HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Venom
Date: 25 Sep, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Venom
Approved By	Marcin Ugarenko | Lead Solidity SC Auditor at Hacken OÜ
Type	Liquid Staking
Platform	Venom
Language	TON-Solidity
Methodology	Link
Website	https://venomstake.com
Changelog	25.08.2023 - Initial Review 18.09.2023 - Second Review 25.09.2023 - Third Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Table of Contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Venom (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

Venom Staking – a smart contract system designed for the EVER liquid staking. By aggregating real network validators into one platform, it ensures an automatic balance of staked EVER across these validators (achieved by Governance software system). This design eliminates the decision-making burden on users to select individual validators.

Furthermore, upon staking their EVER, users are issued a staked version of the SEVER token. This token representation can then be utilized across other DeFi platforms, allowing users to leverage the locked value of their stake.

In-scope contracts:

- *StEverVault* – This central contract oversees the majority of operations. It serves as the primary interface for user interactions, such as deposits and withdrawals, as well as administrative and governance tasks. Additionally, it manages strategy, emergency protocols, and validator interactions, drawing on the capabilities of several abstract contracts it inherits from.
  - *StEverStrategiesManager* – An abstract contract that enriches the `StEverVault` with functionalities for Cluster and DePool strategy management.
  - *StEverVaultBase* – This abstract contract establishes the foundation for the `StEverVault`. Extending the `StEverVaultStorage`, it incorporates a plethora of modifiers, utility functions, and external functions to ensure the vault operates seamlessly.
  - *StEverVaultEmergency* – An abstract contract designed to offer emergency features for users, especially when the Governance system becomes inactive and fails to process user withdrawal requests.
  - *StEverVaultStorage* – An abstract contract where all storage variables are defined and maintained.
  - *StEverVaultStrategiesController* – An abstract contract equipped with functionalities for deposits, withdrawals from strategies, and rewards processing. It also facilitates the Governance rebalancing process.
  - *StEverVaultValidators* – An abstract contract providing helper view functions. These are utilized to pre-verify the accuracy of Governance deposits and withdrawal requests.
- *StEverAccount* – A contract representing individual user accounts, recording their withdrawal requests.
- *StEverCluster* – The contract represents a Validator's account. Once validated, a dedicated contract is created for them. This contract facilitates the deployment of new strategies via the

DepoolStrategyFactory and manages the addition or removal of these strategies within the StEverVault.

- *StrategyDePool* – A contract bridging to the DePool contract, streamlining the integration of the staking process with DePool functionalities.
- *DepoolStrategyFactory* – A factory contract utilized by StEverCluster validators to deploy StrategyDePool contracts.

Roles

- StEverVault
  - User - Permitted to deposit and withdraw from the StEverVault. Additionally, the user can invoke an emergency state if the Governance system becomes inactive.
  - Owner - Has the authority to create new Cluster accounts for validators, approve strategies, and oversee the upgradability of contracts.
  - Governance - Tasked with processing withdrawal requests for users. It also manages the rebalancing of deposits among Validator strategies.
- StEverAccount
  - Vault - All functions inside the StEverAccount are executed by the StEverVault.
- StEverCluster
  - ClusterOwner - Empowered to deploy new strategies and add these deployed strategies to the StEverVault.
  - StEverOwner - Has the authority to remove a Cluster from the system, with an option to penalize the validator. Can also set the assurance limit value.
  - Vault - Validates responses to contract actions, ensuring they originate from the StEverVault.
- StrategyDePool
  - Vault - Ensures that function calls made to StrategyDePool are initiated by StEverVault.
  - DePool - Ensures that function calls made to StrategyDePool are initiated by DePool.
- DepoolStrategyFactory
  - Owner - Granted the capability to introduce new StrategyDePool code and upgrade existing strategies to the latest code.

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are in place.
- Technical description is robust.
- Exceptional NatSpec comments are provided, including functional and technical descriptions.

Code quality

The total Code Quality score is **10** out of **10**.

- Code is well-written and designed.
- The development environment is configured.

Test coverage

The coverage of the project with tests is about **80%**.

- The positive paths of major functionalities have been tested.
- Most of the "set" functionalities have been subjected to testing.
- Tests have been conducted on the view functions.
- There is sometimes a lack of coverage for negative cases.

Security score

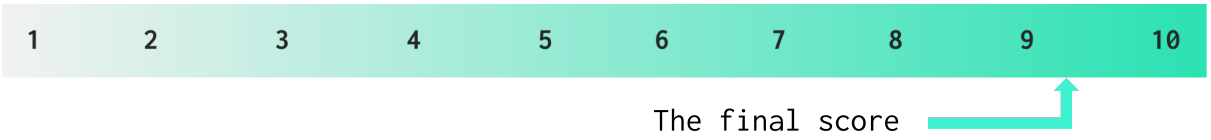
As a result of the audit, the code contains **no** security issues. The security score is **10** out of **10**.

All found issues are displayed in the Findings section of the report.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.3**.

The system users should acknowledge all the risks summed up in the Risks section of the report.



Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
25 August 2023	17	13	4	2
18 September 2023	2	3	2	0
25 September 2023	0	0	0	0

Risks

- Contracts sourced from the "*@broxus/contracts*" and "*broxus-ton-tokens-contracts*" from npm.js were not included in the scope of this audit.
- The withdrawal process from *StEverVault* has dependencies on specific user transaction settings, which the audited contract does not enforce. For a successful withdrawal:
    - Users must transfer SEVER tokens to StEverVault.
    - The `notify` setting must be set to `true`.
    - At least 3 EVER should be included as `msg.value` to cover gas costs.

  Not following these steps can result in a significant risk of losing SEVER tokens.
- The *deployStrategy()* function in *DepoolStrategyFactory* lacks access control, allowing any user to deploy a *StrategyDePool* contract. While this does not directly compromise the audited code, the emitted *NewStrategyDeployed* event could cause unexpected backend behaviors if not properly filtered.
- The system is highly centralized; each privilege role, if compromised, can lead to a loss of user funds. Multi-signature wallets for all privileged roles in the system should be used.
- The *startEmergencyProcess()* function lets users initiate a withdrawal if the centralized system fails, once the *timeAfterEmergencyCanBeActivated* value is reached after withdrawal request unlock time. This timeframe can be up to 365 days, creating potential delays and inconvenience.

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status
Integer Overflow and Underflow	All math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	It is recommended to use a recent version of the TON-Solidity compiler.	Passed
Access Control & Authorization	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Block values as a proxy for time	Block numbers should not be used for time calculations.	Not Relevant
Signature Reuse	Signed messages that represent an approval of an action should not be reusable.	Not Relevant
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Passed
Presence of Unused Variables	The code should not contain unused variables until they are not justified by design.	Passed
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed
Gas Management	Transaction execution costs should not depend dramatically on the amount of data stored in the contract. Contracts should validate that incoming value is enough to perform the whole call chain.	Passed
Compiler Warnings	The code should not force the compiler to throw warnings.	Passed
Style Guide Violation	Style guides and best practices should be followed.	Passed
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Test Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. The usage of contracts by multiple users should be tested.	Failed (Code Coverage)
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Findings

■■■■ Critical

C01. Transaction Replay Attack

Impact	High
Likelihood	High

In the *forceWithdrawFromStrategies()* function of the *StEverVaultStrategiesController* contract, the *availableAssets* and *totalAssets* variables are decreased after the for loop ends.

This implementation creates a flaw. Inside the for loop, checks ensure the *config.fee* value can be transferred (availableAssets is sufficient) and is not greater than totalAssets. These checks become invalid without a deduction during each loop iteration.

As a result of this oversight, the function accumulates the decrease amount in two memory variables and deducts them from *availableAssets* and *totalAssets* only after the loop, leading to potential underflow. Given the function's use of *tvm.accept()* for external messages and the possibility of a revert due to the underflow, a vulnerability arises for a Transaction Replay Attack.

This vulnerability could lead to a total depletion of the contract's EVER balance. The situation is worsened by the arbitrary nature of the config.fee, meaning there is no cap on potential loss.

Path:
./contracts/base/StEverVaultStrategiesController.tsol : forceWithdrawFromStrategies()

Recommendation: Move the decrements for *availableAssets* and *totalAssets* inside the loop to ensure accurate and timely deductions.

Valuable link: everscale.guide/smart_contracts/replay_protection

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

C02. Transaction Replay Attack

Impact	High
Likelihood	High

The *withdrawStEverFee()* function in the *StEverVault* contract is vulnerable to a Transaction Replay Attack.

The function invokes *tvm.accept()* prior to executing all *require()* checks. For instance, the function may revert due to insufficient *availableAssets*.

Though pragma *AbiHeader expire* is utilized, it does not provide protection as the validator is not restricted from replaying the transaction multiple times within a block.

Moreover, there are additional concerns:

1. The use of the *onlyGovernanceOrSelfAndAccept()* modifier is inappropriate; the correct modifier should be *onlyGovernanceAndAccept()*.
2. The comment for *tvm.rawReserve()* suggests that the fee should be paid by the admin. However, gas costs are deducted from the _amount value, leading to a potential out-of-gas error if msg.value is zero and _amount is minimal.
3. The owner's account usage for the *transfer()* function seems incorrect. The transfer should be made to the *governance* or an account provided as a parameter, use of the owner account indicates the function should employ the *onlyOwner()* modifier.

Path:
./contracts/StEverVault.tsol : withdrawStEverFee()

Recommendation: Discontinue using the onlyGovernanceOrSelfAndAccept() modifier. Implement an access control modifier named onlyGovernance() that doesn't utilize tvm.accept().

Correct the _amount deduction process to align with the comment or revise the comment to reflect the current behavior.

Ensure that the appropriate account (either governance or a specified parameter) receives the transfer, or employ the onlyOwner() modifier if the intended recipient is indeed the owner.

Valuable link: everscale.guide/smart_contracts/replay_protection

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

High

H01. Incorrect Function Used

Impact	High
Likelihood	Medium

In the *withdraw()* function of the StrategyDePool contract, there is a misuse of the *depositNotHandled()* function.

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Specifically, when the state is not in its *INITIAL* state, the function *depositNotHandled()* is triggered instead of the more appropriate *withdrawError()*.

This incorrect function call will place the Strategy in a non-Active state. The repercussion of this is that any subsequent deposits to the Strategy will be denied, leading to a Denial-of-Service (DoS) situation, where no more funds can be effectively deposited.

Path:
./contracts/StrategyDePool.tsol : withdraw()

Recommendation: Replace the incorrect function *depositNotHandled()* with the *withdrawError()* function.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

H02. Incorrect Gas Management

Impact	**High**
Likelihood	**Medium**

In the *emergencyWithdrawToUser()* function, defined in the *StEverVaultEmergency* contract, there is a mismatch between the required *msg.value* and the actual Gas needs during an emergency state.

The function is designed to operate on all pending user withdrawal requests. The assumption is that the *msg.value* should be calculated from pending withdrawal requests multiplied by 1 EVER, but this is not correctly enforced.

The current setup leads to the potential risk that if there is a maximum of 50 withdrawal requests and the last one fails, the entire recovery of withdrawal requests needs to be reset (*resetPendingValues()*). Under the present requirement of minimum 1 EVER, the contract will invariably face an out-of-gas scenario.

Path:
./contracts/StrategyDePool.tsol : emergencyWithdrawToUser()

Recommendation: The *msg.value* requirement should be updated to accurately reflect the number of pending withdrawals. As determining the exact number of withdrawal requests dynamically may be complex, a safer approach would be to assume a worst-case scenario and require a *msg.value* equivalent to the maximum number of possible withdrawal requests (e.g., 50 EVER for a max of 50 requests).

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

H03. Lack of Validation

Impact	High
Likelihood	Medium

The *setWithdrawHoldTimeInSeconds()* function in StEverVaultBase the contract, which can only be called by the contract owner, lacks a check on the maximum limit for the *withdrawHoldTime* value.

This can create a potential risk where if the owner sets a *withdrawHoldTime* greater than *TIME_AFTER_EMERGENCY_CAN_BE_ACTIVATED*, users will not be able to process their withdrawal requests.

This scenario can inadvertently or maliciously enable users to initiate an emergency process, forcing withdrawals from all strategies.

Path:
./contracts/base/StEverVaultBase.tsol :
setWithdrawHoldTimeInSeconds()

Recommendation: It is essential to introduce a safety check ensuring that *withdrawHoldTime* cannot be set to a value greater than *TIME_AFTER_EMERGENCY_CAN_BE_ACTIVATED*. This will prevent potential manipulation or inadvertent mistakes that can lock user funds or trigger unintended contract behaviors.

Found in: b293dd8

Status: Fixed (Revised commit: bb9497b)

H04. Data Consistency

Impact	High
Likelihood	Medium

The *setStrategiesTotalAssets()* function in the *StEverVaultStrategiesController* contract carries with it considerable risks tied to the manipulation of *totalAssets* for individual strategies. Such operations have the potential to cascade into unintended consequences and even result in the loss of funds.

Two primary concerns have been identified:

1. The use of *Constants.INCREASE_STRATEGY_TOTAL_ASSETS_CORRECTION* is not only unclear but also seems unnecessary, possibly leading to inaccurate accounting of assets.
2. If an admin reduces *totalAssets* for a strategy to 0, and subsequently, the strategy is removed while a pending withdrawal process exists, there can be complications. The DePool, during *onRoundComplete()*, might not be able to send the

withdrawal to *StEverVault*, causing loss of funds. Moreover, a more concerning issue is the incorrect accounting of *totalAssets* in the StEverVault. representation. Given the contract's invariant that *StEverVault.totalAssets* should always be greater than or equal to the combined *totalAssets* of all the strategies, any modification in *strategies[strategy].totalAssets* mandates a corresponding adjustment in *totalAssets*. This is crucial to maintain the data's integrity and accuracy.

Path:
./contracts/base/StEverVaultStrategiesController.tsol : setStrategiesTotalAssets()

Recommendation: Given that the function is labeled as a migration function, the safest course of action would be its complete removal from the codebase.

If retaining this function is essential for some specific workflows, consider the following modifications:

- Clarify the role of Constants.INCREASE_STRATEGY_TOTAL_ASSETS_CORRECTION in the function. If its utility is not apparent or it is found to be superfluous, it should be eliminated from the function.
- It is paramount to ensure that any alterations made to the assets of individual strategies are duly reflected in the overarching *totalAssets*. This is crucial not just for consistency, but to maintain accurate data representation throughout the system.

Found in: b293dd8

Status: Fixed (Revised commit: bb9497b)

Medium

M01. Inconsistent Ownership Control

Impact	Medium
Likelihood	Medium

In the *_init()* function within the *StEverCluster* contract, the *stEverOwner* variable is set based on the provided input of the current owner of the StEverVault, and cannot be changed after.

However, there is a potential inconsistency in the management of ownership between the StEverCluster contract and the StEverVault.

When the vault owner is changed in StEverVault using the *transferOwnership()* function, this change is not mirrored or updated in the StEverCluster. As a result, functionalities in the

StEverCluster that are governed by ownership will not recognize the new owner from the StEverVault.

This misalignment can lead to unexpected behaviors, reduced control, or potential deadlocks where the recognized owner is unable to invoke necessary functions.

Path:
./contracts/StEverCluster.tsol : _init(), onlyStEverOwner(), stEverOwnerOrClusterOwner()

Recommendation: Update the access control mechanism for the StEverCluster contract.

Specifically, all functions that are currently guarded by the *onlyStEverOwner()* or *stEverOwnerOrClusterOwner()* modifier should now be protected by a new *onlyVault()* modifier.

These functions should be callable only by the *StEverVault* contract.

Ensure that this allows the owner of the StEverVault to execute those functions indirectly, preserving the intention of centralized control while resolving the ownership inconsistency.

Found in: b293dd8

Status: Fixed (Revised commit: bb9497b)

M02. Missing Validation

Impact	High
Likelihood	Low

In the *transferOwnership()* function within the *StEverVaultBase* contract, the new owner's address is set directly from the input parameter *_newOwner* without validating its legitimacy. This means that a careless call to this function can set the ownership of the contract to the zero address.

A critical point to consider, especially in the TVM environment, is that transactions can be constructed with a zero address as the sender using an external message.

Setting the owner to the zero address can lead to irreversible consequences, making all functionalities that rely on the *onlyOwner()* modifier widely open. This poses a severe risk and can lead to loss of all funds.

Path:
./contracts/base/StEverVaultBase.tsol : transferOwnership()

Recommendation: Before setting the *owner* variable, add a validation check to ensure that *_newOwner* is not the zero address.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

M03. Missing Validation

Impact	High
Likelihood	Low

In the *transferGovernance()* function of the *StEverVaultBase* contract, the new *governance* address is directly updated from the *_newGovernance* input parameter without any checks to validate its legitimacy.

A critical point to consider, especially in the TVM environment, is that transactions can be constructed with a zero address as the sender using an external message.

Functions or operations reliant on the *governance* role could become without any access control, possibly leading to irreversible crucial contract operations.

Path:
./contracts/base/StEverVaultBase.tsol : transferGovernance()

Recommendation: Before updating the *governance* variable, incorporate a validation check to ensure *_newGovernance* is not the zero address.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

M04. Implementation Error

Impact	Low
Likelihood	High

In the *_reserve()* function of the *StEverCluster* contract, the computation to calculate the needed reserved amount for *rawReserve()* calls leverages the *StEverAccountGas.CONTRACT_MIN_BALANCE* value. However, it appears that the correct value to reference should be *ClusterGas.CONTRACT_MIN_BALANCE*.

Using an inaccurate reference for the minimum balance can lead to potential discrepancies in the reserve calculation. This might inadvertently cause the contract to reserve an incorrect amount of balance, potentially leading to functionality disruptions or failures in subsequent operations that depend on the reserved value.

In the *minCallValue()* modifier of the given contract the similar issue occurs, the constant *StEverVaultGas.MIN_CALL_MSG_VALUE* is used to check the minimum required msg.value. However, it appears that the

incorrect constant is being used and it should be *ClusterGas.MIN_CALL_VALUE* instead.

Path:
./contracts/StEverCluster.tsol : _reserve(), minCallValue()

Recommendation: Update the *_reserve()* function to utilize the *ClusterGas.CONTRACT_MIN_BALANCE* value in place of *StEverAccountGas.CONTRACT_MIN_BALANCE*.

Update the *minCallValue()* function to utilize the *ClusterGas.MIN_CALL_VALUE* value in place of *StEverVaultGas.MIN_CALL_MSG_VALUE*.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

M05. Incorrect Gas Management

Impact	**Medium**
Likelihood	**Medium**

In the *deployStrategies()* function of the *StEverCluster* contract, there is a calculation for the required message value to deploy strategies based on the number of _dePools provided.

While the calculation multiplies the ClusterGas.STRATEGY_DEPLOY_VALUE and ClusterGas.MIN_CALL_VALUE by the length of _dePools, it does not account for the additional Gas required for executing the *deployStrategies()* function itself.

The omission of this additional Gas reservation could lead to out-of-gas errors during the function's execution.

Path:
./contracts/StEverCluster.tsol : deployStrategies()

Recommendation: Update the Gas calculation to include an extra *ClusterGas.MIN_CALL_VALUE* to account for the Gas consumed by the *deployStrategies()* function.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

M06. Race Condition

Impact	High
Likelihood	Low

In the *StEverVault* contract, functions *requestWithdraw()* and *onPendingWithdrawAccepted()* exhibit a vulnerability when they operate in an asynchronous TVM environment due to their reliance on user-provided nonces.

The *requestWithdraw()* function facilitates the creation of a *PendingWithdraw* entry using a user-provided nonce. This means if another user provides the same nonce (intentionally or unintentionally) before the first user's transaction is completed, the initial user's withdrawal request can be overridden. While this does not directly lead to fund loss, it can potentially force the contract into an emergency state or create unexpected behaviors.

In the *onPendingWithdrawAccepted()* function, due to the asynchronous nature of calls in TVM, a scenario can arise where calls from two different users (A and B) are interleaved. In this situation, the acceptance of user A's withdrawal can be overridden by user B's subsequent calls. This can lead to the withdrawal event for user A being missed, and the Governance not processing the withdrawal.

Path:
./contracts/StEverVault.tsol : requestWithdraw(), onPendingWithdrawAccepted()

Recommendation: There are multiple ways to solve the problem:

- Instead of relying on user-provided nonces, consider implementing an internal nonce management mechanism to ensure uniqueness and protect against overrides.
- Implement a locking mechanism to prevent concurrent access to PendingWithdraw data associated with the same nonce. This can prevent unwanted overrides in an asynchronous environment.
- Add validation checks to ensure that a PendingWithdraw associated with a given nonce exists and belongs to the expected user before processing it.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

M07. Denial of Service

Impact	High
Likelihood	Low

The *onClusterRemoved()* function in the *StEverCluster* contract, which is designed to handle the removal of clusters associated with a particular *_clusterOwner*, contains a problematic operation.

When all clusters for a given _clusterOwner are removed, it deletes the entire _clusterOwner entry from clusterPools.

However, this deletion inadvertently resets the *currentClusterNonce* for the _clusterOwner. The implications are severe, as nonces determine the address for deploying new clusters. If the nonce is reset, any subsequent deployment for the _clusterOwner will occur at an address that has previously been used.

This could lead to unexpected behaviors, vulnerabilities, and even lock the _clusterOwner from acquiring new clusters in the future.

Path:
./contracts/base/StEverStrategiesManager.tsol : onClusterRemoved()

Recommendation: Retain the *currentClusterNonce* value even when clearing other data fields related to the *_clusterOwner*. This can ensure that every new deployment will always use a unique nonce and, by extension, a fresh address.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

M08. Missing Validation

Impact	High
Likelihood	Low

Two functions *StEverCluster.handleDelegateStrategies()* and *StEverStrategiesManager.delegateStrategies()* are used to manage strategy delegation.

These functions present issues that could lead to unintended behaviors or inconsistencies in the state of the contract.

In *handleDelegateStrategies()*, when looping through *_strategies*, only those strategies not already present in *strategies* are considered. The problem arises when a strategy is already part of the cluster. In such a case, there is no mechanism to notify the caller or make any adjustments. This means a strategy can indefinitely remain in the *TRANSFERRING* state.

In *delegateStrategies()*, the input *_strategies* size is arbitrary and lacks batching, potentially causing block size or Gas limit issues.

The *handleDelegateStrategies()* function misses validation checks on *maxStrategiesCount* and *Assurance* size.

In *delegateStrategies()*, there is an absence of a mechanism to revert or adjust the operation if not all strategies are accepted by the destination cluster.

Paths:
./contracts/base/StEverStrategiesManager.tsol : delegateStrategies()
./contracts/StEverCluster.tsol : handleDelegateStrategies()

Recommendation: Implement a revert mechanism or callback to the original caller, ensuring they are informed about any strategies not processed. This can help in rectifying the state of those strategies or making further decisions.

Implement a batching mechanism or set a maximum limit to _strategies size. This will ensure the function does not hit block or Gas limitations, especially when working with a large number of strategies.

Introduce checks that validate the length of _strategies against maxStrategiesCount and Assurance size. This will ensure that the cluster does not exceed its allowed number of strategies and maintains the expected assurance size.

Found in: b293dd8

Status: Fixed (Revised commit: bb9497b)

M09. Invalid Validation

Impact	Low
Likelihood	High

The *onRoundComplete()* function in the *StrategyDePool* contract conducts a balance check to decide the need for additional funds. It does this by checking if the contract's balance is below *THRESHOLD_BALANCE* to take certain actions.

However, the function does not account for the fact that *msg.value* is part of *address(this).balance* when performing the check. This leads to an inaccurate assessment of the contract's original balance before the function's transaction took place. This discrepancy is especially pronounced when *msg.value* carries withdrawal from DePool.

Path:
./contracts/StrategyDePool.tsol : onRoundComplete()

Recommendation: Subtract *msg.value* from *address(this).balance* to ensure an accurate representation of the contract's balance.

Found in: b293dd8

Status: Fixed (Revised commit: bb9497b)

M10. Division By Zero

Impact	High
Likelihood	Low

The *setFullUnlockRewardSeconds()* function in the *StEverVaultBase* contract allows the owner to change the *fullUnlockSeconds* value. However, it lacks validation to prevent the value from being set to 0.

A zero value for *fullUnlockSeconds* will subsequently cause a division by zero issue in the *StEverVaultStrategiesController.strategyReport()* function. This could lead to a denial of service (DoS) scenario, making it impossible for the many functions to execute correctly, which can disrupt the normal operations of the contract.

Path:
./contracts/base/StEverVaultBase.tsol : setFullUnlockRewardSeconds()

Recommendation: It is crucial to introduce a validation check ensuring that fullUnlockSeconds cannot be set to 0. This will prevent potential disruptions and safeguard against unintentional mistakes that can compromise contract operations.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

M11. Incorrect Gas Management

Impact	High
Likelihood	Low

The *processSendToUsers()* function in the *StEverVault* contract does not adequately check the size of the *config.nonces* array. There is a potential for the array to have a size larger than the intended limit, specifically greater than *MAX_PENDING_COUNT*, which is set to 50.

When calculating the *unusedIterationFee* and the value parameter for the *IStEverAccount(account).processWithdraw* function call, the code multiplies the length of config.nonces by respective fee constants. If the length of config.nonces exceeds MAX_PENDING_COUNT, this can lead to unanticipated behavior, including an incorrect fee calculation.

Path:
./contracts/StEverVault.tsol : processSendToUsers()

Recommendation: Before using the length of config.nonces, check whether its length exceeds MAX_PENDING_COUNT. If it does, handle it accordingly by limiting the *unusedIterationFee* to (MAX_PENDING_COUNT - 1) * FEE_FOR_WITHDRAW_TO_USER_ITERATION and value of the *IStEverAccount(account).processWithdraw* to MAX_PENDING_COUNT * WITHDRAW_FEE + unusedIterationFee.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

M12. Invalid Validation

Impact	Low
Likelihood	High

In the *withdrawExtraMoney()* function of the *StrategyDePool* contract, the contract's balance is evaluated to determine if it exceeds the *MAX_BALANCE*. However, the current balance calculation mistakenly includes *msg.value*, leading to potential misinterpretations of the available balance.

Path:
./contracts/StrategyDePool.tsol : withdrawExtraMoney()

Recommendation: Adjust the balance calculation by subtracting *msg.value* from *address(this).balance* to accurately represent the contract's original balance.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

M13. Missing Validation

Impact	High
Likelihood	Low

In the *setStrategyFactory()* function of the StEverVaultBase contract, the new *strategyFactory* address is directly updated from the *_strategyFactory* input parameter without any checks to validate its legitimacy.

A critical point to consider, especially in the TVM environment, is that transactions can be constructed with a zero address as the sender using an external message.

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Functions or operations reliant on the *strategyFactory* role could become without any access control, possibly leading to irreversible crucial contract operations.

Path:
./contracts/base/StEverVaultBase.tsol : setStrategyFactory()

Recommendation: Before updating the *strategyFactory* variable, incorporate a validation check to ensure *_strategyFactory* is not the zero address.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

Low

L01. Floating Pragma

Impact	Low
Likelihood	Medium

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Paths:
./contracts/*

Recommendation: Consider locking the pragma version whenever possible and avoid the usage of a floating pragma in the final deployment.

Found in: b293dd8

Status: Mitigated (The *0.62.0* version has no disclosed bugs related to the code.)

L02. Outdated Compiler Version

Impact	Low
Likelihood	Medium

The *0.62.0* version of the TON-Solidity Compiler is used, however, it is considered to be outdated. Newer compiler versions contain fixes for various issues.

Paths:
./contracts/*

Recommendation: Use the relevant compiler version.

Found in: b293dd8

Status: Mitigated (The *0.62.0* version has no disclosed bugs related to the code.)

L03. Storage Variable Shadowing

Impact	Low
Likelihood	Low

In the *onCodeUpgrade()* function in the StEverAccount contract, the local variables *vault*, *user* and *currentVersion* shadow the storage variables with the same names.

In the *getDepositStEverAmountFor()* function in the StEverVaultBase contract, the local variable *effectiveEverAssets* shadows the storage variable with the same name.

In the *getWithdrawEverAmountFor()* function in the StEverVaultBase contract, the local variable *effectiveEverAssets* shadows the storage variable with the same name.

In the *onPendingWithdrawRemoved()* function in the StEverVault contract, parameter name *nonce* shadows the storage variable with the same name.

In the *getAndCheckWithdrawToUserInfo()* function in the StEverVaultBase contract, the local variable *nonce* shadows the storage variable with the same name.

This can lead to confusion and potential mistakes when trying to access or modify them.

Paths:
./contracts/StEverAccount.tsol : onCodeUpgrade()
./contracts/base/StEverVaultBase.tsol : getDepositStEverAmountFor(), getWithdrawEverAmountFor(), getAndCheckWithdrawToUserInfo()
./contracts/StEverVault.tsol : onPendingWithdrawRemoved()

Recommendation: It is advisable to rename the local variables by using _ (underscore) as a prefix.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

L04. Missing Validation

Impact	Low
Likelihood	Low

The *setGainFee()* function in the *StEverVaultBase* contract allows the owner to change the *gainFee* value. However, there is a missing validation check for the _gainFee value. As per the comment, there is a requirement that 1 ever is the minimum for _gainFee, but the function does not enforce this constraint.

Path:
./contracts/base/StEverVaultBase.tsol : setGainFee()

Recommendation: Implement a validation check to ensure that _gainFee is greater than or equal to the minimum threshold of 1 ever.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

L05. Incorrect Gas Management

Impact	Low
Likelihood	Low

The calculation for *requiredMsgValue* in the *startEmergencyProcess()* function should also consider the increase by *MIN_CALL_VALUE* on top of the calculated value.

Path:
./contracts/base/StEverVaultEmergency.tsol : startEmergencyProcess()

Recommendation: Adjust the calculation of requiredMsgValue to include MIN_CALL_VALUE in addition to the current calculations.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

L06. Missing Validation

Impact	Medium
Likelihood	Low

In the *constructor()* of the *StEverVault* contract, the *gainFee* variable is set directly from the _gainFee input parameter without any checks to validate its legitimacy. There is a requirement that 1 ever is the minimum for *gainFee*, but the constructor does not enforce this constraint.

In the *constructor()* of the *StEverVault* contract, the *stEverFeePercent* is directly set from the _stEverFeePercent input parameter without validation checks. There is a requirement that *stEverFeePercent* should be less than or equal to 1000, but the constructor does not enforce this constraint.

Path:
./contracts/StEverVault.tsol : constructor()

Recommendation: Introduce a validation check to ensure that _gainFee is greater than or equal to the minimum threshold of 1 ever.

Introduce a validation check to ensure that _stEverFeePercent is less than or equal to 1000 to prevent setting an invalid fee percentage.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

L07. Missing Validation

Impact	Low
Likelihood	Low

In the *createCluster()* function of the *StEverStrategiesManager* contract, a new cluster can be deployed potentially with an unset *strategyFactory*. If *strategyFactory* is address(0), then the deployed cluster will not be able to create any strategy.

This can lead to a malfunction in the contract's intended behavior.

Path:
./contracts/base/StEverStrategiesManager.tsol : createCluster()

Recommendation: Introduce a validation check before deploying the cluster to ensure that strategyFactory is not address(0).

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

L08. Funds Lock

Impact	Low
Likelihood	Medium

In the *onPendingWithdrawRejected()* and the *onPendingWithdrawRemoved()* functions of the *StEverVault* contract, the *WITHDRAW_FEE* is reserved to be returned to the user.

However, the *FEE_FOR_WITHDRAW_TO_USER_ITERATION* should also be returned to the user, but it is currently not being handled in the function.

This could lead to users not receiving the full amount they are entitled to when a withdrawal request is rejected.

Path:
./contracts/StEverVault.tsol : onPendingWithdrawRejected(), onPendingWithdrawRemoved()

Recommendation: Update both functions to also reserve and return the FEE_FOR_WITHDRAW_TO_USER_ITERATION to the user in addition to the WITHDRAW_FEE.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

L09. Incorrect Gas Management

Impact	Medium
Likelihood	Low

In the *upgrade()* function of the *DepoolStrategyFactory* contract, there is no enforcement on the minimum msg.value required to call this function.

Given the Gas considerations and operations within the function, it is crucial to ensure that the caller provides a sufficient amount of Gas.

Path:
./contracts/DepoolStrategyFactory.tsol : upgrade()

Recommendation: Integrate the *minCallValue()* modifier to the upgrade function to enforce a minimum msg.value for the call.

Found in: b293dd8

Status: Fixed (Revised commit: bb9497b)

L10. Potential Underflow

Impact	Medium
Likelihood	Low

In the *onStrategyHandledWithdrawRequest()* function of the *StEverVaultStrategiesController* contract, the *HANDLING_STRATEGY_CB_FEE* is deducted from *msg.value* when updating the *availableAssets* and *totalAssets*. However, if *config.fee* is an arbitrary low value, it could cause msg.value - StEverVaultGas.HANDLING_STRATEGY_CB_FEE to underflow.

In the *onStrategyDidntHandleDeposit()* function of the *StEverVaultStrategiesController* contract, the *HANDLING_STRATEGY_CB_FEE* is deducted from *msg.value* when updating the

availableAssets. There is a potential for underflow in edge cases, for example, if minStrategyDepositValue is set to 0 and depositConfig.fee is very low.

Path:
./contracts/base/StEverVaultStrategiesController.tsol : onStrategyHandledWithdrawRequest(), onStrategyDidntHandleDeposit()

Recommendation: Ensure the deduction of *HANDLING_STRATEGY_CB_FEE* occurs in the *processWithdrawFromStrategies()* function, or introduce a validation check before the subtraction to prevent potential underflow.

It is advisable to ensure that the deduction of HANDLING_STRATEGY_CB_FEE is handled safely in the *onStrategyDidntHandleDeposit()* function. If this fee is indeed necessary, consider attaching this value in the depositToStrategies() function and decrease or attach the fee there accordingly.

Found in: b293dd8

Status: Fixed (Revised commit: bb9497b)

L11. Incorrect Gas Recipient

Impact	Medium
Likelihood	Low

In the *onStrategyRemoved()* function of the *StEverCluster* contract, the remaining Gas is always transferred to *clusterOwner*. However, the *removeStrategies()* function can also be called by *stEverOwner*.

This discrepancy could lead to unintended behavior where the wrong entity receives the remaining Gas.

Path:
./contracts/StEverCluster.tsol : onStrategyRemoved()

Recommendation: Ensure that the remaining Gas is transferred to the correct entity. This can be achieved by passing the correct recipient information through the chain-of-calls using a _sendGasTo parameter or another appropriate mechanism.

Found in: b293dd8

Status: Mitigated (with Customer notice:

"*This is a centralized protocol. We are whitelisting cluster owners and we are ready to pay a fee in case the strategy is deleted by us.*")

L12. Incorrect Gas Management

Impact	Low
Likelihood	Low

In the *withdrawAssurance()* function of the *StEverCluster* contract, there is no enforcement on the minimum *msg.value* required to call this function.

If the cluster owner provides too low a msg.value, they can potentially lose their SEVER.

Path:
./contracts/StEverCluster.tsol : withdrawAssurance()

Recommendation: Integrate the *minCallValue()* modifier to the *withdrawAssurance()* function to enforce a minimum msg.value for the call.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

L13. Incorrect Gas Management

Impact	Low
Likelihood	Low

In the *upgradeStEverAccount()* function of the *StEverVaultBase* contract, the *minCallValue()* modifier is used to enforce a minimum msg.value required to call this function. However, the same amount is forwarded to the upgrade function, which could lead to insufficient funds for the entire operation.

In the *upgradeStEverCluster()* function of the *StEverVaultBase* contract, the attached value (msg.value) is checked to be at least 1 EVER due to the minCallValue modifier. However, this might not be sufficient because the same amount is forwarded to the upgrade function of the cluster. This can lead to potential out-of-gas issues and disrupt the function's intended behavior.

Path:
./contracts/base/StEverVaultBase.tsol : upgradeStEverAccount(), upgradeStEverCluster()

Recommendation: Increase the enforced msg.value or adjust the amount forwarded to ensure that the entire operation can be funded.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

L14. Potential Underflow

Impact	Medium
Likelihood	Low

In the *receiveExtraMoneyFromStrategy()* function of the *StEverVaultStrategiesController* contract, the calculation of *receivedValue* can potentially underflow, leading to unintended consequences.

This can happen, for instance, when Gas forwarded from *withdrawExtraMoney()* function is below 0.03 ever.

Path:
./contracts/base/StEverVaultStrategiesController.tsol : receiveExtraMoneyFromStrategy()

Recommendation: Implement countermeasures similar to those used in the availableAssetsIncreasedFor calculations to prevent potential underflows. Ensure that the subtraction does not result in negative values, or consider using a safe subtraction function.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

L15. Incorrect Gas Management

Impact	Low
Likelihood	Medium

In the *withdrawExtraEver()* function of the *StEverVault* contract, the call to *tvm.rawReserve()* can cause an out-of-gas exception in scenarios where msg.value is 0 and totalExtraEver is too low. This can lead to unexpected behavior and hinder the intended execution of the function.

Path:
./contracts/StEverVault.tsol : withdrawExtraEver()

Recommendation: Use the *minCallValue()* modifier for this function to ensure that there is a minimum attached msg.value to prevent potential out-of-gas issues. Properly managing the Gas reservation is essential to ensure the function's correct execution.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

L16. Redundant Bounce Flag Set

Impact	Low
Likelihood	Low

In the *upgradeStEverCluster()* function of the *StEverVaultBase* contract, the *bounce* flag is set to *true* when calling the upgrade function of the IStEverCluster contract. However, this is redundant since the *onBounce()* of StEverVault will not process this, potentially leading to unexpected behavior.

In the *upgradeStEverAccount()* function of the *StEverVaultBase* contract, the *bounce* flag is set to *true* when calling the upgrade function of the IStEverAccount contract. However, this is redundant since the *onBounce()* of StEverVault will not process this, potentially leading to unexpected behavior.

In the *_upgradeStEverAccounts()* function of the *StEverVaultBase* contract, the bounce flag is set to true by default when making a call to IStEverAccount(userData).upgrade. However, this is redundant since the onBounce() function of StEverVault does not process such bounces.

In the *_upgradeStEverClusters()* function of the *StEverVaultBase* contract, the bounce flag is set to true by default when making a call to IStEverCluster(clusterAddress).upgrade. This is redundant since the onBounce() function of StEverVault does not process such bounces.

Path:
./contracts/base/StEverVaultBase.tsol : upgradeStEverCluster(), upgradeStEverAccount(), _upgradeStEverAccounts(), _upgradeStEverClusters()

Recommendation: Consider explicitly setting the bounce flag to false to avoid unnecessary Gas usage and potential confusion in the future.

Found in: b293dd8

Status: Mitigated (with Customer notice:

"*This is not really redundant. If an error occurs vault will receive the remaining value, otherwise an onBounce phase won't come so we won't spend any additional value.*")

L17. Incorrect Gas Management

Impact	Low
Likelihood	Low

In the *forceStrategyRemove()* function of the *StEverVaultStrategiesController* contract, the attached value (msg.value) is checked to be at least *StEverVaultGas.REMOVE_STRATEGY_RESERVE*.

However, this might not be sufficient as the same amount is passed to the *onStrategyRemoved()* function of the IStEverCluster contract, and this function might need additional Gas for its operations.

Path:
./contracts/base/StEverVaultStrategiesController.tsol : forceStrategyRemove()

Recommendation: Ensure that the attached msg.value is sufficiently large to cover the Gas costs of both the current function and the subsequent onStrategyRemoved() function.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

Informational

I01. Redundant Modifier

The *onlyGovernanceAndAccept()* modifier in the *StEverVaultBase* contract is designed to ensure that only the Governance account can call certain functions and to accept any incoming funds. However, the modifier is not utilized anywhere in the contract.

The *adminOrClusterOwner()* modifier in the *StEverVaultBase* contract is intended to ensure that either the Owner or the ClusterOwner (identified by _clusterId) can call certain functions. However, the modifier appears to be redundant and not utilized in the contract.

Path:
./contracts/base/StEverVaultBase.tsol : onlyGovernanceAndAccept(), adminOrClusterOwner()

Recommendation: It is advisable to remove the redundant modifiers from the contract to reduce redundancy and improve code clarity. Before removal, ensure that there are no future plans to use those modifiers.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

I02. Redundant Constant Variable

A constant variable, DEPOSIT_FEE, is defined in the library StEverVaultGas but is not used elsewhere in the contracts. This can lead to confusion and unnecessary clutter in the codebase.

A constant variable, UPGRADE_VALUE, is defined in the library ClusterGas but is not used elsewhere in the contracts. This can lead to confusion and unnecessary clutter in the codebase.

Constant variables INITIAL_AVAILABLE_ASSETS, DEPLOY_VAULT_FEE and EMERGENCY_DURATION, are defined in the library Constants but are not used elsewhere in the contracts. This can lead to confusion and unnecessary clutter in the codebase.

Constant variables STATUS_SUCCESS, STATUS_STAKE_TOO_SMALL, STATUS_DEPOOL_CLOSED, STATUS_NO_PARTICIPANT, STATUS_PARTICIPANT_ALREADY_HAS_VESTING, STATUS_WITHDRAWAL_PERIOD_GREATER_TOTAL_PERIOD, STATUS_TOTAL_PERIOD_MORE_18YEARS, STATUS_WITHDRAWAL_PERIOD_IS_ZERO, STATUS_TOTAL_PERIOD_IS_NOT_DIVISIBLE_BY_WITHDRAWAL_PERIOD, STATUS_REMAINING_STAKE_LESS_THAN_MINIMAL, STATUS_PARTICIPANT_ALREADY_HAS_LOCK, STATUS_TRANSFER_AMOUNT_IS_TOO_BIG, STATUS_TRANSFER_SELF, STATUS_TRANSFER_TO_OR_FROM_VALIDATOR, STATUS_FEE_TOO_SMALL, STATUS_INVALID_ADDRESS, STATUS_INVALID_DONOR, STATUS_NO_ELECTION_ROUND, STATUS_INVALID_ELECTION_ID, STATUS_TRANSFER_WHILE_COMPLETING_STEP, STATUS_NO_POOLING_STAKE and STATUS_NOT_ALLOWED_PARTICIPANT, are defined in the StrategyDePool contract but are not used elsewhere in the contracts. This can lead to confusion and unnecessary clutter in the codebase.

Paths:
./contracts/utils/Gas.tsol : DEPOSIT_FEE, UPGRADE_VALUE
./contracts/utils/Constants.tsol : INITIAL_AVAILABLE_ASSETS, DEPLOY_VAULT_FEE, EMERGENCY_DURATION
./contracts/StrategyDePool.tsol : STATUS_SUCCESS, STATUS_STAKE_TOO_SMALL, STATUS_DEPOOL_CLOSED, STATUS_NO_PARTICIPANT, STATUS_PARTICIPANT_ALREADY_HAS_VESTING, STATUS_WITHDRAWAL_PERIOD_GREATER_TOTAL_PERIOD, STATUS_TOTAL_PERIOD_MORE_18YEARS, STATUS_WITHDRAWAL_PERIOD_IS_ZERO, STATUS_TOTAL_PERIOD_IS_NOT_DIVISIBLE_BY_WITHDRAWAL_PERIOD, STATUS_REMAINING_STAKE_LESS_THAN_MINIMAL, STATUS_PARTICIPANT_ALREADY_HAS_LOCK, STATUS_TRANSFER_AMOUNT_IS_TOO_BIG, STATUS_TRANSFER_SELF, STATUS_TRANSFER_TO_OR_FROM_VALIDATOR, STATUS_FEE_TOO_SMALL, STATUS_INVALID_ADDRESS, STATUS_INVALID_DONOR, STATUS_NO_ELECTION_ROUND, STATUS_INVALID_ELECTION_ID, STATUS_TRANSFER_WHILE_COMPLETING_STEP, STATUS_NO_POOLING_STAKE, STATUS_NOT_ALLOWED_PARTICIPANT

Recommendation: Remove the redundant constant variables from the code. Before removal, ensure that there are no future plans to use those constants.

Found in: b293dd8

Status: Fixed (Revised commit: bb9497b)

I03. Redundant Imports

The contract StEverStrategiesManager includes an import for ClusterLib.tsol which appears to be unnecessary as none of its members or functionalities are referenced within the contract.

The interface IStEverAccount includes an import for IStEverVault.tsol which appears to be unnecessary as none of its members or functionalities are referenced within the contract.

The interface IStEverCluster includes an import for IStEverVault.tsol which appears to be unnecessary as none of its members or functionalities are referenced within the contract.

Redundant imports can lead to confusion and unnecessary bloat in the codebase.

Paths:
./contracts/base/StEverStrategiesManager.tsol : ClusterLib.tsol
./contracts/interfaces/IStEverAccount.tsol : IStEverVault.tsol
./contracts/interfaces/IStEverCluster.tsol : IStEverVault.tsol

Recommendation: Remove redundant import statements from the contracts to maintain clarity and reduce potential confusion. Before removal, ensure that there are no future plans to use those imports.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

I04. Redundant Event Declaration

The event Receive is declared in the code but appears to be redundant as it is not being used elsewhere in the contract.

The event RemoveStrategyError is declared in the code but appears to be redundant as it's not being used elsewhere in the contract.

Unnecessary event declarations can lead to confusion and clutter in the codebase.

Paths:
./contracts/interfaces/IStEverAccount.tsol : Receive
./contracts/interfaces/IStEverCluster.tsol : RemoveStrategyError

Recommendation: Remove the redundant event declarations from the contracts to maintain clarity and reduce potential confusion. Before removal, ensure that there are no future plans to use this event.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

I05. Redundant Use Of Modifier

In the function startEmergencyProcess(), the minCallValue() modifier appears to be redundant. The msg.value is evaluated later in the code, making this initial check unnecessary. Additionally, the value of 1 EVER is not appropriate for this function.

In the function upgradeStEverAccounts(), the minCallValue() modifier appears to be redundant. The msg.value is evaluated later in the code with a specific require statement, making this initial check unnecessary.

Paths:
./contracts/base/StEverVaultEmergency.tsol : startEmergencyProcess()
./contracts/base/StEverVaultBase.tsol : upgradeStEverAccounts()

Recommendation: Remove the minCallValue() modifier from the startEmergencyProcess() function declaration to maintain clarity and reduce potential confusion.

Remove the minCallValue() modifier from the upgradeStEverAccounts() function declaration to maintain clarity and reduce potential confusion.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

I06. Magic Numbers Usage

Constant number values are used as message flags that obscure their purpose.

Path:
./contracts/StEverCluster.tsol : deployStrategies(), addStrategies()

Recommendation: Use *MsgFlag* library to make flags declarative. For example, use MsgFlag.SENDER_PAYS_FEES for flag: 1.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

I07. Testing Functions in Production Interface

The interface IDePool contains definitions for testing functions roundComplete(), setClosed(), and setWithdrawalsClosed().

These testing functions should not be part of a production interface, as they can introduce risks and potential misuse in a live environment.

Path:
./contracts/interfaces/IDePool.tsol : roundComplete(), setClosed(), setWithdrawalsClosed()

Recommendation: Consider declaring these testing functions in a separate testing-only interface or provide proper documentation and guards to ensure they are not misused in a production environment.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

I08. Style Guide Violation

There are variable names defined in the snake case.

Paths:
./contracts/StEverAccount.tsol : _upgrade_data, send_gas_to
./contracts/StEverVault.tsol : _withdraw_nonce
./contracts/base/StEverVaultBase.tsol : constructor_params

Recommendation: Follow the style guides, use camel case for variable and parameter names.

Found in: b293dd8

Status: Fixed (Revised commit: bb9497b)

I09. Typographical Error

The name of the modifier onyClusterOwner() contains a typographical error.

The name of the modifier onyStrategyFactory() contains a typographical error.

The parameter name _poofNonce in the function startEmergencyProcess() contains a typographical error.

Typographical error can lead to confusion and potential misuse.

Paths:
./contracts/StEverCluster.tsol : onyClusterOwner(), onyStrategyFactory()
./contracts/base/StEverVaultEmergency.tsol : _poofNonce

Recommendation: Rename the modifier from onyClusterOwner to onlyClusterOwner to accurately reflect its purpose and avoid potential confusion.

Rename the modifier from onyStrategyFactory to onlyStrategyFactory to accurately reflect its purpose and avoid potential confusion.

Rename the parameter from _poofNonce to _proofNonce to accurately reflect its purpose and avoid potential confusion.

Found in: b293dd8

Status: Fixed (Revised commit: bb9497b)

I10. Code Consistency

In the function setStEverFeePercent(), the transfer method is invoked using msg.sender.transfer(). However, in other onlyOwner functions, the convention used is owner.transfer().

In the function stopEmergencyProcess(), the transfer method is invoked using msg.sender.transfer(). However, in other onlyOwner functions, the convention used is owner.transfer().

This inconsistency can lead to confusion and potential misuse in future code updates.

Paths:
./contracts/base/StEverVaultBase.tsol : setStEverFeePercent()
./contracts/base/StEverVaultEmergency.tsol : stopEmergencyProcess()

Recommendation: For code consistency, replace msg.sender.transfer() with owner.transfer() to align with the convention used in other onlyOwner functions.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

I11. Missing NatSpec

All contracts throughout the codebase lack NatSpec documentation.

Comprehensive documentation is crucial for understanding the purpose, functionality, and usage of contracts and their functions, especially in a decentralized system where transparency is essential.

Paths:
./contracts/*

Recommendation: Add comprehensive NatSpec comments to all contracts that lack them. This will ensure clarity, transparency, and ease of understanding for developers and users interacting with the code.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

I12. TODO Comment

A TODO comment has been found in the codebase, indicating an unresolved task or pending action related to the grammar in a comment. Such comments in production code can lead to confusion and potential misunderstandings about the state of the code or its intended behavior.

Path:
./contracts/base/StEverVaultEmergency.tsol : startEmergencyProcess()

Recommendation: Address the TODO comment by either taking the necessary action or removing the comment if it is no longer relevant. Ensure that all comments in the codebase are clear, grammatically correct, and provide meaningful context.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

I13. Redundant State Change and Event Emission

In the function setIsPaused(), the state variable isPaused is always updated and the event PausedStateChanged is always emitted regardless of whether the passed value _isPaused is different from the current state.

This can lead to unnecessary Gas consumption and redundant event logs in cases where the state remains unchanged.

Path:
./contracts/base/StEverVaultBase.tsol : setIsPaused()

Recommendation: Introduce an if condition to check if isPaused is different from _isPaused. Only change the state and emit the event when they are different. This will optimize Gas usage and avoid redundant event logs.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

I14. Missing Error Code

In the external function createCluster(), the require() statement that checks if msg.value is sufficient lacks an error code.

Consistent usage of error codes is crucial for debugging and understanding the cause of a transaction failure.

Path:
./contracts/base/StEverStrategiesManager.tsol : createCluster()

Recommendation: Add the missing error code ErrorCodesCluster.LOW_MSG_VALUE to the require() statement to provide clarity on the nature of the failure if the condition is not met.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

I15. Magic Numbers Usage

In the function onStrategyRemoved(), a magic number 0.2 ever is used twice for the value parameter in contract method calls.

In the internal function isDePoolMakingWithdraw(), a magic number 0.05 ever is used to compare with _msgValue.

In the function onRoundComplete(), magic numbers 0.11 ever and 0.1 ever are used for the value parameter in contract method calls.

Using magic numbers can lead to confusion and make the code harder to maintain or modify.

Paths:
./contracts/StEverCluster.tsol : onStrategyRemoved()
./contracts/StrategyDePool.tsol : isDePoolMakingWithdraw(), onRoundComplete()

Recommendation: Consider declaring a named constant for the value 0.2 ever to improve code readability and maintainability.

Consider declaring a named constant for the value 0.05 ever to improve code readability and maintainability.

Consider declaring a named constant for the 0.11 ever and 0.1 ever values to improve code readability and maintainability.

Using a descriptive name for the constant will provide context and clarity about its purpose.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

I16. Redundant Contract Declaration

The contract RPlatform appears to be a redundant wrapper around the Platform contract without any additional functionality or modifications, and it is not used by other contracts.

The contract Wallet appears to be a redundant wrapper around the Account contract without any additional functionality or modifications, and is not used by other contracts.

Such redundancy can lead to confusion and unnecessary clutter in the codebase.

Paths:
./contracts/RPlatform.tsol
./contracts/Wallet.tsol

Recommendation: Consider removing the RPlatform contract or provide justification for its existence.

Consider removing the Wallet contract or provide justification for its existence.

Found in: b293dd8

Status: Mitigated (Used to build contract artifacts.)

I17. Redundant Code Block

In the function onStrategyRemoved(), the clusterOwner.transfer({value: 0, flag: MsgFlag.ALL_NOT_RESERVED, bounce: false}); statement is executed both within an if block and outside of it. This redundancy can lead to unnecessary gas consumption and potential confusion about the intended behavior.

Path:
./contracts/StEverCluster.tsol : onStrategyRemoved()

Recommendation: Remove the redundant clusterOwner.transfer({value: 0, flag: MsgFlag.ALL_NOT_RESERVED, bounce: false}); statement from inside the if block, as the same statement is executed immediately after the if block concludes. This will optimize Gas usage and clarify the function's logic.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

I18. Redundant Function Declaration

The internal function _reserveWithValue() is declared in the StrategyDePool contract but appears to be redundant as it is not being used elsewhere in the contract.

The internal function _reserveExceptFee() is declared in the code but appears to be unused elsewhere in the contract. From the constant variables and the purpose of the function, it seems it was initially designed to be used in the deposit() function to extract DEPOSIT_FEE from msg.value.

Unnecessary function declarations can lead to confusion and clutter in the codebase.

Paths:
./contracts/StrategyDePool.tsol : _reserveWithValue()
./contracts/base/StEverVaultBase.tsol : _reserveExceptFee()

Recommendation: Remove the redundant internal function declaration _reserveWithValue() from the StrategyDePool contract to maintain clarity and reduce potential confusion. Before removal, ensure that there are no future plans to use this function.

Remove the redundant internal function declaration _reserveExceptFee() from the StEverVaultBase contract to maintain clarity and reduce potential confusion. Before removal, ensure that there are no future plans to use this function.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

I19. Code Consistency

In the function getDetails() in StEverVaultBase contract, the order of bounce and flag fields in the return object is inconsistent with the order found in most other contracts in the codebase.

In the function installNewStrategyCode() in DepoolStrategyFactory contract, the order of bounce and flag fields in the return object is inconsistent with the order found in most other contracts in the codebase.

Consistent code structure and ordering enhance readability and maintainability.

Path:
./contracts/base/StEverVaultBase.tsol : getDetails()
./contracts/DepoolStrategyFactory.tsol : installNewStrategyCode()

Recommendation: Reorder the fields in the return object of the getDetails() function to match the convention found in other contracts.

Reorder the fields in the _sendGasTo.transfer() function to match the convention found in other contracts.

Specifically, ensure that the bounce field is declared last, as seen in most cases.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

I20. Redundant tvm.accept()

In the function onRoundComplete(), the tvm.accept(); statement is redundant.

Given that this function is always called via a TVM internal message and it always carries some value, this statement is unnecessary.

Furthermore, a subsequent tvm.rawReserve call is made to ensure the proper Gas value reservation, making the prior tvm.accept(); call superfluous.

Redundant code can lead to confusion and unnecessary Gas consumption.

Path:
./contracts/base/StEverVaultBase.tsol : getDetails()

Recommendation: Remove the redundant tvm.accept(); statement from the function. Ensure that the remaining tvm.rawReserve call sufficiently handles the Gas value reservation for the intended behavior of the function.

Found in: b293dd8

Status: Mitigated (with Customer notice:

"*It isn't redundant, because the DePool contract sends only 1 nEver. So we need to make a tvm.accept() otherwise the transaction will be immediately terminated.*")

I21. Duplicate Field in Return Object

In the function getDetails() in StrategyDePool contract, the field bounce: false is duplicated in the return object. Duplicate fields can lead to confusion and potential misunderstandings about the intended structure and behavior of the returned object.

Path:
./contracts/StrategyDePool.tsol : getDetails()

Recommendation: Remove the duplicated bounce: false field from the return object to maintain clarity and ensure the correct behavior of the function.

Found in: b293dd8

Status: Fixed (Revised commit: 0ee727f)

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	**Critical**	**High**	**Medium**
Medium Likelihood	**High**	**Medium**	**Low**
Low Likelihood	**Medium**	**Low**	**Low**

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Hacken OÜ
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia
support@hacken.io

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/broxus/stEver-contracts
Commit	b293dd889b154e77bbf83d48ec5e0db413173ed7
Technical description	README.md
Contracts	File: contracts/DepoolStrategyFactory.tsol SHA3: 594c540f72046e1933f81499246efcde804419c892adf57310b9c29bee122743 File: contracts/Platform.tsol SHA3: d6b53325cf9dfe7145e00c0e860df32ceac31d68d4e51fc624a8845dcfae5f04 File: contracts/StEverAccount.tsol SHA3: db87d1ff02cdcedfa8e5b90d53372a3181db13a64b03c2459f5badd18f716d9c File: contracts/StEverCluster.tsol SHA3: ada22a75c646f59b1dcf33089a51d6b3e5b59d3d2faf96cfe6d35d61c8157637 File: contracts/StEverVault.tsol SHA3: e77361f0cb5c28fd4732c6e24c4c6deae02bae57cef3e4815956a3de20ceedd8 File: contracts/StrategyDePool.tsol SHA3: 2f9ba228b54bf3272bad3c2ca14c12251f1f414a11be36a4014e09b3dd597440 File: contracts/Wallet.tsol SHA3: d8681a227bec361b72e7dcad6dffa8648511c05d03c6d68fb5f5404135a04745 File: contracts/base/StEverStrategiesManager.tsol SHA3: 9efc4a2521bc92fdc5f9444ce4eb64fa344e63e8cdff98c0950157df6ba0829d File: contracts/base/StEverVaultBase.tsol SHA3: 5b1fc774d35fca41d5f0942a21cacfe0893314931cf5f91bc0f09a01343e9dab File: contracts/base/StEverVaultEmergency.tsol SHA3: 5782f69152078a99add2a30730fe2723c5e3ed11fb00c0737431e3d52c49ddd8 File: contracts/base/StEverVaultStorage.tsol SHA3: 3a0f1ace82ed3c7ad45bdbfacae3d6171d2964161d7d8afde53351ec4edf1b74 File: contracts/base/StEverVaultStrategiesController.tsol SHA3: f6c7a97c531d2b5c574e7d9a3eb7087e731045eeb6e79c5ebd2c3fc616d18eac File: contracts/base/StEverVaultValidators.tsol SHA3: f3c70ed88bae906217083122812fdc6727ef1c1b4fdf78535a0eec017c3fad09 File: contracts/utils/ClusterLib.tsol SHA3: d98dd5ae052786baf30ff2be18eddfd471fade143507e61c52bcec4065f730f5 File: contracts/utils/Constants.tsol SHA3: e48fcf15af3a5b245fd568b01a9e40e08785ae913f761b6db6474fd3e50f2ca5 File: contracts/utils/ErrorCodes.tsol SHA3: 850edef415f85112b5a161a80ea2b9c5b379b8e9f4c94afac2491843db9e6818

	File: contracts/utils/Gas.tsol SHA3: 19a74f9afbdffe3da2647266a779ba7501074407c9a0cbab97927eb1efa1f0af

Second review scope

Repository	https://github.com/broxus/stEver-contracts
Commit	0ee727f3528477e9492dc22718f29934de7b5482
Requirements	README.md https://st-ever-docs.netlify.app/contracts
Technical description	README.md NatSpec comments https://st-ever-docs.netlify.app/contracts
Contracts	File: contracts/DepoolStrategyFactory.tsol SHA3: 658d30dc084d31915becc5f2f3a0a19bd1c650f85a03442aebf18248b4e7ca4b File: contracts/Platform.tsol SHA3: de9c43e5c810212d7a6ef2b9f9b683eac1088ea30c5a56b853ca002e3643ec83 File: contracts/StEverAccount.tsol SHA3: 59994c936021c7482d66dcd8b07e5869e4e6d75680129216159ae162b9301f74 File: contracts/StEverCluster.tsol SHA3: 02eccf5212feb08fc0926bb139e5816d98f23ef388e3ff1b5235593af84318a7 File: contracts/StEverVault.tsol SHA3: 3a798a46a61b7f3eead3ccd2f30aadcf5891538d36c25e666229f2370a7945dc File: contracts/StrategyDePool.tsol SHA3: 1703ed0fcaf04aa93c73a286e22d90117a29c960ebdee70d83716f3ec53ca2f0 File: contracts/Wallet.tsol SHA3: 9d904be7fffdb30794661737bd2c73fd018a094e0045dab19601c5627e52a1d2 File: contracts/base/StEverStrategiesManager.tsol SHA3: 612dd78ebe7f4b879a815761603b0505c9d33944a254c0ade5439783c8a6f131 File: contracts/base/StEverVaultBase.tsol SHA3: ae95f9a148c21b59354e3c132b92e05ab62f386ff9610b22bf6591a43a899e5c File: contracts/base/StEverVaultEmergency.tsol SHA3: 654a92e1659cd8c6848c5073aff8329abccf465216e90a8370ebe461e214ffbc File: contracts/base/StEverVaultStorage.tsol SHA3: 576ae57e4f9cb8bd60b67ea5dae66bad4dbd4304275d382d8d131b7da488fa77 File: contracts/base/StEverVaultStrategiesController.tsol SHA3: f1a980f913fb1efa64227b2c2d0f4987efda6d597fccef3aa0c9398dba8e1d12 File: contracts/base/StEverVaultValidators.tsol SHA3: 29fc015496a8bb5ef9cd27d6213037d7ea4f83ba91636ba892d6b4576b4037e8 File: contracts/interfaces/IDePool.tsol SHA3: 1711ae39189dfea517ad6c8377cf8033d06c9378d9579cca0a08b1e9cd7edabd File: contracts/interfaces/IDePoolStrategy.tsol SHA3: 4a274dddf30f3f3c04041928805cd7411071fae51c059b5e77d03ad78f702b53 File: contracts/interfaces/IDepoolStrategyFactory.tsol SHA3: f5d7efbfd94e06aedb3a7e27c877da69b1111434a8e31cde376e3c737bcd9003

	File: contracts/interfaces/IParticipant.tsol SHA3: bc370f6c977d085f5cc24bd6fd29a14aa9558b249a2c1174e058095f40999067 File: contracts/interfaces/IStEverAccount.tsol SHA3: ca1c6ef7a4225a3fe401de6e8be41d66df3c6bb5d2b30a3aa095044f08d80016 File: contracts/interfaces/IStEverCluster.tsol SHA3: 484b9476c9e4dc931c66dc0783cec7d8142e3230d33ceb4100d5d213fe92607a File: contracts/interfaces/IStEverVault.tsol SHA3: dc33e123d5ef37d8ac67d192b9a46de904bc285ca50478aa0c9d50a8db7bdeaa File: contracts/interfaces/IStrategy.tsol SHA3: d09108942b2902bca11e20372a531d13bbf1005776b792ad5fb0b56031c46241 File: contracts/utils/ClusterLib.tsol SHA3: d98dd5ae052786baf30ff2be18eddfd471fade143507e61c52bcec4065f730f5 File: contracts/utils/Constants.tsol SHA3: 56b84b00737c93492021fbb78f96a7a6720bf5f204c19bc2bdd6398295341d54 File: contracts/utils/ErrorCodes.tsol SHA3: a1036f90217ffc059fdce799bd4c4fbc74be778475032f0108333ff3a9428e79 File: contracts/utils/Gas.tsol SHA3: 902c64a0404ac301adc874554be34ec6a20b87573d10ea5b72fa7acf18118ddd File: contracts/utils/Utils.tsol SHA3: 1f574bec9b64f4dd856b425a832591a37ef7b8cec4a729bdad8cd76e8476e7f2

Third review scope

Repository	https://github.com/broxus/stEver-contracts/tree/venom-main
Commit	bb9497b55bddbcec5767e3c2d5caa6f4239f9697
Requirements	README.md https://st-ever-docs.netlify.app/contracts
Technical description	README.md NatSpec comments https://st-ever-docs.netlify.app/contracts
Contracts	File: contracts/DepoolStrategyFactory.tsol SHA3: 648eedad3beed14bfacbd7432870106cec3291d9fccc3bfe999391748410ab8f File: contracts/Platform.tsol SHA3: de9c43e5c810212d7a6ef2b9f9b683eac1088ea30c5a56b853ca002e3643ec83 File: contracts/StEverAccount.tsol SHA3: 73a3cc4cb75277df66c95edc015702b1334e8e162e6ee52f2c9df63bad0bd27c File: contracts/StEverCluster.tsol SHA3: 9bfaf70545ac57154f7a4d9b57fcfb182beb63f2af245019cff475902e64d39d File: contracts/StEverVault.tsol SHA3: cb5989f58cfba29ae7ae11d52a2e294a5b82fa9c0d339aa569c38032e537c9dc File: contracts/StrategyDePool.tsol SHA3: 6ca840a4b6852a66b510510be2deac3be3830310a00088272f35d8d6acf91401 File: contracts/Wallet.tsol

SHA3: 9d904be7fffdb30794661737bd2c73fd018a094e0045dab19601c5627e52a1d2

File: contracts/base/StEverStrategiesManager.tsol
SHA3: 0f6f47654c5c0f2776c6ecd532062d8430603c995d36202ebd2b02c40b8caa7f

File: contracts/base/StEverVaultBase.tsol
SHA3: 74703b2a4b17e3b718c634aad0431b4203c4ae95be6dbfbb2dd229acdd812b9c

File: contracts/base/StEverVaultEmergency.tsol
SHA3: 3af5e2d9e942a774268ec1d5c0064c1121b9166fc520b4e8993812be4e090ad8

File: contracts/base/StEverVaultStorage.tsol
SHA3: 576ae57e4f9cb8bd60b67ea5dae66bad4dbd4304275d382d8d131b7da488fa77

File: contracts/base/StEverVaultStrategiesController.tsol
SHA3: 3bebe89ac6868fe9332875e3fff753d3df3ecd3830155bd4554aba9d40533421

File: contracts/base/StEverVaultValidators.tsol
SHA3: 29fc015496a8bb5ef9cd27d6213037d7ea4f83ba91636ba892d6b4576b4037e8

File: contracts/interfaces/IDePool.tsol
SHA3: 1711ae39189dfea517ad6c8377cf8033d06c9378d9579cca0a08b1e9cd7edabd

File: contracts/interfaces/IDePoolStrategy.tsol
SHA3: 4a274dddf30f3f3c04041928805cd7411071fae51c059b5e77d03ad78f702b53

File: contracts/interfaces/IDepoolStrategyFactory.tsol
SHA3: f5d7efbfd94e06aedb3a7e27c877da69b1111434a8e31cde376e3c737bcd9003

File: contracts/interfaces/IParticipant.tsol
SHA3: bc370f6c977d085f5cc24bd6fd29a14aa9558b249a2c1174e058095f40999067

File: contracts/interfaces/IStEverAccount.tsol
SHA3: 71bae72b7f4b36aa1bbb54a8c8c88535d91bd19af71ac17c0ed8d9717336e4d6

File: contracts/interfaces/IStEverCluster.tsol
SHA3: c12819975b4b4c985af956ca5038bb9066ade5d00b2dc6bac4a29870d91a293f

File: contracts/interfaces/IStEverVault.tsol
SHA3: 75a9b0b5a83e6aedd27c11e2441f253a0f08998c77126297a002ae8bdebd77ab

File: contracts/interfaces/IStrategy.tsol
SHA3: d09108942b2902bca11e20372a531d13bbf1005776b792ad5fb0b56031c46241

File: contracts/utils/ClusterLib.tsol
SHA3: d98dd5ae052786baf30ff2be18eddfd471fade143507e61c52bcec4065f730f5

File: contracts/utils/Constants.tsol
SHA3: 26a96a3b97b31c40a441c8030469d6be0a654016a90665c9b68d174fbdf02fd0

File: contracts/utils/ErrorCodes.tsol
SHA3: 8e3fba97751e800852a08e36851ec50a31424fe523ff6cf294aa892ce459b56b

File: contracts/utils/Gas.tsol
SHA3: 4e46bb6823a5bfcd5cef726c2831ae8043d34dd9342c41e816d20cf2f818112e

File: contracts/utils/Utils.tsol
SHA3: 1f574bec9b64f4dd856b425a832591a37ef7b8cec4a729bdad8cd76e8476e7f2