OMNISCIA

July 4, 2025

SMART CONTRACT AUDIT REPORT

Size Credit

Rebasing Token Vault


omniscia.io


info@omniscia.io


Online report: size-credit-rebasing-token-vault

Omniscia.io is one of the fastest growing and most trusted blockchain security firms and has rapidly become a true market leader. To date, our team has collectively secured over 370+ clients, detecting 1,500+ high-severity issues in widely adopted smart contracts.

Founded in France at the start of 2020, and with a track record spanning back to 2017, our team has been at the forefront of auditing smart contracts, providing expert analysis and identifying potential vulnerabilities to ensure the highest level of security of popular smart contracts, as well as complex and sophisticated decentralized protocols.

Our clients, ecosystem partners, and backers include leading ecosystem players such as L'Oréal, Polygon, AvaLabs, Gnosis, Morpho, Vesta, Gravita, Olympus DAO, Fetch.ai, and LimitBreak, among others.

To keep up to date with all the latest news and announcements follow us on twitter @omniscia_sec.


omniscia.io


info@omniscia.io

Online report: size-credit-rebasing-token-vault

Rebasing Token Vault Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
6911b4564c	June 30th 2025	98c2e2d721
9dcf85ecb3	July 2nd 2025	7f5a3c9210
9dcf85ecb3	July 4th 2025	03c667a05c

Audit Overview

We were tasked with performing an audit of the Size Credit codebase and in particular their Rebasing Token Vault module.

The Rebasing Token Vault is meant to closely integrate with the rest of the Size Credit codebase that is out-of-scope for the purposes of this audit engagement. As such, the validation performed was contained to the contracts within the audit scope and how they interact between them.

The system implements a rebasing token mechanism that can be attached to several **EIP-4626** vaults as well as an Aave token implementation as long as all contracts have the same underlying token.

Over the course of the audit, we identified a flaw in the arithmetic calculations carried out in the Aave adapter implementation, potential issues in how liquidity is validated post-interaction for each vault, as well as issues related to adapter management.

We advise the Size Credit team to closely evaluate all minor-and-above findings identified in the report and promptly remediate them as well as consider all optimizational exhibits identified in the report.

Post-Audit Conclusion

The Size Credit team iterated through all findings within the report and provided us with a revised commit hash to evaluate all exhibits on.

We evaluated all alleviations performed by Size Credit and confirmed that all exhibits have either been alleviated, safely acknowledged, or nullified.

To note, the audit scope was not updated to include the market-related interactions and those interactions should be validated at a later point.

We consider all outputs of the audit report properly consumed by the Size Credit team with no outstanding remediative actions remaining.

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	1	0	0	1
Informational	12	3	0	9
Minor	5	5	0	0
Medium	3	2	0	1
Major	0	0	0	0

During the audit, we filtered and validated a total of **0 findings utilizing static analysis** tools as well as identified a total of **21 findings during the manual review** of the codebase. We strongly recommend that any minor severity or higher findings are dealt with promptly prior to the project's launch as they can introduce potential misbehaviours of the system as well as exploits.

- **Scope**
- **Compilation**
- **Static Analysis**
- **Manual Review**
- **Code Style**

Scope

The audit engagement encompassed a specific list of contracts that were present in the commit hash of the repository that was in scope. The tables below detail certain meta-data about the target of the security assessment and a navigation chart is present at the end that links to the relevant findings per file.

Target

- Repository: **https://github.com/SizeCredit/size-solidity**
- Commit: 6911b4564c01a77cb30657b6593af3fcf3640778
- Language: Solidity
- Network: Ethereum, Base
- Revisions: **6911b4564c**, **9dcf85ecb3**

Contracts Assessed

File	Total Finding(s)
src/market/token/adapters/AaveAdapter.sol (AAR)	5
src/market/token/adapters/ERC4626Adapter.sol (ERC)	4
src/market/libraries/Math.sol (MHT)	1
src/market/token/NonTransferrableRebasingTokenVault.sol (NTR)	11
src/helpers/ReentrancyGuardUpgradeableWithViewModifier.sol (RGU)	0

Compilation

The project utilizes `foundry` as its development pipeline tool, containing an array of tests and scripts coded in Solidity.

To compile the project, the `build` command needs to be issued via the `forge` CLI tool:

```bash
forge build
```

The `forge` tool automatically selects Solidity version `0.8.23` based on the version specified within the `foundry.toml` file.

The project contains discrepancies with regards to the Solidity version used, however, they are solely contained in external dependencies and can thus be safely ignored.

The `pragma` statements have been locked to `0.8.23` (`=0.8.23`), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `foundry` pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

Static Analysis

The execution of our static analysis toolkit identified **13 potential issues** within the codebase of which **13 were ruled out to be false positives** or negligible findings.

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Size Credit's non-transferrable rebasing token vault.

As the project at hand integrates with multiple share-based systems, intricate care was put into ensuring that the **flow of funds within the system conforms to the specifications and restrictions** laid forth within the protocol's specification.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **pinpointed multiple significant vulnerabilities** within the system which could have had **moderate ramifications** to its overall operation; for more information, we advise the relevant medium severity exhibits within the audit report to be evaluated.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to a great extent, however, we advise the assumptions of the market-based integration points to be laid out via inline documentation so as to aid future validation efforts.

A total of **21 findings** were identified over the course of the manual review of which **10 findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

The finding table below enumerates all these security / behavioural findings:

ID	Severity	Addressed	Title
AAR-01M	Minor	Yes	Inexistent Correlation of Underlying Token Between C
AAR-02M	Minor	Nullified	Potentially Insufficient Liquidity Evaluation
AAR-03M	Medium	Acknowledged	Incorrect Mathematical Implementations of Aave Forn
ERC-01M	Minor	Yes	Inexistent Correlation of Underlying Token Between C
ERC-02M	Minor	Nullified	Potentially Insufficient Liquidity Evaluation

NTR-01M	Unknown	Acknowledged	Unknown Market Integration Points
NTR-02M	Informational	Acknowledged	Inefficient Assignments of Boolean
NTR-03M	Minor	Nullified	Potentially Insufficient Liquidity Evaluation
NTR-04M	Medium	Yes	Breach of One-to-One Adapter-ID Relation
NTR-05M	Medium	Yes	Insecure Removal of Adapter

Code Style

During the manual portion of the audit, we identified **11 optimizations** that can be applied to the codebase that will decrease the operational cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.

Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.

These optimizations are enumerated below:

ID	Severity	Addressed	Title
AAR-01C	Informational	Acknowledged	Ineffectual Usage of Safe Arithmetics
AAR-02C	Informational	Acknowledged	Redundant Logic Implementation Import
ERC-01C	Informational	Acknowledged	Ineffectual Usage of Safe Arithmetics
ERC-02C	Informational	Acknowledged	Redundant Logic Implementation Import
MHT-01C	Informational	Acknowledged	Redundant Return Data
NTR-01C	Informational	Acknowledged	Generic Typographic Mistakes
NTR-02C	Informational	Yes	Inefficient Loop Limit Evaluation
NTR-03C	Informational	Yes	Logical Branch Optimization
NTR-04C	Informational	Acknowledged	Potential Mutability Optimization
NTR-05C	Informational	Acknowledged	Redundant Boolean Variables

NTR-06C	Informational	Yes	Repetitive Value Literal

AaveAdapter Manual Review Findings

AAR-01M: Inexistent Correlation of Underlying Token Between Contracts

Type	Severity	Location
Logical Fault	Minor	AaveAdapter.sol:L45, L47

Description:

The `underlyingToken` of the `AaveAdapter` is configured separately from the `underlyingToken` of the `NonTransferrableRebasingTokenVault`; a practice we consider non-standard.

Impact:

It is presently possible to attach a particular `AaveAdapter` to any `NonTransferrableRebasingTokenVault` regardless of the underlying token of each contract.

Example:

src/market/token/adapters/AaveAdapter.sol

SOL

```sol
36  constructor(NonTransferrableRebasingTokenVault _tokenVault, IPool _aavePool,
IERC20Metadata _underlyingToken)
37      Ownable(address(_tokenVault))
38  {
39      if (
40          address(_tokenVault) == address(0) || address(_aavePool) == address(0)
41              || address(_underlyingToken) == address(0)
42      ) {
43          revert Errors.NULL_ADDRESS();
44      }
45      tokenVault = _tokenVault;
```

Example (Cont.):

```sol
46      aavePool = _aavePool;
47      underlyingToken = _underlyingToken;
48      aToken =
IAToken(_aavePool.getReserveData(address(_underlyingToken)).aTokenAddress);
49  }
```

Recommendation:

As the two contracts need to support the same underlying token, we advise the `AaveAdapter` contract to query the underlying token of the `NonTransferrableRebasingTokenVault` and utilize that for its own `underlyingToken` configuration, ensuring consistency between the two contract instances.

Alleviation:

The **`AaveAdapter::constructor`** was updated to properly query the configuration of its attached `NonTransferrableRebasingTokenVault` instance, addressing this exhibit.

AAR-02M: Potentially Insufficient Liquidity Evaluation

Type	Severity	Location
Logical Fault	Minor	AaveAdapter.sol:L145-L149

Description:

The `AaveAdapter::checkLiquidity` implementation is theoretically meant to ensure that a full withdrawal of the recipient account is possible after a transfer operation succeeds, however, this trait is incorrect evaluated as the `AaveAdapter::checkLiquidity` function is utilized with the transferred amount rather than the full amount of the `to` recipient.

Impact:

The liquidity evaluation mechanism of the `AaveAdapter` appears to be insecure as it will validate transfers to accounts who have amalgamated enough funds that are no longer redeemable in a single interaction.

Example:

src/market/token/adapters/AaveAdapter.sol

SOL

```sol
122 /// @inheritdoc IAdapter
123 function transferFrom(address vault, address from, address to, uint256 value)
external onlyOwner {
124     checkLiquidity(vault, value);
125
126     uint256 shares = Math.mulDivDown(value, WadRayMath.RAY, liquidityIndex());
127     if (tokenVault.sharesOf(from) < shares) {
128         revert IERC20Errors.ERC20InsufficientBalance(from, balanceOf(vault,
from), value);
129     }
130     tokenVault.setSharesOf(from, tokenVault.sharesOf(from) - shares);
131     tokenVault.setSharesOf(to, tokenVault.sharesOf(to) + shares);
```

Example (Cont.):

```sol
}

/// @inheritdoc IAdapter
function validate(address vault) external pure {
    if (vault != DEFAULT_VAULT) {
        revert Errors.INVALID_VAULT(vault);
    }
}

/// @inheritdoc IAdapter
function checkLiquidity(address, /*vault*/ uint256 amount) public view {
    if (underlyingToken.balanceOf(address(aToken)) < amount) {
        revert InsufficientAssets(DEFAULT_VAULT,
underlyingToken.balanceOf(address(aToken)), amount);
    }
}
```

Recommendation:

We advise the full amount to be evaluated, ensuring that there is adequate `AToken` balance left in the `NonTransferrableRebasingTokenVault` to satisfy a potential withdrawal.

Alleviation:

The Size Credit team evaluated this exhibit and clarified that the intended requirement it is meant to impose is the ability to withdraw the transferred amount at a given point in time rather than the recipient's full balance.

As such, we consider the original implementation in line with its specification and thus this exhibit nullified.

AAR-03M: Incorrect Mathematical Implementations of Aave Formulae

Type	Severity	Location
Logical Fault	Medium	AaveAdapter.sol:L126, L154

Description:

The referenced operations are meant to mimic how the Aave protocol functions, however, they employ multiplication and division operations that round downward.

These operations contradict the **Aave arithmetic behaviour** which rounds toward the nearest ray, resulting in potential fund loss that can accumulate to non-negligible amounts.

Impact:

The arithmetic operations employed by the `AaveAdapter` contradict the ones that the Aave protocol utilizes as they round downward regardless of the interim multiplication value.

Example:

src/market/token/adapters/AaveAdapter.sol

SOL

```
122 /// @inheritdoc IAdapter
123 function transferFrom(address vault, address from, address to, uint256 value)
external onlyOwner {
124     checkLiquidity(vault, value);
125
126     uint256 shares = Math.mulDivDown(value, WadRayMath.RAY, liquidityIndex());
127     if (tokenVault.sharesOf(from) < shares) {
128         revert IERC20Errors.ERC20InsufficientBalance(from, balanceOf(vault,
from), value);
129     }
130     tokenVault.setSharesOf(from, tokenVault.sharesOf(from) - shares);
131     tokenVault.setSharesOf(to, tokenVault.sharesOf(to) + shares);
```

Example (Cont.):

```sol
}

/// @inheritdoc IAdapter
function validate(address vault) external pure {
    if (vault != DEFAULT_VAULT) {
        revert Errors.INVALID_VAULT(vault);
    }
}

/// @inheritdoc IAdapter
function checkLiquidity(address, /*vault*/ uint256 amount) public view {
    if (underlyingToken.balanceOf(address(aToken)) < amount) {
        revert InsufficientAssets(DEFAULT_VAULT,
underlyingToken.balanceOf(address(aToken)), amount);
    }
}

/// @inheritdoc IAaveAdapter
function liquidityIndex() public view returns (uint256) {
    return aavePool.getReserveNormalizedIncome(address(underlyingToken));
}

function _unscale(uint256 scaledAmount) private view returns (uint256) {
    return Math.mulDivDown(scaledAmount, liquidityIndex(), WadRayMath.RAY);
}
```

Recommendation:

We advise the same arithmetic operations to be employed by the `AaveAdapter`, ensuring consistency across the Aave protocol and the adapter meant to be curated for it.

Alleviation:

The Size Credit team assessed this exhibit and stated that Denial-of-Service issues have arisen when they attempted to rectify it in previous iterations.

As the current rounding behaviour is at the expense of the user to a negligible amount, we consider this exhibit safely acknowledged due to the effort involved in properly addressing it.

ERC4626Adapter Manual Review Findings

ERC-01M: Inexistent Correlation of Underlying Token Between Contracts

Type	Severity	Location
Logical Fault	Minor	ERC4626Adapter.sol:L39, L40

Description:

The `underlyingToken` of the `ERC4626Adapter` is configured separately from the `underlyingToken` of the `NonTransferrableRebasingTokenVault`; a practice we consider non-standard.

Impact:

It is presently possible to attach a particular `ERC4626Adapter` to any `NonTransferrableRebasingTokenVault` regardless of the underlying token of each contract.

Example:

src/market/token/adapters/ERC4626Adapter.sol

SOL

```sol
33  constructor(NonTransferrableRebasingTokenVault _tokenVault, IERC20Metadata
_underlyingToken)
34      Ownable(address(_tokenVault))
35  {
36      if (address(_tokenVault) == address(0) || address(_underlyingToken) ==
address(0)) {
37          revert Errors.NULL_ADDRESS();
38      }
39      tokenVault = _tokenVault;
40      underlyingToken = _underlyingToken;
41  }
```

Recommendation:

As the two contracts need to support the same underlying token, we advise the `ERC4626Adapter` contract to query the underlying token of the `NonTransferrableRebasingTokenVault` and utilize that for its own `underlyingToken` configuration, ensuring consistency between the two contract instances.

Alleviation:

The **`ERC4626Adapter::constructor`** was updated to properly query the configuration of its attached `NonTransferrableRebasingTokenVault` instance, addressing this exhibit.

ERC-02M: Potentially Insufficient Liquidity Evaluation

Type	Severity	Location
Logical Fault	Minor	ERC4626Adapter.sol:L145-L149

Description:

The `ERC4626Adapter::checkLiquidity` implementation is theoretically meant to ensure that a full withdrawal of the recipient account is possible after a transfer operation succeeds, however, this trait is incorrect evaluated as the `ERC4626Adapter::checkLiquidity` function is utilized with the transferred amount rather than the full amount of the `to` recipient.

Impact:

The liquidity evaluation mechanism of the `ERC4626Adapter` appears to be insecure as it will validate transfers to accounts who have amalgamated enough funds that are no longer redeemable in a single interaction.

Example:

src/market/token/adapters/ERC4626Adapter.sol

SOL

```
123 /// @inheritdoc IAdapter
124 function transferFrom(address vault, address from, address to, uint256 value)
external onlyOwner {
125     checkLiquidity(vault, value);
126
127     uint256 shares = IERC4626(vault).convertToShares(value);
128
129     if (tokenVault.sharesOf(from) < shares) {
130         revert IERC20Errors.ERC20InsufficientBalance(from, balanceOf(vault,
from), value);
131     }
132
```

Example (Cont.):

```sol
    tokenVault.setSharesOf(from, tokenVault.sharesOf(from) - shares);
    tokenVault.setSharesOf(to, tokenVault.sharesOf(to) + shares);
}

/// @inheritdoc IAdapter
function validate(address vault) external view {
    if (IERC4626(vault).asset() != address(underlyingToken)) {
        revert Errors.INVALID_VAULT(vault);
    }
}

/// @inheritdoc IAdapter
function checkLiquidity(address vault, uint256 amount) public view {
    if (IERC4626(vault).maxWithdraw(address(tokenVault)) < amount) {
        revert InsufficientAssets(address(vault),
IERC4626(vault).maxWithdraw(address(tokenVault)), amount);
    }
}
```

Recommendation:

We advise the full amount to be evaluated, ensuring that **EIP-4626** vaults with per-withdrawal limitations can still function as expected in the `NonTransferrableRebasingTokenVault` system.

Alleviation:

The Size Credit team evaluated this exhibit and clarified that the intended requirement it is meant to impose is the ability to withdraw the transferred amount at a given point in time rather than the recipient's full balance.

As such, we consider the original implementation in line with its specification and thus this exhibit nullified.

NonTransferrableRebasingTokenVault Manual Review Findings

NTR-01M: Unknown Market Integration Points

Type	Severity	Location
Standard Conformity	Unknown	**NonTransferrableRebasingTokenVault.sol**: • I-1: **L214-L246** • I-2: **L308-L333** • I-3: **L337-L347** • I-4: **L351-L369** • I-5: **L372-L384**

Description:

The referenced integration points appear to be interacted with by Size markets that are out-of-scope of this security engagement.

Impact:

As the market implementations authorized to interact with the `NonTransferrableRebasingTokenVault` implementation are unknown, it is not possible to validate that the input values of the referenced functions are properly authorized and relayed (for example, `NonTransferrableRebasingTokenVault::transferFrom` does not validate approvals and this should be delegated to the top-level contract that the `tx.origin` interacts with).

Example:

src/market/token/NonTransferrableRebasingTokenVault.sol

SOL

```sol
214 function transferFrom(address from, address to, uint256 value)
215     public
216     virtual
217     onlyMarket
218     nonReentrant
219     returns (bool)
220 {
221     if (from == address(0)) {
222         revert ERC20InvalidSender(address(0));
223     }
```

Example (Cont.):

SOL

```
224     if (to == address(0)) {
225         revert ERC20InvalidReceiver(address(0));
226     }
227
228     if (value > 0) {
229         if (vaultOf[from] == vaultOf[to]) {
230             IAdapter adapter = getWhitelistedVaultAdapter(vaultOf[from]);
231             adapter.transferFrom(vaultOf[from], from, to, value);
232         } else {
233             IAdapter adapterFrom = getWhitelistedVaultAdapter(vaultOf[from]);
234             IAdapter adapterTo = getWhitelistedVaultAdapter(vaultOf[to]);
235             // slither-disable-next-line unused-return
236             adapterFrom.withdraw(vaultOf[from], from, address(adapterTo), value);
237             // slither-disable-next-line unused-return
238             adapterTo.deposit(vaultOf[to], to, value);
239             adapterTo.checkLiquidity(vaultOf[to], value);
240         }
241     }
242
243     emit Transfer(from, to, value);
244
245     return true;
246 }
```

Recommendation:

We advise the Size Credit team to introduce the relevant contracts to the security audit scope for the `NonTransferrableRebasingTokenVault` contract to be audited to the fullest extent possible.

Additionally, we advise the Size Credit team to ensure that the integrations have been securely defined (i.e. input is validated, top-level caller approval is enforced, etc.) to prevent cross-contract vulnerabilities from arising.

Alleviation:

The Size Credit team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

NTR-02M: Inefficient Assignments of Boolean

Type	Severity	Location
Logical Fault	Informational	NonTransferrableRebasingTokenVault.sol:L479, L480

Description:

The referenced `bool` is overwritten by a different value right after its assignment, causing its initial value to remain unused.

Impact:

Although the current logic will cause one data point to remain unused, the two mapping entries are assigned to simultaneously thereby causing their boolean returns to have the same value.

Example:

src/market/token/NonTransferrableRebasingTokenVault.sol

SOL

```sol
477 function _removeAdapter(bytes32 id) private {
478     address adapter = IdToAdapterMap.get(id);
479     bool removed = IdToAdapterMap.remove(id);
480     removed = adapterToIdMap.remove(adapter);
481
482     if (removed) {
483         emit AdapterRemoved(id, adapter);
484     }
485 }
```

Recommendation:

We advise the two values to either be combined via an AND (`&&`) or OR (`||`) conditional depending on the intended purpose, ensuring that removal operations are assessed properly.

Alleviation:

The Size Credit team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

NTR-03M: Potentially Insufficient Liquidity Evaluation

Type	Severity	Location
Logical Fault	Minor	NonTransferrableRebasingTokenVault.sol:L239

Description:

The `NonTransferrableRebasingTokenVault::transferFrom` function will invoke the liquidity validation hook of an adapter by utilizing the transferred `value` rather than the full amount that the recipient vault retains, rendering the liquidity check ineffective.

Impact:

The current liquidity check mechanism employed after a transfer operation succeeds appear to be insufficient as it validates the transferred amount rather than the full balance of the recipient.

Example:

src/market/token/NonTransferrableRebasingTokenVault.sol

SOL

```sol
214 function transferFrom(address from, address to, uint256 value)
215     public
216     virtual
217     onlyMarket
218     nonReentrant
219     returns (bool)
220 {
221     if (from == address(0)) {
222         revert ERC20InvalidSender(address(0));
223     }
```

Example (Cont.):

SOL

```
224     if (to == address(0)) {
225         revert ERC20InvalidReceiver(address(0));
226     }
227
228     if (value > 0) {
229         if (vaultOf[from] == vaultOf[to]) {
230             IAdapter adapter = getWhitelistedVaultAdapter(vaultOf[from]);
231             adapter.transferFrom(vaultOf[from], from, to, value);
232         } else {
233             IAdapter adapterFrom = getWhitelistedVaultAdapter(vaultOf[from]);
234             IAdapter adapterTo = getWhitelistedVaultAdapter(vaultOf[to]);
235             // slither-disable-next-line unused-return
236             adapterFrom.withdraw(vaultOf[from], from, address(adapterTo), value);
237             // slither-disable-next-line unused-return
238             adapterTo.deposit(vaultOf[to], to, value);
239             adapterTo.checkLiquidity(vaultOf[to], value);
240         }
241     }
242
243     emit Transfer(from, to, value);
244
245     return true;
246 }
```

Recommendation:

We advise the full balance that the recipient vault possesses to be passed into the `IAdapter::checkLiquidity` call, ensuring liquidity checks are properly enforced.

Alleviation:

The Size Credit team evaluated this exhibit and clarified that the intended requirement it is meant to impose is the ability to withdraw the transferred amount at a given point in time rather than the recipient's full balance.

As such, we consider the original implementation in line with its specification and thus this exhibit nullified.

NTR-04M: Breach of One-to-One Adapter-ID Relation

Type	Severity	Location
Logical Fault	Medium	NonTransferrableRebasingTokenVault.sol:L465-L473, L47

Description:

The `NonTransferrableRebasingTokenVault::_setAdapter` function permits the same ID to be reconfigured to a new adapter without cleaning up the previous adapter entry from the `adapterToIdMap`.

As a result, any replaced adapter will be permanently left within the `adapterToIdMap` and thus will be able to access functions guarded by the `NonTransferrableRebasingTokenVault::onlyAdapter` modifier incorrectly.

Impact:

Any adapter that has been replaced via the `NonTransferrableRebasingTokenVault::_setAdapter` function will continue to have access to sensitive functions of the `NonTransferrableRebasingTokenVault` and will be impossible to remove due to the 1-to-1 association between IDs and their adapters being breached.

Example:

src/market/token/NonTransferrableRebasingTokenVault.sol

SOL

```
463 /// @notice Sets the adapter
464 // slither-disable-start unused-return
465 function _setAdapter(bytes32 id, IAdapter adapter) private {
466     if (address(adapter) == address(0)) {
467         revert Errors.NULL_ADDRESS();
468     }
469
470     IdToAdapterMap.set(id, address(adapter));
471     adapterToIdMap.set(address(adapter), id);
472     emit AdapterSet(id, address(adapter));
```

Example (Cont.):

SOL

```
473 }
474 // slither-disable-end unused-return
475
476 /// @notice Removes the adapter
477 function _removeAdapter(bytes32 id) private {
478     address adapter = IdToAdapterMap.get(id);
479     bool removed = IdToAdapterMap.remove(id);
480     removed = adapterToIdMap.remove(adapter);
481
482     if (removed) {
483         emit AdapterRemoved(id, adapter);
484     }
485 }
```

Recommendation:

We advise an overwrite of an adapter that already exists for a particular ID to be accompanied by the removal of the adapter from the `adapterToIdMap`, ensuring that a 1-to-1 relation between adapters and their IDs is upheld at all times.

Alleviation:

The function was updated to properly remove a previously attached adapter of a particular ID whenever a replacement occurs, alleviating this exhibit.

NTR-05M: Insecure Removal of Adapter

Type	Severity	Location
Logical Fault	Medium	NonTransferrableRebasingTokenVault.sol:L477-L485

Description:

The removal of an adapter will not properly handle any vaults that point to it, resulting in them becoming inoperable as any transfer operation involving them would fail.

Additionally, certain functions such as `NonTransferrableRebasingTokenVault::totalSupply` that are sensitive will revert due to the outdated `vaultToIdMap` entry resulting in significant misbehaviours for all integrators of the `NonTransferrableRebasingTokenVault` implementation.

Impact:

Any removal of an adapter is presently insecure as it will cause sensitive functions, such as the `NonTransferrableRebasingTokenVault::totalSupply` function, to fail and will cause all funds in vaults that use a particular adapter to be lost until an adapter with the same ID is re-introduced.

Example:

src/market/token/NonTransferrableRebasingTokenVault.sol

SOL

```
476 /// @notice Removes the adapter
477 function _removeAdapter(bytes32 id) private {
478     address adapter = IdToAdapterMap.get(id);
479     bool removed = IdToAdapterMap.remove(id);
480     removed = adapterToIdMap.remove(adapter);
481
482     if (removed) {
483         emit AdapterRemoved(id, adapter);
484     }
485 }
```

Recommendation:

We advise the removal of an adapter to be prohibited, and an adapter replacement to be permitted instead.

Alternatively, we advise the removal of an adapter to be accompanied by a migration or cleanup of outdated vault associations to ensure that the system continues to behave as expected.

Alleviation:

The removal function has been omitted from the codebase altogether, indirectly alleviating this exhibit as the vulnerability described is no longer present.

AaveAdapter Code Style Findings

AAR-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	**AaveAdapter.sol**: • I-1: **L106** • I-2: **L130**

Description:

The linked mathematical operations are guaranteed to be performed safely by surrounding conditionals evaluated in either `require` checks or `if-else` constructs.

Example:

src/market/token/adapters/AaveAdapter.sol

SOL

```sol
102 if (userSharesBefore < shares) {
103     revert IERC20Errors.ERC20InsufficientBalance(from, userBalanceBefore,
amount);
104 }
105
106 tokenVault.setSharesOf(from, userSharesBefore - shares);
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statements to be wrapped in `unchecked` code blocks thereby optimizing their execution cost.

Alleviation:

The Size Credit team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

AAR-02C: Redundant Logic Implementation Import

Type	Severity	Location
Gas Optimization	Informational	AaveAdapter.sol:L20-L22

Description:

The `NonTransferrableRebasingTokenVault` contract is imported as a logic contract yet is utilized as an `interface`.

Example:

src/market/token/adapters/AaveAdapter.sol

SOL

```sol
import {
    DEFAULT_VAULT, NonTransferrableRebasingTokenVault
} from "@src/market/token/NonTransferrableRebasingTokenVault.sol";

contract AaveAdapter is Ownable, IAaveAdapter {
    using SafeERC20 for IERC20Metadata;

    // slither-disable-start uninitialized-state
    // slither-disable-start constable-states
    NonTransferrableRebasingTokenVault public immutable tokenVault;
```

Example (Cont.):

```sol
    IPool public immutable aavePool;
    IERC20Metadata public immutable underlyingToken;
    IAToken public immutable aToken;
```

Recommendation:

We advise an `interface` to be declared for the referenced contract and to be imported to the codebase, optimizing its gas cost.

Alleviation:

The Size Credit team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

ERC4626Adapter Code Style Findings

ERC-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	**ERC4626Adapter.sol**: • I-1: **L104** • I-2: **L133**

Description:

The linked mathematical operations are guaranteed to be performed safely by surrounding conditionals evaluated in either `require` checks or `if-else` constructs.

Example:

src/market/token/adapters/ERC4626Adapter.sol

SOL

```sol
99  if (userSharesBefore < shares) {
100     revert IERC20Errors.ERC20InsufficientBalance(from, userBalanceBefore,
amount);
101 }
102
103 underlyingToken.safeTransfer(to, assets);
104 tokenVault.setSharesOf(from, userSharesBefore - shares);
```

Recommendation:

Given that safe arithmetics are toggled on by default in `pragma` versions of `0.8.X`, we advise the linked statements to be wrapped in `unchecked` code blocks thereby optimizing their execution cost.

Alleviation:

The Size Credit team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

ERC-02C: Redundant Logic Implementation Import

Type	Severity	Location
Gas Optimization	Informational	ERC4626Adapter.sol:L14

Description:

The `NonTransferrableRebasingTokenVault` contract is imported as a logic contract yet is utilized as an `interface`.

Example:

src/market/token/adapters/ERC4626Adapter.sol

SOL

```sol
14  import {NonTransferrableRebasingTokenVault} from
"@src/market/token/NonTransferrableRebasingTokenVault.sol";
15  import {IAdapter} from "@src/market/token/adapters/IAdapter.sol";
16
17  contract ERC4626Adapter is Ownable, IAdapter {
18      using SafeERC20 for IERC20Metadata;
19
20      struct Vars {
21          uint256 sharesBefore;
22          uint256 assetsBefore;
23          uint256 userSharesBefore;
```

Example (Cont.):

```sol
    }

    // slither-disable-start uninitialized-state
    // slither-disable-start constable-states
    NonTransferrableRebasingTokenVault public immutable tokenVault;
```

Recommendation:

We advise an `interface` to be declared for the referenced contract and to be imported to the codebase, optimizing its gas cost.

Alleviation:

The Size Credit team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

Math Code Style Findings

MHT-01C: Redundant Return Data

Type	Severity	Location
Gas Optimization	Informational	Math.sol:L63

Description:

The `Math::binarySearch` function implements a binary search algorithm that, when a particular element is within the range of values in the array, will yield two indexes that either equate and point to the element within the array or are one unit apart and point to the two indexes in between the element would normally be found.

Example:

src/market/libraries/Math.sol

SOL

```sol
/// @notice Find the index of `value` in the sorted list `array`
/// @dev If `value` is below the lowest value in `array` or above the highest value in `array`, the function returns (type(uint256).max, type(uint256).max)
///      Formally verified with Halmos (check_Math_binarySearch)
/// @param array The sorted list to search
/// @param value The value to search for
/// @return low The index of the largest element in `array` that is less than or equal to `value`
/// @return high The index of the smallest element in `array` that is greater than or equal to `value`
function binarySearch(uint256[] memory array, uint256 value) internal pure returns (uint256 low, uint256 high) {
    low = 0;
    high = array.length - 1;
```

Example (Cont.):

SOL

```
    if (value < array[low] || value > array[high]) {
        return (type(uint256).max, type(uint256).max);
    }
    while (low <= high) {
        uint256 mid = (low + high) / 2;
        if (array[mid] == value) {
            return (mid, mid);
        } else if (array[mid] < value) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return (high, low);
}
```

Recommendation:

We advise the code to yield one data point instead, the left index bound of the value, permitting the second data point to be extrapolated (i.e. if element does not exist at yielded `value`, then it exists in `[value, value + 1]`).

Alleviation:

The Size Credit team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

NonTransferrableRebasingTokenVault Code Style Findings

NTR-01C: Generic Typographic Mistakes

Type	Severity	Location
Code Style	Informational	**NonTransferrableRebasingTokenVault.sol**: • I-1: **L73** • I-2: **L74** • I-3: **L75**

Description:

The referenced lines contain typographical mistakes (i.e. `private` variable without an underscore prefix) or generic documentational errors (i.e. copy-paste) that should be corrected.

Example:

src/market/token/NonTransferrableRebasingTokenVault.sol

SOL

```
73  EnumerableMap.AddressToBytes32Map internal vaultToIdMap;
```

Recommendation:

We advise them to be corrected enhancing the legibility of the codebase.

Alleviation:

The Size Credit team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

NTR-02C: Inefficient Loop Limit Evaluation

Type	Severity	Location
Gas Optimization	Informational	NonTransferrableRebasingTokenVault.sol:L445, L446

Description:

The linked `for` loop evaluates its limit inefficiently on each iteration.

Example:

src/market/token/NonTransferrableRebasingTokenVault.sol

SOL

```
445 vaults = new address[](vaultToIdMap.length());
```

Recommendation:

We advise the statements within the `for` loop limit to be relocated outside to a local variable declaration that is consequently utilized for the evaluation to significantly reduce the codebase's gas cost. We should note the same optimization is applicable for storage reads present in such limits as they are newly read on each iteration (i.e. `length` members of arrays in storage).

Alleviation:

All referenced loop limit evaluations were optimized as advised.

NTR-03C: Logical Branch Optimization

Type	Severity	Location
Gas Optimization	Informational	NonTransferrableRebasingTokenVault.sol:L311-L313,

Description:

The referenced distinct logical branches execute the same statements.

Example:

src/market/token/NonTransferrableRebasingTokenVault.sol

SOL

```
311 } else if (!vaultToIdMap.contains(vault)) {
312     revert Errors.INVALID_VAULT(vault);
313 } else if (vaultOf[user] != vault) {
314     if (forfeitOldShares) {
315         _setSharesOf(user, 0);
316     } else if (sharesOf[user] > 0) {
317         IAdapter adapterOld = getWhitelistedVaultAdapter(vaultOf[user]);
318         IAdapter adapterNew = getWhitelistedVaultAdapter(vault);
319         uint256 assetsWithdrawn = adapterOld.fullWithdraw(vaultOf[user], user,
address(adapterNew));
320         if (assetsWithdrawn > 0) {
```

Example (Cont.):

```sol
            uint256 assetsDeposited = adapterNew.deposit(vault, user,
assetsWithdrawn);
            if (assetsWithdrawn > assetsDeposited) {
                emit Transfer(user, address(0), assetsWithdrawn -
assetsDeposited);
            } else if (assetsDeposited > assetsWithdrawn) {
                emit Transfer(address(0), user, assetsDeposited -
assetsWithdrawn);
            }
        }
    }
    _setVaultOf(user, vault);
} else {
    revert Errors.INVALID_VAULT(vault);
}
```

Recommendation:

We advise the branches to be merged into a single one, optimizing the code's gas cost.

Alleviation:

The logical branches were combined as recommended, optimizing the code's gas cost.

NTR-04C: Potential Mutability Optimization

Type	Severity	Location
Gas Optimization	Informational	NonTransferrableRebasingTokenVault.sol:L146, L148

Description:

The referenced configurational variables are assigned to only once during the contract's initialization hook (`NonTransferrableRebasingTokenVault::initialize`).

Example:

src/market/token/NonTransferrableRebasingTokenVault.sol

SOL

```sol
125 function initialize(
126     ISizeFactory sizeFactory_,
127     IPool aavePool_,
128     IERC20Metadata underlyingToken_,
129     address owner_,
130     string memory name_,
131     string memory symbol_,
132     uint8 decimals_
133 ) external initializer {
134     __Ownable_init(owner_);
```

Example (Cont.):

```sol
    __Ownable2Step_init();
    __ReentrancyGuard_init();
    __UUPSUpgradeable_init();

    if (
        address(sizeFactory_) == address(0) || address(aavePool_) == address(0)
            || address(underlyingToken_) == address(0)
    ) {
        revert Errors.NULL_ADDRESS();
    }

    sizeFactory = sizeFactory_;

    aavePool = aavePool_;
    underlyingToken = underlyingToken_;

    name = name_;
    symbol = symbol_;
    decimals = decimals_;
}
```

Recommendation:

We advise their assignments to be relocated to the contract's `NonTransferrableRebasingTokenVault::constructor`, permitting the variables to be set to `immutable` in a proxy-compatible manner.

Alleviation:

The Size Credit team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

NTR-05C: Redundant Boolean Variables

Type	Severity	Location
Gas Optimization	Informational	**NonTransferrableRebasingTokenVault.sol**: • I-1: **L479**, **L480**, **L482** • I-2: **L500**, **L501**

Description:

The referenced instances contain a redundant `bool` declaration as the variables will immediately be utilized and consumed in the `if` statement that follows them.

Example:

src/market/token/NonTransferrableRebasingTokenVault.sol

SOL

```
500 bool removed = vaultToIdMap.remove(vault);
501 if (removed) {
```

Recommendation:

We advise the expressions the booleans are assigned to to be relocated to each `if` conditional directly, optimizing the code's gas cost.

Alleviation:

The Size Credit team evaluated this exhibit but opted to acknowledge it in the current iteration of the codebase.

NTR-06C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	NonTransferrableRebasingTokenVault.sol:L171, L172, L408

Description:

The linked value literal is repeated across the codebase multiple times.

Example:

src/market/token/NonTransferrableRebasingTokenVault.sol

SOL

```sol
171 _setAdapter(bytes32("AaveAdapter"), aaveAdapter);
```

Recommendation:

We advise it to be set to a `constant` variable instead, optimizing the legibility of the codebase.

In case the `constant` has already been declared, we advise it to be properly re-used across the code.

Alleviation:

The referenced repetitive value literal (`"AaveAdapter"`) as well as its ERC4626 counterpart (`"ERC4626Adapter"`) were properly relocated to `constant` declarations, optimizing the code's maintainability.

Finding Types

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted if blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Circom language boasts that discerns it from other conventional programming languages.

Curve Specific

Circom defaults to using the BN128 scalar field (a 254-bit prime field), but it also supports BSL12-381 (which has a 255-bit scalar field) and Goldilocks (with a 64-bit scalar field). However, since there are no constants denoting either the prime or the prime size in bits available in the Circom language, some Circomlib templates like `Sign` (which returns the sign of the input signal), and `AliasCheck` (used by the strict versions of `Num2Bits` and `Bits2Num`), hardcode either the BN128 prime size or some other constant related to BN128. Using these circuits with a custom prime may thus lead to unexpected results and should be avoided.

Code Style

In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a local-level variable contains the same name as a toplevel variable in the circuit.

Mathematical Operations

This category is used when a mathematical issue is identified. This implies an issue with the implementation of a calculation compared to the specifications.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

Privacy Concern

This category is used when information that is meant to be kept private is made public in some way.

Proof Concern

Under-constrained signals are one of the most common issues in zero-knowledge circuits. Issues with proof generation fall under this category.

Severity Definition

In the ever-evolving world of blockchain technology, vulnerabilities continue to take on new forms and arise as more innovative projects manifest, new blockchain-level features are introduced, and novel layer-2 solutions are launched. When performing security reviews, we are tasked with classifying the various types of vulnerabilities we identify into subcategories to better aid our readers in understanding their impact.

Within this page, we will clarify what each severity level stands for and our approach in categorizing the findings we pinpoint in our audits. To note, all severity assessments are performed **as if the contract's logic cannot be upgraded** regardless of the underlying implementation.

Severity Levels

There are five distinct severity levels within our reports; `unknown`, `informational`, `minor`, `medium`, and `major`. A TL;DR overview table can be found below as well as a dedicated chapter to each severity level:

	Impact (None)	Impact (Low)	Impact (Moderate)	Impact (High)
Likelihood (None)	Informational	Informational	Informational	Informational
Likelihood (Low)	Informational	Minor	Minor	Medium
Likelihood (Moderate)	Informational	Minor	Medium	Major
Likelihood (High)	Informational	Medium	Major	Major

Unknown Severity

The `unknown` severity level is reserved for misbehaviors we observe in the codebase that cannot be quantified using the above metrics. Examples of such vulnerabilities include potentially desirable system behavior that is undocumented, reliance on external dependencies that are out-of-scope but could result in some form of vulnerability arising, use of external out-of-scope contracts that appears incorrect but cannot be pinpointed, and other such vulnerabilities.

In general, `unknown` severity level vulnerabilities require follow-up information by the project being audited and are either adjusted in severity (if valid), or marked as nullified (if invalid).

Additionally, the `unknown` severity level is sometimes assigned to centralization issues that cannot be assessed in likelihood due to their exploitation being tied to the honesty of the project's team.

Informational Severity

The `informational` severity level is dedicated to findings that do not affect the code functionally and tend to be stylistic or optimizational in nature. Certain edge cases are also set under `informational` vulnerabilities, such as overflow operations that will not manifest in the lifetime of the contract but should be guarded against as a best practice, to give an example.

Minor Severity

The `minor` severity level is meant for vulnerabilities that require functional changes in the code but tend to either have little impact or be unlikely to be recreated in a production environment. These findings can be acknowledged except for findings with a moderate impact but low likelihood which must be alleviated.

Medium Severity

The `medium` severity level is assigned to vulnerabilities that must be alleviated and have an observable impact on the overall project. These findings can only be acknowdged if the project deems them desirable behavior and we disagree with their point-of-view, instead urging them to reconsider their stance while marking the exhibit as acknowledged given that the project has ultimate say as to what vulnerabilities they end up patching in their system.

Major Severity

The `major` severity level is the maximum that can be specified for a finding and indicates a significant flaw in the code that must be alleviated.

Likelihood & Impact Assessment

As the preface chapter specifies, the blockchain space is constantly reinventing itself meaning that new vulnerabilities take place and our understanding of what security means differs year-to-year.

In order to reliably assess the likelihood and impact of a particular vulnerability, we instead apply an abstract measurement of a vulnerability's impact, duration the impact is applied for, and probability that the vulnerability would be exploited in a production environment.

Our proposed definitions are inspired by multiple sources in the security community and are as follows:

- Impact (High): A core invariant of the protocol can be broken for an extended duration.
- Impact (Moderate): A non-core invariant of the protocol can be broken for an extended duration or at scale, or an otherwise major-severity issue is reduced due to hypotheticals or external factors affecting likelihood.
- Impact (Low): A non-core invariant of the protocol can be broken with reduced likelihood or impact.
- Impact (None): A code or documentation flaw whose impact does not achieve low severity, or an issue without theoretical impact; a valuable best-practice
- Likelihood (High): A flaw in the code that can be exploited trivially and is ever-present.
- Likelihood (Moderate): A flaw in the code that requires some external factors to be exploited that are likely to manifest in practice.
- Likelihood (Low): A flaw in the code that requires multiple external factors to be exploited that may manifest in practice but would be unlikely to do so.
- Likelihood (None): A flaw in the code that requires external factors proven to be impossible in a production environment, either due to mathematical constraints, operational constraints, or system-related factors (i.e. EIP-20 tokens not being re-entrant).

Disclaimer

The following disclaimer applies to all versions of the audit report produced (preliminary / public / private) and is in effect for all past, current, and future audit reports that are produced and hosted under Omniscia:

IMPORTANT TERMS & CONDITIONS REGARDING OUR SECURITY AUDITS/REVIEWS/REPORTS AND ALL PUBLIC/PRIVATE CONTENT/DELIVERABLES

Omniscia ("Omniscia") has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. Omniscia and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

This audit report shall not be printed, saved, disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Omniscia.

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. Omniscia is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. Omniscia's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

Omniscia in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will Omniscia, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the “client”) with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

Omniscia will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

Omniscia will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.