GUARDIAN AUDITS

SMART CONTRACT SECURITY AUDIT OF

Rest Finance

Summary

Audit Firm Guardian

Prepared By Daniel Gelfand, Owen Thurm, Kiki, ABA, Kristian Apostolov

Client Firm Rest Finance

Final Report Date January 26, 2024

Audit Summary

Rest engaged Guardian to review the security of its LST vault, allowing users to deposit and withdraw a variety of LST's with Eigenlayer. From the 15th of January to the 22nd of January, a team of 5 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Notice that the examined smart contracts are not resistant to internal exploit. For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **Ethereum**

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite: https://github.com/GuardianAudits/RestPoCs

Table of Contents

Project Overview

Project Summary

Project Name	Rest Finance
Language	Solidity
Codebase	https://github.com/umbralxyz/rest-finance-impl
Commit(s)	Initial Commit: 2df40792fdee04d1ee243293faaf90ce2682d320 Final Commit: 95cf1014af2c2748b661c7f543e3eb076c28ded8

Audit Summary

Delivery Date	January 26, 2024
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Unresolved	Acknowledged	Partially Resolved	Resolved
● High	9	0	0	1	2	6
● Medium	13	0	0	6	1	6
● Low	22	0	3	12	2	5

Audit Scope & Methodology

ID	File	SHA-1 Checksum(s)
EWT	EiganWithdrawlTracker.sol	05ebe0b55dd88bd15d4415513977d5594f05ab09
VLT	Vault.sol	4cb3121b6fbd612b923d56dfbcd2dd28a3f5f353
SFETH	sfrxETHAdapter.sol	ede1305a6e93c39028793c3ed2266a571328336b
LSETH	LsETHAdapter.sol	57def27a359ce2c7c3a6ba95a7aa4d12fb114037
CBETH	cbETHAdapter.sol	93ba7037734f395c850e116dbf49527d75391f92
METH	mETHAdapter.sol	d1f71cc02373c236b65b2a557f25b63c6c9a31d5
STETH	stETHAdapter.sol	55a8707dab10f23f845942e0429f17513c356902
RETH	rETHAdapter.sol	c25681be438185e470394f63542b48aea0ca1c5d
CONST	Constants.sol	0774f17377735146aeda08e147e69ec4b2e2af53
PENW	PendingWithdraw.sol	94b460e517f132728fc8823293d7259256d3037d
ISU	Issuance.sol	1dfe3c91178c3a935ff57d9bdc61fa1851b5c660

Audit Scope & Methodology

Vulnerability Classifications

Vulnerability Level	Classification
🔴 High	Exploitable with high impact, causing loss/manipulation of assets or data.
🟡 Medium	Inherent risk of future exploits that may or may not impact the smart contract execution.
🟢 Low	Minor deviation from best practices.

Methodology

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.
- Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Invariants Assessed

During Guardian's review of Rest's Vault, fuzz-testing with Foundry was performed on the protocol's main functions. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 10,000+ runs up to a depth of 20 with a prepared Foundry fuzzing suite.

ID	Description	Tested	Passed	Run Count
VLT-01	Depositing Into Rest Increases User Share Balance	✅	✅	10,000+
VLT-02	Depositing Into Rest Increases Total Supply	✅	✅	10,000+
VLT-03	Depositing Specified LST Into Rest Decreased User Balance By Passed Amount	✅	❌	-
VLT-04	Depositing Into Eigen Does Not Change Vault Supply	✅	✅	10,000+
VLT-05	Depositing Into Eigen Increases Staker Strategy Shares	✅	✅	10,000+
VLT-06	Depositing Into Eigen Increases LST Balance of Strategy By Deposited Amount	✅	❌	-
VLT-07	Withdrawing With Assets Decreases Vault Supply By Share Amount Minus Fee	✅	✅	10,000+
VLT-08	Withdrawing With Assets Increases User LST Balance By Previewed Amount	✅	❌	-
VLT-09	Withdrawing Eigen Shares Decreases Staker Strategy Shares	✅	✅	10,000+
VLT-10	Withdrawing Eigen Shares Decreases User Vault Share Balance By Passed Amount	✅	✅	10,000+
VLT-11	Withdrawing Eigen Shares Decreases Vault's Total Assets	✅	✅	10,000+

Invariants Assessed

ID	Description	Tested	Passed	Run Count
VLT-12	Completing a Withdrawal Does Not Modify The Vault Supply	✅	✅	10,000+
VLT-13	Completing a Withdrawal Does Not Decrease User LST Balance	✅	✅	10,000+
VLT-14	Vault Supply = Sum of Actor Share Balances	✅	✅	10,000+
VLT-15	Total Assets Is Non-Zero If Vault Has LST Balance	✅	✅	10,000+
VLT-16	Total Assets Is Non-Zero If Vault Has Strategy Balance	✅	✅	10,000+

Findings & Resolutions

ID	Title	Category	Severity	Status
EWT-1	All Attempted Withdrawals Will Fail in EigenLayer M1	Logical Error	● High	Resolved
GLOBAL-1	Issuance Withdrawals Completed Through Vault	Logical Error	● High	Resolved
GLOBAL-2	stETH Rebase Frontrunning	Frontrunning	● High	Acknowledged
ISU-1	Tokens Stolen When Completing Withdraw	Logical Error	● High	Resolved
ISU-2	All Early Withdrawals Fail	Logical Error	● High	Resolved
ISU-3	Unbounded Call Leads To Gas Griefing and DoS	Griefing	● High	Partially Resolved
STEA-1	stETH Price Hardcoded To 1 ETH	Logical Error	● High	Partially Resolved
VLT-1	restETH Becomes Cheaper Upon Withdrawals	Logical Error	● High	Resolved
VLT-2	Vault Can Be Drained Through Overlooked Mint Function	Logical Error	● High	Resolved
GLOBAL-3	Eigen Airdrop Cannot Be Claimed	Logical Error	● Medium	Resolved
GLOBAL-4	Lack Of Migration Or Extension Mechanisms	Upgradability	● Medium	Resolved
GLOBAL-5	Anyone Can Remove All LST Deposits From A Strategy	Logical Error	● Medium	Acknowledged
ISU-4	Potential For Trapped Ether	Trapped Ether	● Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
ISU-5	Incorrect Withdrawal Maturity Check	Logical Error	● Medium	Resolved
ISU-6	Users Can Avoid Withdrawal Fee	Logical Error	● Medium	Acknowledged
ISU-7	Pending Withdrawals Can Be Bought When Eigen Is Paused	Logical Error	● Medium	Resolved
PENW-1	DoS pendingWithdrawals	DoS	● Medium	Acknowledged
VLT-3	Deposits Can Be Griefed	Griefing	● Medium	Acknowledged
VLT-4	Invalid Flagship LST Amount Transferred In	Logical Error	● Medium	Resolved
VLT-5	Vault is not ERC4626 Compliant	Compliance	● Medium	Partially Resolved
VLT-6	Attacker can Front-Run Removal of Asset	Frontrunning	● Medium	Acknowledged
VLT-7	Yield In Strategies Can Be Stolen	Gaming	● Medium	Acknowledged
CONST-1	Unresolved TODO	Best Practices	● Low	Acknowledged
GLOBAL-6	No Incentive For Unsupported Eigen Assets To Use Rest	Incentives	● Low	Acknowledged
GLOBAL-7	Use SafeERC20 For Token Operations	Best Practices	● Low	Unresolved
GLOBAL-8	Possible Trapped Funds When Using LST With Blacklist	Best Practices	● Low	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
GLOBAL-9	Redundant Code	Superfluous Code	● Low	Partially Resolved
GLOBAL-10	Missing Input Validations	Validation	● Low	Acknowledged
GLOBAL-11	Lack of CEI	Reentrancy	● Low	Resolved
ISU-8	Pending Withdraws Not Cleared	Best Practices	● Low	Unresolved
ISU-9	New Owner Cannot Decide wantsETH Value	Logical Error	● Low	Acknowledged
ISU-10	Potential Duplicate Of Withdrawal Funds	Logical Error	● Low	Resolved
VLT-8	Lack of Fee Basis Points Validation	Validation	● Low	Resolved
VLT-9	Zero Address Fee Receiver Bricks Withdrawals	Logical Error	● Low	Resolved
VLT-10	Missing Events On Key State Changes	Events	● Low	Acknowledged
VLT-11	Added LST Lacks Adapter Validation	Validation	● Low	Partially Resolved
VLT-12	Risk Of Too Many LSTs	Validation	● Low	Acknowledged
VLT-13	Single Step Ownership Transfer	Centralization Risk	● Low	Acknowledged
VLT-14	Typo	Typo	● Low	Unresolved

Findings & Resolutions

ID	Title	Category	Severity	Status
VLT-15	Withdraw Using Assets Can Always Be Blocked	Logical Error	● Low	Acknowledged
VLT-16	Missing Zero Fee Amount Check	Validation	● Low	Acknowledged
VLT-17	Fee Receiver Cannot Withdraw 100% of Amount	Logical Error	● Low	Resolved
VLT-18	User Can Withdraw Before Slashing	Future Changes	● Low	Acknowledged
VLT-19	safeTransferFrom After External Call	Reentrancy	● Low	Acknowledged

EWT-1 | All Attempted Withdrawals Will Fail in EigenLayer M1

Category	Severity	Location	Status
Logical Error	● High	EiganWithdrawlTracker.sol: 45	Resolved

Description PoC

When a staker wants to withdraw their eigen shares through function withdrawUsingEigenShares, the function _queueWithdraw is called which queues a withdrawal through the delegation manager: delegationManager.queueWithdrawals(arrayify(withdraw));

However, the currently deployed M1 DelegationManager does not support function queueWithdrawals and all calls to it will revert. Consequently, users are entirely unable to withdraw their restaked assets.

Recommendation

Consider making the contracts upgradeable such that when EigenLayer upgrades to the M2 contracts, the Rest Vault's functionality can be updated.

Furthermore, add functions for user to withdraw their eigen shares through the current M1 StrategyManager with function queueWithdrawal and completeQueuedWithdrawal.

Resolution

Rest Team: Resolved.

GLOBAL-1 | Issuance Withdrawals Can Be Trapped

Category	Severity	Location	Status
Logical Error	● High	Global	Resolved

Description PoC

Anyone may complete a withdrawal that was queued through the Issuance contract by calling the `completeWithdraw` function on the Vault contract directly.

Because the `owner` is the Issuance address for withdrawals done through the Issuance contract, a user can complete a withdrawal for another user and their funds will be sent to the Issuance contract and stuck. Furthermore, the `pendingWithdraws` for the true owner will not be updated in the Issuance contract.

Recommendation

Refactor the integration between the Issuance contract and the Vault contract such that the `pendingWithdraws` in the Issuance contract cannot be circumvented.

Resolution

Rest Team: The issue was resolved in commit d93fb8e.

GLOBAL-2 | stETH Rebase Frontrunning

Category	Severity	Location	Status
Frontrunning	● High	Global	Acknowledged

Description PoC

stETH rebases once a day to reflect the yield earned from staking ETH. This creates issues in the Vault and Issuance contracts.

Firstly, in the Vault contract a malicious actor may deposit a different LST, say rETH, before the stETH rebase and withdraw more rETH directly after the stETH rebase -- as the value of the vault has increased by the stETH yield.

Secondly, a malicious actor may frontrun the stEth rebase and use the `completeWithdrawEarly` function in the Issuance contract to buy out a user's stETH pending withdrawal before the rebase is recorded, therefore obtaining the assets at a discount.

Recommendation

To address the Vault issue, ensure that the withdrawal fee is significant enough to deter make any extraction of the stEth rebase unprofitable.

To address the Issuance issue, consider implementing a protocol fee for the buyer of stEth withdrawals such that any value gained from the stEth rebase is overshadowed by the amount paid to the protocol.

Resolution

Rest Team: Acknowledged.

ISU-1 | Tokens Stolen When Completing Withdraw

Category	Severity	Location	Status
Logical Error	● High	Issuance.sol: 99	Resolved

Description PoC

The Issuance contract was created to facilitate early buyouts of pending withdraws. The function `completeWithdraw` must be called after a withdraw has matured from a queued state.

This function incorrectly sends the funds to the caller of the function, rather than the owner of the pending withdraw. This makes it possible for anyone to call this function, with any non-pending withdraw and steal the funds.

Recommendation

Validate that the owner of the withdrawal is the `msg.sender`.

Resolution

Rest Team: The issue was resolved in commit 90dc6f3.

ISU-2 | All Early Withdrawals Fail

Category	Severity	Location	Status
Logical Error	● High	Issuance.sol: 63	Resolved

Description PoC

The `completeWithdrawalEarly` function from the `Issuance` contract allows users to "buy out" the withdrawal of someone else in return for the withdrawal's equivalent in their desired LST/ETH.

It checks the opposite of what it should. Only withdrawals that have a set root should continue being executed, when the check is doing quite the opposite. This will completely DoS the function from being used as intended and will further introduce unexpected behavior.

Recommendation

```
if (pendingWithdraws.owner(root) == address(0)) revert UnknownRoot()
```

Resolution

Rest Team: The issue was resolved in commit b5343d8.

ISU-3 | Unbounded Call Leads To Gas Griefing and DoS

Category	Severity	Location	Status
Griefing	● High	Issuance.sol: 82	Partially Resolved

Description PoC

Issuance.completeWithdrawalEarly() is used to "buy out" a withdrawal from someone in return for the equivalent the withdrawal's value in the LST of the withdrawal or in ETH if the withdrawal was queued with wantsEth = true.

That equivalent value gets sent to the oldOwner of the withdrawal either through and ERC20.transfer or through an address.call in the case of ETH.

Calling an address through address.call() without a gas stipend allows the call recipient to use as much as 63/64 of the gas left for execution.

This allows the recipient to do two things:

1. Maliciously expend gas in order to make the transaction be very expensive for the caller.

2. Trigger a revert, which will make the transaction fail and disallow it from being completed, thus causing DoS.

Recommendation

Implement the classical pull pattern, instead of the current push implementation, with regards to how the original withdrawal owner gets the payment for his withdrawal.

Resolution

Rest Team: The issue was partially resolved in commit edeb661.

Guardian: It is still possible for a user to block anyone from early withdrawing their shares by depositing in the Issuance contract through a contract that does not have a receiver/fallback. Consider implementing the pull pattern instead.

STEA-1 | stETH Price Hardcoded To 1 ETH

Category	Severity	Location	Status
Logical Error	● High	stETHAdapter.sol: 28	Partially Resolved

Description PoC

stETH is a rebasing token. It distributes staking rewards through increasing the balances of users with their accrued APR daily. Since the total supply of stETH represents the total staked ETH plus the staking rewards it has a price that is usually very close to that of ETH.

The issue arises due to the price of stETH being hardcoded to `1e18`. Even though the token is loosely pegged to ETH, 1 stETH ≠ 1 ETH.

1. When price of stETH < the price of ETH:
    A. Deposits in stETH will mint more to the user than supposed to, devaluing the shares.
    B. Withdrawing in stETH will make users receive less than intended, thus losing them funds.

2. When price of stETH > ETH:
    A. Depositing stETH will output less shares than should, thus damaging the depositor.
    B. Withdrawing stETH will be more profitable than should be, thus devaluing the shares.

Recommendation

To mitigate the issue change the stETH adapter's value() function to use the stETH/ETH Chainlink Price Feed.

Resolution

Rest Team: The issue was partially resolved in commit cdbb74a.

VLT-1 | restETH Becomes Cheaper Upon Withdrawals

Category	Severity	Location	Status
Logical Error	● High	Vault.sol: 163	Resolved

Description

In the completeWithdraw function the restETH shares are only burned upon completion of the withdrawal. However the shares attributed to the Vault contract in the Eigenlayer StrategyManager contract are reduced upon the queuing of a withdrawal.

The strategy contract will often refer to the StrategyManager.stakerStrategyShares to produce the shares result (see StrategyBase). Therefore the totalAssets value is reduced immediately upon calling the withdrawUsingEiganShares function, while the corresponding decrease in the restETH supply only occurs when the withdrawal is completed in the completeWithdraw function.

As a result, the price of restETH will errantly drop when users initiate a withdrawal with the withdrawUsingEiganShares function.

Recommendation

Reduce the shares of restETH immediately in the withdrawUsingEiganShares function, as this is when the corresponding reduction in totalAssets occurs.

Resolution

Rest Team: The issue was resolved in commit 2f68a92.

VLT-2 | Vault Can Be Drained Through Overlooked Mint Function

Category	Severity	Location	Status
Logical Error	● High	Vault.sol	Resolved

Description PoC

The RestEthVault contract Vault has an adapter for each LST it supports, including for the flagship asset, that will be set as the ERC4626 vault asset. The contract mistakenly does not overwrite the ERC4626.mint(uint256,address) function, meaning that when receiving shares through that function, the required asset LST asset amount will be mapped as a parity of 1:1 with ETH.

If the flagship LST has a moment when it is valued higher than ETH, by depositing through this function an attacker could then instantly resell the shares, using the withdrawUsingAssets function for a profit, after fees.

The attack would require that funds be existing in the vault, thus back-running any deposit call and would require the difference in price in the flagship LST and ETH so that it is profitable after withdraw fees.

Recommendation

Override the ERC4626.mint(uint256,address) function and use the adapter provided price.

Resolution

Rest Team: The issue was resolved in commit dcbb07b.

GLOBAL-3 | Eigen Airdrop Cannot Be Claimed

Category	Severity	Location	Status
Logical Error	● Medium	Global	Resolved

Description

Stakers which have their LST’s staked in EigenLayer will be eligible for an airdrop. However, there is currently no way for users of the Rest Vault to claim these funds.

Recommendation

Consider adding the functionality to withdraw airdropped tokens and/or make the contracts upgradeable.

Resolution

Rest Team: Resolved.

GLOBAL-4 | Lack Of Migration Or Extension Mechanisms

Category	Severity	Location	Status
Upgradability	● Medium	Global	Resolved

Description

The current implementation of the RestEthVault contract lacks a significant part of protocol target functionality.

Besides protocol specific functionality it also lacks a means to migrate funds to a new contract or a means to delegate to an operator in the EigenLayer ecosystem. The second aspect may be relevant for any future airdrop or yield increase.

Recommendation

Until the protocol reaches maturity, in order to support incremental feature development, use an upgradable pattern.

Resolution

Rest Team: Resolved.

GLOBAL-5 | Anyone Can Remove All LST Deposits From A Strategy

Category	Severity	Location	Status
Logical Error	● Medium	Global	Acknowledged

Description

Anyone can force the system to withdraw the entirety of any specific asset from Eigenlayer by depositing a different asset and then using the `withdrawUsingEigenShares` function to queue a withdrawal for the vaults entire strategy shares.

This way the Rest system would be unable to earn yield and may not be eligible for an airdrop. A competing LST protocol may do this to gain their own position in a strategy that has maximum TVL limits.

Recommendation

Consider implementing a mechanism to prevent malicious actors from removing the Vault's allocation in strategies.

Resolution

Rest Team: Acknowledged.

ISU-4 | Potential For Trapped Ether

Category	Severity	Location	Status
Trapped Ether	● Medium	Issuance.sol: 84	Resolved

Description

In the `wantsEth[root] == false` case, it is possible for Ether to become trapped in this contract since the `msg.value` is not validated to be 0 nor refunded to the user.

Recommendation

Either validate that the `msg.value` is 0 when `wantsEth[root] == false` and the `else` case is entered, or refund this ETH to the caller.

Resolution

Rest Team: The issue was resolved in commit fe74bf2.

ISU-5 | Incorrect Withdrawal Maturity Check

Category	Severity	Location	Status
Logical Error	● Medium	Issuance.sol: 104	Resolved

Description

`Issuance` checks whether a withdrawal has matured in order to allow calling `completeWithdrawEarly` only on non-matured withdrawals. EigenLayer makes withdrawal requests wait roughly a week after they got queued in order to be able to react and punish in cases where the staker/operator of the staker acted maliciously in any sort of way.

On the contrary the logic that enforces this in the `Issuance` contract implements the following access control:

```
block.number > withdraw.startBlock + vault.delegationManager().withdrawalDelayBlocks()
```

It explicitly requires that the current block is greater than the queue start plus the delay period, thus allowing early withdrawals even when the EigenLayer withdrawal has already matured. This allows matured loans to be "bought out" in a block they are already completable in, thus introducing unexpected behavior and possible loss of funds for the `oldOwner` through MEV.

Recommendation

```
block.number >= withdraw.startBlock + vault.delegationManager().withdrawalDelayBlocks()
```

Resolution

Rest Team: The issue was resolved in commit 373eed6.

ISU-6 | Users Can Avoid Withdrawal Fee

Category	Severity	Location	Status
Logical Error	● Medium	Issuance.sol: 56-91	Acknowledged

Description

When a user submits a withdrawal, either through direct assets or through EigenLayer, a fee is deducted. When using the Issuance contract, users have the possibility to complete other's withdraws by paying the due value and changing ownership of the pending withdrawal.

This mechanism, however, does not deduct any fees from the new owner, this results in:

- Disincentivizing anyone from initiating a direct withdrawal from EigenLayer and simply waiting for others to initiate the withdraw and changing it, so that they may not pay the fee
- The protocol does not receive fees for withdrawal done in this manner

Recommendation

Deduct the withdraw fee also on the `Issuance` contract when doing an `completeWithdrawEarly` function call.

Resolution

Rest Team: Acknowledged.

ISU-7 | Pending Withdrawals Can Be Bought When Eigen Is Paused

Category	Severity	Location	Status
Logical Error	● Medium	Issuance.sol: 55-91	Resolved

Description

Users that queue their withdraw on EigenLayer using the `Issuance` contract can have their withdrawals taken over by others as long as they are paid the equivalent withdraw amount in exchange.

This mechanism leaves a potential abuse situation when EigenLayer has withdrawal completion paused (`PAUSED_EXIT_WITHDRAWAL_QUEUE`) so that nobody would be able to call `completeQueuedWithdrawal` at that time.

During this time, users of the `Issuance` contract can take ownership of withdrawals that, unbeknown to them, can't be finalized at that time, basically buying into a blocked position.

Recommendation

Do not allow changing the owner of pending withdrawals if EigenLayer has withdrawal completion paused. Checking that withdrawal completion is paused can be done by calling the `Pausable.paused(uint8)` method with the `PAUSED_EXIT_WITHDRAWAL_QUEUE(2)` value.

Resolution

Rest Team: Resolved.

PENW-1 | DoS pendingWithdrawals

Category	Severity	Location	Status
DoS	● Medium	PendingWithdraw.sol: 20	Acknowledged

Description

There is no limit on the amount of pending withdrawals a user can have, nor is there a minimum amount of shares that ought to be redeemed per withdrawal. Therefore it can be economically viable to submit many withdrawals each with a single wei in order to expand the `pendingWithdraws` list for a malicious user.

As a result third party contracts interacting with the `PendingWithdraws.withdraws` function can be DoS'd simply because the amount of gas required to load the `pendingWithdraws` list into memory is greater than the block gas limit.

Recommendation

Consider adding either a limit on the amount of pending withdrawals a single user can have, or a minimum on the amount of shares necessary for a withdrawal to be queued or both.

Resolution

Rest Team: Acknowledged.

VLT-3 | Deposits Can Be Griefed

Category	Severity	Location	Status
Griefing	● Medium	Vault.sol: 136	Acknowledged

Description

In the depositIntoEigan function, a user can to deposit an arbitrary amount of vault assets into EigenLayer. This being permissionless, allows the protocol to gain yield more efficiently because it will not have to wait for a trusted party to move these funds.

The issue, however, is that the amount that is being deposited is not guaranteed to be available. If another user wanted to grief the protocol, they could do so by frontrunning a call to the depositIntoEigan function by calling the withdrawUsingAssets function. They could then withdraw just enough from the vault so that the other user's deposit call will fail.

Long term, this can pose a problem for the protocol because the way for users to gain yield through Rest is to have their funds deposited into EigenLayer. If a user can delay and reduce the efficiency with which these funds move to EigenLayer, the less yield the Rest users will receive.

Recommendation

Consider having a depositIntoEigan function which deposits the balanceOf(address(this)) instead of a specified amount. This will prevent any griefing of deposits.

Resolution

Rest Team: Acknowledged.

VLT-4 | Invalid Flagship LST Amount Transferred In

Category	Severity	Location	Status
Logical Error	● Medium	Vault.sol: 229	Resolved

Description

In the deposit(uint256 assets, address _receiver) function the assets amount passed to the super.deposit function is an ether value of the flagship asset amount. However the ERC4626 deposit function will transfer in the ether value amount of the flagship asset rather than the flagship asset amount.

The value of the amount transferred in is correctly converted to shares as the previewDeposit function is correctly overridden in the Vault contract, however the user still transfers in an unexpected amount of the flagship asset.

Consider the following example:

- Flagship asset price is 0.8 ether
- Bob calls deposit(1 * 1e18, address(bob))
- The vault transfers 0.8 * 1e18 of the flagship asset from Bob

This is unexpected for Bob as he specified 1 * 1e18 of the flagship asset to be transferred in.

Recommendation

Do not convert the specified assets amount to an ether amount in the deposit(uint256 assets, address receiver) function, as this conversion is already accounted for in the previewDeposit function.

Resolution

Rest Team: The issue was resolved in commit dcbb07b.

VLT-5 | Vault is not ERC4626 Compliant

Category	Severity	Location	Status
Compliance	● Medium	Vault.sol	Partially Resolved

Description

The `Vault` does not respect several requirements needed to be `ERC4626` compliant, although the internal documentation states that it is compliant:
/// @notice While this contract is fully ERC4626 compliant, it also has additional functionality
The non-compliance issues are:

- `withdraw` and `redeem` always revert; does not respect any of EIP requirements
- if the `flagshipToken` is removed from the vault via `removeLst` then `deposit` reverts. At this point `maxDeposit` must return 0, but it does not.
    - *MUST factor in both global and user-specific limits, like if deposits are entirely disabled (even temporarily) it MUST return 0.*
- `convertToShares` also reverts when this is not allowed
    - *MUST NOT revert unless due to integer overflow caused by an unreasonably large input.*
- `previewRedeem` and `previewWithdraw` do not take into consideration vault fees
    - *MUST be inclusive of withdrawal fees. Integrators should be aware of the existence of withdrawal fees.*

Recommendation

Modify the above issues to match the standard. Also do not allow the flagship asset to be removed from the allowed LST list.

Resolution

Rest Team: Partially Resolved.

VLT-6 | Attacker can Front-Run Removal of Asset

Category	Severity	Location	Status
Frontrunning	● Medium	Vault.sol	Acknowledged

Description

When an LST is removed with the `removeLst` function, the value of that LST in the system will no longer be attributed to the totalAssets.

Therefore a malicious actor may frontrun the removal of an LST and deposit that exact LST into the system before withdrawing that value in a different LST token, as a result the value of restEth will drop and holders will immediately lose the value of the removed LST tokens in the system.

If the protocol were to add the removed LST back it would create a positive stepwise jump in the value of restEth. This way an attacker could game the stepwise jump by depositing before the LST is added back and withdrawing afterwards for an immediate profit.

Recommendation

Introduce a mechanism for deposits of a certain LST to be paused, or deposits as a whole to be paused. Additionally, consider implementing validation that an LST cannot be removed unless the vault holds no value in the corresponding strategy and no balance of that LST directly in the vault contract. This prevents unlisted tokens from being lost to depositors.

Resolution

Rest Team: Acknowledged.

VLT-7 | Yield In Strategies Can Be Stolen

Category	Severity	Location	Status
Gaming	● Medium	Vault.sol	Acknowledged

Description

In the event that yield is distributed to a strategy in a single transaction, that yield amount can be vampire attacked by a malicious actor. The actor may deposit into the vault right before the reward is distributed, and then withdraw the gained funds with their Rest shares. These funds end up siphoned from the veritable vault depositors.

Recommendation

Ensure the vault withdrawal fee is large enough to deter a vampire attack such as this. Additionally be sure to implement a fee for the `completeWithdrawEarly` function so that the attacker cannot avoid the fee by using the Issuance contract.

Resolution

Rest Team: Acknowledged.

CONST-1 | Unresolved TODO

Category	Severity	Location	Status
Best Practices	● Low	Constants.sol: 22-25	Acknowledged

Description

In the Constants folder there is an unresolved TODO, which serves to warn to resolve the issue of the owner, fee receiver and starting withdrawal fee being 0.

Recommendation

Resolve the indicating TODO.

Resolution

Rest Team: Acknowledged.

GLOBAL-6 | No Incentive For Unsupported Eigen Assets To Use Rest

Category	Severity	Location	Status
Incentives	● Low	Global	Acknowledged

Description

Currently Rest supports 6 LSTs for deposit, cbETH (Coinbase), stETH (Lido), rETH (Rocket Pool), sfrxETH (Frax), LsETH (Liquid) and mETH. Out of these, the last 3 are currently not supported by EigenLayer.

At this point in time, there is no incentive for users of the Rest protocol to deposit any tokens that do not have a backing EigenLayer strategy. Also, users lose funds on withdraw due to the fees, while not gaining any yield on their deposit.

Recommendation

Clearly document this and do not add LSTs without EigenLayer strategies to the Vault.

Resolution

Rest Team: Acknowledged.

Guardian: If mETH is to be supported, the current constant points to the staking contract rather than mETH itself. As a result, the mETH token will not be able to be transferred to or from the vault, preventing it from being used in Rest entirely.

GLOBAL-7 | Use SafeERC20 For Token Operations

Category	Severity	Location	Status
Best Practices	● Low	Global	Unresolved

Description

In the EiganWithdrawlTracker._completeWithdraw and Issuance.completeWithdraw functions, upon sending funds to the user, the transfer function is used.

Though often the token used will not have a transfer implementation that returns false instead of reverting, out of an abundance of caution safeTransfer should be used.

Recommendation

Consider using safeTransfer instead of transfer.

Resolution

Rest Team: Unresolved.

GLOBAL-8 | Possible Trapped Funds When Using LST With Blacklist

Category	Severity	Location	Status
Best Practices	● Low	Global	Acknowledged

Description

Some LSTs such as cbEth have a blacklist functionality, however the Rest system does not confirm that a withdrawer is not blacklisted upon queuing a withdrawal from Eigenlayer with the queueWithdrawals function.

Therefore it is possible that withdrawals are queued which will attempt to send the LST tokens to blacklisted accounts upon completion of the withdrawal. These withdrawals will not be completable and the funds will be stuck.

Recommendation

Consider validating that the msg.sender is not blacklisted for the _asset in the withdrawUsingEigenShares function.

Resolution

Rest Team: Acknowledged.

GLOBAL-9 | Redundant Code

Category	Severity	Location	Status
Superfluous Code	● Low	Global	Partially Resolved

Description

There are several instances of duplicate, superfluous or redundant code in the project.

- `RestEthVault.addLst` uses the `isValidAsset` modifier instead of checking directly
- `RestEthVault.removeLst` has both the `isValidAsset` modifier and checks again redundantly. Remove the direct check
- `RestEthVault.withdrawUsingEigenShares` has both `hasStrategy` modifier and checks again redundantly. Remove the direct check

Recommendation

Implement the above mentioned changes.

Resolution

Rest Team: Partially Resolved.

.

GLOBAL-10 | Missing Input Validations

Category	Severity	Location	Status
Validation	● Low	Global	Acknowledged

Description

There are several locations throughout the codebase where validation for input is not done.

- `RestEthVault.constructor`
- `_strategyManager` and `_delegationManager` are not checked for zero address or for actually being the EigenLayer contracts.
- `flagshipAsset` token address not checked for `address(0)`
- `RestEthVault.addLst` or `RestEthVault._addLst`
- `_asset` is not checked for `address(0)`
- `RestEthVault.previewMint(address,uint256)` there is no `isValidAsset` modifier, although the `adapter[_asset].price()` execution reverts, it should be added for a clearer message

Recommendation

Add the above indicated validations.

Resolution

Rest Team: Acknowledged.

GLOBAL-11 | Lack of CEI

Category	Severity	Location	Status
Reentrancy	● Low	Global	Resolved

Description

In the Vault.withdrawUsingAssets function the assets are transferred before the restate amount is burned from the msg.sender. This allows the tx execution to be passed to an arbitrary address with an invalid state in the case of tokens with callbacks.

In the EiganWithdrawlTracker._completeWithdraw function the transfer call occurs before the removePending function is invoked. Thought there is no obvious path to exploitation, the pending withdrawal ought to be removed from storage before the transfer is made.

This avoids potentially passing tx execution to an arbitrary address with an invalid state in the case of tokens with callbacks.

Recommendation

1. Out of an abundance of caution, perform the safeTransfer at the end of the withdrawUsingAssets function.

2. Perform the transfer at the end of the _completeWithdraw function.

Resolution

Rest Team: Resolved.

ISU-8 | Pending Withdraws Not Cleared

Category	Severity	Location	Status
Best Practices	● Low	Issuance.sol: 94	Unresolved

Description

In the completeWithdraw function the pendingWithdraws are not cleared. There is no immediate risk, however the pendingWithdraw ought to be cleared for consistent accounting.

Recommendation

Clear the pendingWithdraw in the completeWithdraw function.

Resolution

Rest Team: Unresolved.

ISU-9 | New Owner Cannot Decide wantsETH Value

Category	Severity	Location	Status
Logical Error	● Low	Issuance.sol: 56	Acknowledged

Description

In the `completeWithdrawEarly` function the new owner does not get to change the value of `wantsEth` for the root, therefore if the new owner wishes to accept the underlying token as opposed to ether for an early withdrawal they do not have this optionality.

Recommendation

Consider allowing the new owner to change the `wantsEth` value in the event they would prefer a different option for an early withdrawal.

Resolution

Rest Team: Acknowledged.

ISU-10 | Potential Duplicate Of Withdrawal Funds

Category	Severity	Location	Status
Logical Error	● Low	Issuance.sol: 98	Resolved

Description

In the completeWithdraw function the underlying token is sent from the Issuance contract regardless of if the withdrawal was queued through the Issuance contract.

Therefore a user may queue a withdrawal by directly calling the withdrawUsingEigenShares function and complete it by calling the completeWithdraw function on the Issuance contract and receive tokens directly from the vault as well as from the Issuance contract, as long as the Issuance contract is holding enough balance of that token.

Recommendation

Only allow withdrawals that were queued through the Issuance contract to be completed through the Issuance contract.

Resolution

Rest Team: Resolved.

VLT-8 | Lack of Fee Basis Points Validation

Category	Severity	Location	Status
Validation	● Low	Vault.sol: 65, 90	Resolved

Description

The fee basis points has no validation when set, neither in the constructor nor in its dedicated setter. If set above 10,000 it will block all withdrawals because the subtraction when taking the fee out of the full amount would underflow

Recommendation

Validate when setting the fee basis points that it does not surpass 10,000.

Resolution

Rest Team: The issue was resolved in commit 23bfab1.

VLT-9 | Zero Address Fee Receiver Bricks Withdrawals

Category	Severity	Location	Status
Logical Error	● Low	Vault.sol: 164, 191	Resolved

Description

If the feeReceiver for the Vault contract is set to address(0), withdrawals will revert because the ERC20._mint function reverts when minting to address(0).

Recommendation

Do not allow setting a fee receiver as the zero address, both in the constructor or in the dedicated setFeeReceiver function.

Resolution

Rest Team: The issue was resolved in commit a23cfd4.

VLT-10 | Missing Events On Key State Changes

Category	Severity	Location	Status
Events	● Low	Vault.sol	Acknowledged

Description

Throughout the Vault contract there are instances where important contract changes were made but no event was emitted.

- Setting the fee receiver via `Vault.setFeeReceiver` or in the constructor
- Setting the fee BPS via `Vault.setFeeBasisPoints` or in the constructor
- Adding or removing a LST from the vault
- Minting shares via the function `Vault.mint` does not emit a Deposit event specific to this behavior

Recommendation

Emit events in all the mentioned locations.

Resolution

Rest Team: Acknowledged.

VLT-11 | Added LST Lacks Adapter Validation

Category	Severity	Location	Status
Validation	● Low	Vault.sol: 98-104	Partially Resolved

Description

When a LST token is added to the Vault contract, either via the constructor or the addLst function, an adapter must always exist and be associated with the corresponding LST token.

Currently there is no such check and mistakenly adding a token without an adaptor, or with an incorrect adaptor that would cause severe issues such as token mispricing. Furthermore, the strategy is written without validating that it is indeed compatible with that asset.

Recommendation

In the _addLst function, validate that the _adapater is not address(0) and that the IValueAdapter.asset equals the asset it will be mapped to. Also validate that the adapter and strategy provided are indeed compatible with the asset in the _addLst function.

Resolution

Rest Team: The issue was partially resolved in commit 7c3fe88.

VLT-12 | Risk Of Too Many LSTs

Category	Severity	Location	Status
Validation	● Low	Vault.sol: 82	Acknowledged

Description

In the EigenLayer StrategyManager contract there is a MAX_STAKER_STRATEGY_LIST_LENGTH of 32.

However in the `_addLst` function there is no validation that the amount of supported assets is less than the MAX_STAKER_STRATEGY_LIST_LENGTH.

Recommendation

Consider implementing a maximum of the MAX_STAKER_STRATEGY_LIST_LENGTH for the amount of LST tokens that can be added.

Resolution

Rest Team: Acknowledged.

VLT-13 | Single Step Ownership Transfer

Category	Severity	Location	Status
Centralization Risk	● Low	Vault.sol	Acknowledged

Description

The Vault uses a single-step ownership transfer, which poses a potential risk if ownership was transferred to the wrong address.

Recommendation

Consider using OpenZeppelin’s `Ownable2Step` contract.

Resolution

Rest Team: Acknowledged.

VLT-14 | Typo

Category	Severity	Location	Status
Typo	● Low	Vault.sol: 202	Unresolved

Description

In the `deposit` function, the `receiver` parameter is misspelled as `reciver`.

Recommendation

Replace `reciver` with `receiver`.

Resolution

Rest Team: Unresolved.

VLT-15 | Withdraw Using Assets Can Always Be Blocked

Category	Severity	Location	Status
Logical Error	● Low	Vault.sol: 180-192	Acknowledged

Description

Users of the Rest vault can opt to withdraw, not through EigenLayer but through funds directly available in the Vault. This option is provided with the `withdrawUsingAssets` function depending on token availability.

This option can be completely blocked by a user continuously spamming the `depositIntoEigan` function whenever funds are available in the vault, making the functionality potentially redundant.

Recommendation

Consider adding a minimum cooldown wait period between subsequent calls to `depositIntoEigan`, to allow users the chance to be able to withdraw the assets instantly. Otherwise, clearly document this behavior to users.

Resolution

Rest Team: Acknowledged.

VLT-16 | Missing Zero Fee Amount Check

Category	Severity	Location	Status
Validation	● Low	Vault.sol: 164	Acknowledged

Description

In the withdrawUsingEigenShares function, the fee amount is minted to the feeReceiver. However, with a small withdrawal amount, it is possible for the fee to be 0. In this case, the protocol will attempt to mint 0 tokens to the feeReceiver.

This is inconsistent with the protocol's design, as seen in the withdrawUsingAssets function, the fee amount is only minted if fee is greater than 0.

Recommendation

Implement the same check in the withdrawUsingEigenShares function that is in the withdrawUsingAssets function.

Resolution

Rest Team: Acknowledged.

VLT-17 | Fee Receiver Cannot Withdraw 100% of Amount

Category	Severity	Location	Status
Logical Error	● Low	Vault.sol	Resolved

Description

Whenever a withdrawal occurs, a percentage of the `restEthAmt` passed in by the caller will be designated as a `fee` for the `feeReceiver`. When the `feeReceiver` attempts to make their own withdraw using the collected fees, they will also have to pay the same `fee` to themselves. This leads to a situation where 100% of the lst cannot easily be withdrawn.

Recommendation

Consider not charging a fee if the `feeReceiver` is the one withdrawing.

Resolution

Rest Team: The issue was resolved in commit d93fb8e.

VLT-18 | User Can Withdraw Before Slashing

Category	Severity	Location	Status
Future Changes	● Low	Vault.sol	Acknowledged

Description

In the future when slashing logic is implemented in Eigenlayer and Rest Finance adopts delegation logic, a malicious user may observe that a particular delegated allocation is about to be slashed and frontrun the slashing transaction to withdraw from a separate strategy in order to avoid losing funds due to slashing.

Recommendation

Consider making withdrawals a two step action which requires time to pass, therefore no user may trivially frontrun a slashing transaction to protect themselves. Otherwise consider implementing some other solution in the future that would prevent users from avoiding the downsides of delegated stakes being slashed.

Resolution

Rest Team: Acknowledged.

VLT-19 | safeTransferFrom After External Call

Category	Severity	Location	Status
Reentrancy	● Low	Vault.sol: 288, 291	Acknowledged

Description

In the deposit(address _asset, uint256 assets, address receiver) and mint(address _asset, uint256 assets, address receiver) functions the safeTransferFrom call should take place before the restEth shares are minted, therefore it would not be possible to hand over tx execution to an arbitrary address in the event of a token with callbacks.

Recommendation

Perform the safeTransferFrom after the shares/assets are computed and before the restEth is minted.

Resolution

Rest Team: Acknowledged.

Disclaimer

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

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

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

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

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits