` event.

Consider emitting events following sensitive changes, including the initial event emission in the constructor where appropriate, and incorporating relevant parameters. This approach will facilitate tracking and alert off-chain clients monitoring the contracts' activity.

Update: *Partially resolved in pull request #28. The Forta team stated:*

> *Only added the first two suggested and one for `redeem`. Other functions have plenty of underlying events that can be used for the same, either events of the ERC-4626 or events of `FortaStaking`, with the exception of `redeem` which is not calling the ERC-4626 implementation.*

M-02 Unbounded Loops in Redeem Function May Cause DoS

When multiple delegations exist within the vault, the `redeem` function's iteration through various subjects and distributors introduces a risk of Denial of Service (DoS) for users. Due to the unbounded nature of these loops, a scenario with numerous delegations and distributors could result in exceeding Polygon's 30 million gas limit upon execution of the `redeem`

function. Although this issue is temporary, it has the potential to significantly degrade user experience.

Consider implementing a limitation on the maximum number of delegations permitted, thereby preventing such undesirable situations.

Update: *Acknowledged, not resolved. The Forta team stated:*

> *Acknowledged. DoS can only be done by the operator who should know and be cautious about it.*

M-03 Wrong and Incomplete Docstrings

Throughout the codebase, there are several parts that have wrong or incomplete docstrings:

- The documentation for the `undelegate` method within the `InactiveSharesDistributor` contract inaccurately states that vault shares are transferred to the vault. In reality, these shares are burned and it is the `FORT` tokens that are sent to the vault instead. In addition, the documentation for the `claim` function misleadingly indicates its use for claiming a portion of the inactive shares owned by individuals, whereas it actually facilitates the claiming of `FORT` tokens.
- The `initiateUndelegate`, `getRedemptionReceiver`, and `claimRedeem` functions in `FortaStakingVault.sol` do not have their return values documented.
- The `initiateUndelegate` and `claim` functions in `InactiveSharesDistributor.sol` do not have their return values documented.
- The `claim` function in `RedemptionReceiver.sol` does not have its parameters nor return values documented.
- The `redeem` and `deposit` functions use `inheritdoc` tags to reference `ERC4626Upgradeable` docstrings but fail to delineate the modifications from the original implementation.

Consider thoroughly documenting all functions/events (and their parameters or return values) that are part of any contract's public API. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

Update: *Resolved in pull request #26.*

Low Severity

L-01 Tokens Trapped in the Vault Might Cause Redemptions to Revert in Low FORT Liquidity Scenarios

The `redeem` function enables users to redeem their staked FORT tokens—or someone else's on their behalf. This function calculates the redeemer's share of the total vault assets and uses this proportion to transfer active staking tokens and distributor shares from the vault to the `redemptionReceiver`. This allows the user to later claim them through the `claimRedeem` function when available. Finally, it transfers the corresponding proportion of available FORT tokens in the vault that have not yet been delegated, utilizing the FORT token's `balanceOf` function for this purpose.

However, during this final operation, if extra tokens are present in the vault—whether they were directly sent to the contract by mistake or otherwise—these tokens will be included in the calculation when deducting the assets transferred to the user from the `_totalAssets` variable. Notably, the `_totalAssets` variable does not account for tokens directly transferred to the vault. Consequently, this discrepancy may lead to the `redeem` function reverting due to an underflow, especially when the last few users attempt to redeem tokens from the vault. A step-by-step proof-of-concept for this scenario can be found in this secret gist.

Consider accounting for all mistakenly sent FORT tokens in the vault in the `_totalAssets` state variable. Otherwise, consider implementing a `sweep` function that withdraws the token amount difference between `_totalAssets` and the actual balance reported by the FORT token contract's `balanceOf` function.

Update: *Resolved in pull request #30.*

L-02 Not Checking if There Are Assets in the Vault to Redeem

The `redeem` function in the `FortaStakingVault` contract allows users to redeem FORT tokens in exchange for shares. This function calculates the proportion of assets to be claimed through the `claim` function for those assets that are either active or inactive and calculates the same proportion of FORT tokens present in the vault for immediate withdrawal. However, in

the case that all the FORT tokens have been delegated and there are no FORT tokens present in the vault, the `redeem` function will still transfer 0 tokens.

To avoid unnecessary operations and extra gas costs, consider checking whether there are assets in the vault to be transferred to the user in the `redeem` function.

Update: *Resolved in pull request #32.*

L-03 Lack of Input Validation

Throughout the codebase, there are some instances in which the lack of input validation could lead to different undesired scenarios:

- The `delegate` function allows the operator to delegate zero assets to the subject. When a subject has no assets, The subject will be added to the subjects array without the ability to remove it through the undelegate function due to the absence of active shares. Attempting to fix this by delegating assets to the same subject to enable the undelegate functionality would add the subject to the array again.
- In the `initialize` function, there are no checks to validate that the `feeTreasury` is not the address 0 and that the `feeInBasisPoints` is lower than the `FEE_BASIS_POINTS_DENOMINATOR` variable, as it happens in the `updateFeeTreasury` and `updateFeeBasisPoints` functions respectively, which is inconsistent.

Consider implementing input validations in functions where parameters must be confined within specific boundaries. Furthermore, ensure that variables used across different functions are checked against the same boundaries to maintain consistency and integrity.

Update: *Resolved in pull request #34 at commit 959e20a.*

L-04 Missing Return Values in Functions Impair Protocol Integration and Information Flow

The `undelegate` function allows anyone to undelegate FORT tokens from the `FortaStakingVault` contract. However, this function does not return the amount of FORT tokens being withdrawn from the `FortaStaking` contract (through the `undelegate` function in the `InactiveSharesDistributor` contract), making it difficult for users to easily track how many tokens have been withdrawn, either off-chain or within a contract that calls the `undelegate` function and needs to store it.

A similar situation happens with the `delegate` function. It allows the operator to stake a given amount of previously deposited FORT tokens for a given subject to the `FortaStaking` contract through the `deposit` function. However, although `deposit` returns how many shares those tokens represent, and `delegate` has access to it, this value is not returned to the caller.

To avoid hindering off-chain operations and to make the vault more easily integrated with other contracts, consider returning the amount of FORT tokens withdrawn for the former and the amount of shares minted by the `FortaStaking` contract for the latter.

Update: *Resolved in pull request #36.*

L-05 Redundant State Variable

Throughout the codebase, one variable duplicate another already defined, accessible state variable:

- The `FortaStakingVault` contract defines the private `_token` state variable to track the underlying asset address. However, since this contract inherits from the `ERC4626Upgradeable` contract, it already has access to this address through the `asset` function.

Even though introducing this duplicate variable does not pose any security risk, it is unnecessary, error-prone, and can confuse developers and auditors.

To improve clarity and adhere to best practices in smart contract development, consider removing this variable and using the aforementioned getter function instead.

Update: *Resolved in pull request #38.*

L-06 Inadequate Visibility of State Variables in `RedemptionReceiver` Contract

The visibility of the state variables `_subjects` and `_subjectsPending` is set to `private`, posing a significant usability concern within the Forta Vault's claim functionality. This restricted visibility forces users to depend excessively on the Forta Vault user interface to know the remaining number of FORT tokens available for claim. Users might incorrectly conclude that they have claimed all entitled tokens following the execution of the `claimReedem` function, unaware that additional subjects may still be pending, awaiting the deadline for eligibility.

The `claim` function iterates through the `_subjects` array, verifying the timestamp against `_subjectsPending` and checking if the subject is in a frozen state. When a subject meets all the criteria, its stake is retrieved, it is then removed from the array, and its related entry in `_subjectsPending` is eliminated. If a subject fails to meet the necessary conditions, it is either because the user must have invoked the function past a certain deadline or because the subject is currently frozen.

Consider providing public access or creating getter functions. This change would let users independently verify their claimable tokens, reducing dependency on the Forta Vault UI and mitigating the risk of misunderstandings regarding their token claims.

Update: *Resolved in pull request #40.*

L-07 Insufficient Code Coverage

The codebase exhibits insufficient code coverage for branches and functions (currently under 77%). This level of coverage may leave critical portions of the code untested, potentially leading to undetected vulnerabilities.

Consider adding more unit tests to increase the code coverage to above 95%, adhering to best practices for software security and reliability. In addition, integrating code coverage tracking into the repository's Continuous Integration process is advised for ongoing quality assurance.

Update: *Acknowledged, will resolve. The Forta team stated:*

> *We will improve it after merging all changes associated with other findings, so we can make sure everything is well covered, especially all the branches.*

L-08 Duplicate Utilization of `FortaStakingUtils` Library

The project currently replicates the `FortaStakingUtils` library directly within its codebase instead of leveraging the Forta contracts repository as an external dependency. This approach introduces potential risks of inconsistencies between the version of the `FortaStakingUtils` used in the project and the latest version available in the Forta contracts repository. Such discrepancies could lead to unforeseen issues and complicate maintenance and updates.

Consider integrating the Forta contracts repository as a dependency. This strategy ensures that the project always utilizes the most current and consistent version of the `FortaStakingUtils`.

Update: *Resolved in pull request #42.*

L-09 Insufficient Project Information in `README.md`

The project's `README.md` file currently lacks substantial content and provides no specific information about the project itself. It appears to be the default file automatically generated by Foundry, which does not offer valuable insights or guidance for users or contributors.

Consider enriching the README.md with detailed project information, including its purpose, features, installation procedures, usage examples, and contribution guidelines. This enhancement will significantly improve the project's documentation, fostering a better understanding of the system and potentially attracting more contributors or users.

Update: *Resolved in pull request #57.*

Notes & Additional Information

N-01 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in these, it becomes easier for the maintainers of those libraries to make contact with the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, there are contracts that do not have a security contact:

- The `FortaStakingUtils` library.
- The `FortaStakingVault` contract.
- The `IFortaStaking` interface.
- The `IRewardsDistributor` interface.
- The `InactiveSharesDistributor` contract.
- The `OperatorFeeUtils` library.
- The `RedemptionReceiver` contract.

Consider adding a NatSpec comment containing a security contact above the contract definitions. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

Update: *Acknowledged, not resolved. The Forta team stated:*

> *No security contact in Forta staking contracts to replicate. Forta users will surely use Forta socials to get in touch.*

N-02 Inadequate Function Visibility

Throughout the codebase, some functions are defined as `public` but are not being accessed from the contract where they are defined:

- In `FortaStakingVault.sol`:
  - The `claimRewards` function
  - The `delegate` function
  - The `initiateUndelegate` function
  - The `undelegate` function
- All public functions in the `RedemptionReceiver` contract
- All public functions in the `InactiveSharesDistributor` contract

When declaring a function as external, its parameters are not copied to memory; instead, they are accessed directly from calldata. Calldata is a read-only, lower-cost area where function arguments are stored. This means that accessing parameters in external functions can consume less gas than accessing parameters in memory.

Consider changing the visibility of the aforementioned functions to `external`.

Update: *Resolved in pull request #44.*

N-03 The Implemented Access Control Presents Potential Risks for the Vault

The `FortaStakingVault` uses the `AccessControlUpgradeable` to incorporate two rules within the vault: the `OPERATOR_ROLE`, which will be in charge of everything related to the delegations, and the `DEFAULT_ADMIN_ROLE`, that is in charge of assigning and revoking the latter role.

As the `DEFAULT_ADMIN_ROLE` is intended to be assigned to only one address, it is not advisable to use the current `AccessControlUpgradeable` implementation. Using `AccessControlUpgradeable` might not effectively prevent mistakes such as assigning the admin role to multiple addresses, granting the role to a wrong address, or revoking its own role.

Consider using the `AccessControlDefaultAdminRules` contract as an alternative to `AccessControlUpgradeable`. This allows only one account to possess the `DEFAULT_ADMIN_ROLE`, ensuring better control. It also establishes a two-step process to shift the `DEFAULT_ADMIN_ROLE` to a different account and includes a configurable delay between the steps. An additional feature allows for the transfer to be canceled before its acceptance. Furthermore, other roles cannot be utilized to manage the `DEFAULT_ADMIN_ROLE`.

Update: *Partially resolved in pull request #46. The current implementation does not specify an initial delay; it defaults to zero. For enhanced readability and explicitness, we recommend using* `__AccessControlDefaultAdminRules_init_unchained` *to set the* `initialDelay` *and to grant the* `DEFAULT_ADMIN_ROLE`*.*

N-04 Multiple Instances of Missing Named Parameters in Mappings

Since Solidity 0.8.18, developers can utilize named parameters in mappings. This means mappings can take the form of `mapping(KeyType KeyName? => ValueType ValueName?)`. This updated syntax provides a more transparent representation of a mapping's purpose.

Throughout the codebase, there are multiple mappings without named parameters:

- The `_assetsPerSubject` state variable in the `FortaStakingVault` contract

- The `_subjectIndex` state variable in the `FortaStakingVault` contract
- The `_subjectInactiveSharesDistributorIndex` state variable in the `FortaStakingVault` contract
- The `_subjectDeadline` state variable in the `FortaStakingVault` contract
- The `_distributorSubject` state variable in the `FortaStakingVault` contract
- The `_subjectsPending` state variable in the `RedemptionReceiver` contract
- The `_distributorsPending` state variable in the `RedemptionReceiver` contract

Consider adding named parameters to the mappings to improve the readability and maintainability of the codebase.

Update: *Acknowledged, not resolved.*

N-05 Dependency on Polygon Mainnet Fork for Testing

The project extensively relies on a fork of the Polygon mainnet to validate tests against the current state of contracts intended for integration. While this approach has its merits, the absence of a locally executable testing framework poses significant drawbacks. The dependency on the Polygon mainnet fork requires a constant connection to an RPC endpoint which increases the test execution times and introduces potential points of failure associated with network reliability.

Consider implementing a comprehensive local testing environment. This environment should simulate all external interactions, allowing for both integration tests with the mainnet fork and independent local tests. Adopting this strategy will not only decrease testing durations by eliminating network dependencies but also ensure that testing can proceed uninterrupted by external network issues, thus streamlining the development and debugging processes.

Update: *Acknowledged, not resolved.*

N-06 Usage of `msg.sender` and `_msgSender`

Throughout the project, `msg.sender` is used to identify the sender of transactions. However, all OpenZeppelin contracts, from whom the project's contracts inherit, employ `_msgSender` instead. If the `_msgSender` function is overridden in the future, for example, to enable other parties to cover the gas costs of transactions, this adaptation will not be accurately represented in instances where `msg.sender` is used.

Consider either using `_msgSender` over `msg.sender` or documenting this distinction clearly to avoid problems. This will help prevent problems if modifications in how the `_msgSender` function works are introduced in the future.

Update: *Resolved in pull request #48.*

N-07 Typographical Errors

The following typographical errors were identified in the codebase:

- `FortaStakingVault.sol`:
  - Line 19: "stategy" should be "strategy", and "forta" should be capitalized as "Forta"
  - Line 169: "Overrided" should be "Overridden".
  - Line 290: "inmediatly" should be "immediately"
  - Line 291: "crated" should be "created"
  - Line 389: "an user" should be "a user"
- `InactiveSharesDistributor.sol`:
  - Line 13: "Inactives shares" should be "Inactive shares"
  - Line 15: "invalidShares" should be "inactive shares"
  - Line 16: "transferrable" should be "transferable"
- `RedemptionReceiver.sol`:
  - Line 34: "Initialiazes" should be "Initializes".

Consider resolving these typographical errors, as well as running an automated spelling/grammar checker on the codebase and correcting any identified errors.

Update: *Resolved in pull request #50.*

N-08 Inconsistent Licensing

The project uses multiple licenses, MIT and UNLICENSED, which could cause legal or operational issues.

If this is unintentional, standardizing the license across the codebase is recommended to avoid confusion. If intentional, clearly document the rationale for using multiple licenses and ensure compatibility between them.

Update: *Resolved in pull request #52.*

N-09 Gas and Code Optimizations

Several opportunities for gas and code optimizations have been identified across various functions and contracts, which could significantly improve efficiency and reduce execution costs:

- **For Loop Optimization**: Current implementations frequently recalculate array lengths within for loop iterations (e.g., line 86 of the `RedemptionReceiver` contract and line 315 of the `FortaStakingVault` contract). Optimizing these loops by caching the array length in a local variable before the loop starts can reduce gas costs and execution times.
- **Redundant Validation Removal**: The `_validateIsOperator()` function appears to be redundant and could be replaced with `onlyRole(OPERATOR_ROLE)` for role checking, simplifying the codebase and potentially saving gas.
- **Immutable State Variables**: Several state variables, such as `_token` and `staking`, do not change after contract initialization and thus could be declared as `immutable` and set in the constructor instead of the `initialize` function. This change could lower gas costs by reducing storage access.
- **Unnecessary Condition in `undelegate`**: The `undelegate` function contains a conditional check that is logically unnecessary. If vault shares are greater than zero, it logically follows that vault assets cannot be equal to zero, making the `if` statement redundant.
- **Local vs. State Variable Usage**: Inconsistencies in using local versus state variables have been noted. In this line, use the local variable `assetsReceived` both times to avoid the unnecessary `SLOAD` operation.

Update: *Partially resolved in pull request #53 and pull request #36. The third item on the list was acknowledged by the Forta team.*

Conclusion

The Forta Staking Vault introduces a way for users who do not want to get involved in all the steps of the delegation of FORT tokens to pools of the Forta protocol to an operator, improving the overall user experience.

One critical-severity and one high-severity issue were identified over the course of the audit: The former arises from not overriding the `mint` function of the ERC-4626 contract, which would lead to asset management issues, while the latter arises from how assets held by contracts are tracked, which could lead to a temporary DoS of certain functionalities of the vault.

During the fix review, the development team introduced a bug while addressing **L01 - Tokens Trapped in the Vault Might Cause Redemptions to Revert in Low FORT Liquidity Scenarios**. We believe this issue could have been detected by adhering to more rigorous testing practices. We recommend that the Forta team resolve **L-07 Insufficient Code Coverage**, which they have acknowledged. Additionally, we advise them to consistently verify that all state variables modified in tests align with expected outcomes, including balances.

Several fixes and recommendations were also suggested to improve the readability and clarity of the codebase and facilitate future audits, integrations, and development. Moreover, good documentation practices were also suggested.