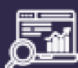
September 3, 2025

SMART CONTRACT AUDIT REPORT

Stake DAO
Strategy Wrapper

omniscia.io

info@omniscia.io


Online report: stake-dao-strategy-wrapper

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: stake-dao-strategy-wrapper

Strategy Wrapper Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
094552b4e3	August 25th 2025	1d25ebdf96
987d25d3c3	August 29th 2025	47e407d4e2
d31b9e5b82	September 3rd 2025	b1e8423877

Audit Overview

We were tasked with performing an audit of the Stake DAO codebase and in particular their Strategy Wrapper module.

The Strategy Wrapper implementation represents a non-transferrable **EIP-20** asset meant to integrate with the Morpho lending system to permit a strategy token to be supplied as collateral while continuing earning rewards within the Stake DAO ecosystem.

Over the course of the audit, we identified several issues around the liquidation functionality that is meant to supplement liquidations on the Morpho ecosystem that we believe need to be addressed promptly.

We advise the Stake DAO 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 Stake DAO 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 Stake DAO and have introduced additional information to be evaluated for exhibit: `SWR-03M`

Additionally, the following `informational` findings remain partially addressed and should be revisited: `SWR-01C`, `SWR-03C`

Post-Audit Conclusion (d31b9e5b82)

The Stake DAO team evaluated our follow-up recommendations on the aforementioned exhibits and proceeded with acknowledging `SWR-03M` whilst addressing `SWR-01C` and `SWR-03C`.

We evaluated the follow-up fixes that the Stake DAO team provided and confirmed that the `SWR-01C` and `SWR-03C` exhibits have been properly addressed in full.

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

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	1	1	0	0
Informational	8	6	0	2
Minor	1	0	0	1
Medium	1	0	0	1
Major	0	0	0	0

During the audit, we filtered and validated a total of **2 findings utilizing static analysis** tools as well as identified a total of **9 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/stake-dao/contracts-monorepo**
- Commit: 094552b4e36adae9cc47e327b422772be9e81e83
- Language: Solidity
- Network: Ethereum
- Revisions: **094552b4e3**, **987d25d3c3**, **d31b9e5b82**

Contracts Assessed

File	Total Finding(s)
packages/strategies/src/wrappers/StrategyWrapper.sol (SWR)	11

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 `0.8.28` based on the version specified within the `foundry.toml` file.

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

All `pragma` statements have been locked to `0.8.28`, 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 **17 potential issues** within the codebase of which **14 were ruled out to be false positives** or negligible findings.

The remaining **3 issues** were validated and grouped and formalized into the **2 exhibits** that follow:

ID	Severity	Addressed	Title
SWR-01S	Informational	Yes	Literal Equality of `bool` Variable
SWR-02S	Informational	Acknowledged	Multiple Top-Level Declarations

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Stake DAO's strategy wrapper.

As the project at hand implements a non-transferrable DeFi integrating EIP-20 asset, 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, kindly consult all exhibits that concern the liquidation functionality of the `StrategyWrapper`.

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, containing extensive in-line documentation that depicts what each function is meant to implement.

A total of **9 findings** were identified over the course of the manual review of which **3 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
SWR-01M	Unknown	Yes	Cross-Contract Security Assumption
SWR-02M	Minor	Acknowledged	Inexistent Association of Liquidator & Victim
SWR-03M	Medium	Acknowledged	Potential Self-Liquidation

Code Style

During the manual portion of the audit, we identified **6 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
SWR-01C	Informational	Yes	Inefficient Address Evaluations
SWR-02C	Informational	Yes	Inefficient Execution of Duplicate Operations
SWR-03C	Informational	Yes	Non-Standard Usage of Library
SWR-04C	Informational	Yes	Redundant Forceful Approval
SWR-05C	Informational	Acknowledged	Redundant Imposition of Separate Checks
SWR-06C	Informational	Yes	Repetitive Value Literal

StrategyWrapper Static Analysis Findings

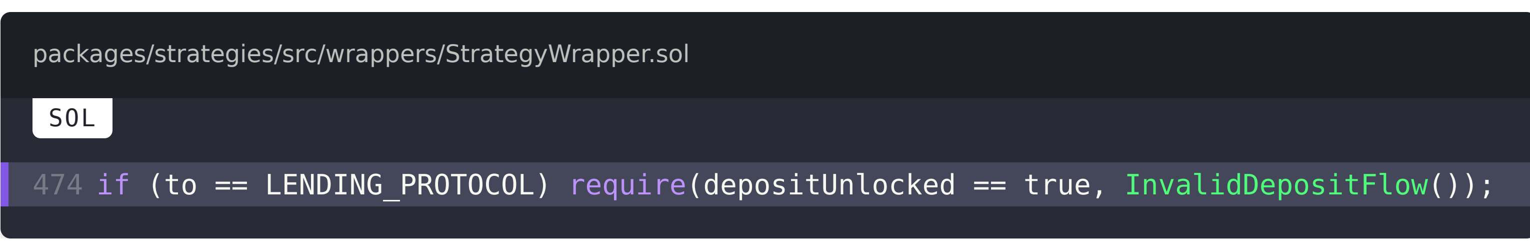
SWR-01S: Literal Equality of `bool` Variable

Type	Severity	Location
Gas Optimization	Informational	StrategyWrapper.sol:L474

Description:

The linked `bool` comparison is performed between a variable and a `bool` literal.

Example:



packages/strategies/src/wrappers/StrategyWrapper.sol

```sol
474 if (to == LENDING_PROTOCOL) require(depositUnlocked == true, InvalidDepositFlow());
```

Recommendation:

We advise the `bool` variable to be utilized directly either in its negated (`!`) or original form.

Alleviation (987d25d3c37124c40c5b9422b05cc12522201ffd):

The literal `bool` equality with `true` has been replaced by a direct evaluation of the `bool` variable, addressing this exhibit.

SWR-02S: Multiple Top-Level Declarations

Type	Severity	Location
Code Style	Informational	StrategyWrapper.sol:L15, L54

Description:

The referenced file contains multiple top-level declarations that decrease the legibility of the codebase.

Example:

packages/strategies/src/wrappers/StrategyWrapper.sol

SOL

```
15 interface IAccountant {
16     /// @notice Tracks reward and supply data for each vault.
17     /// @dev    Packed struct to minimize storage costs (3 storage slots).
18     ///         This structure maintains the core accounting state for reward
distribution.
19     struct VaultData {
20         uint256 integral; // Cumulative reward per token (scaled by
SCALING_FACTOR)
21         uint128 supply; // Total supply of vault tokens
22         uint128 feeSubjectAmount; // Amount of rewards subject to fees
23         uint128 totalAmount; // Total reward amount including fee-exempt rewards
24         uint128 netCredited; // Net rewards already credited to users (after
fees)
```

Example (Cont.):

```sol
25          uint128 reservedHarvestFee; // Harvest fees reserved but not yet paid out
26          uint128 reservedProtocolFee; // Protocol fees reserved but not yet accrued
27      }
28
29      /// @notice Tracks individual user positions within a vault.
30      /// @dev    Integral tracking enables O(1) reward calculations by storing the last known integral
31      ///         value for each user, allowing efficient computation of rewards earned since last update.
32      struct AccountData {
33          uint128 balance; // User's token balance in the vault
34          uint256 integral; // Last integral value when user's rewards were updated
35          uint256 pendingRewards; // Rewards earned but not yet claimed
36      }
37
38      function claim(address[] calldata _gauges, bytes[] calldata harvestData) external;
39      function vaults(address vault) external view returns (VaultData memory);
40      function accounts(address vault, address account) external view returns (AccountData memory);
41      function REWARD_TOKEN() external view returns (address);
42      function SCALING_FACTOR() external view returns (uint128);
43  }
44
45  /// @title Stake DAO Strategy Wrapper
46  /// @notice Non-transferable ERC20 wrapper for Stake DAO RewardVault shares. It is designed for use as collateral in lending markets.
47  /// @dev   - Allows users to deposit RewardVault shares or underlying LP tokens, and receive non-transferable tokens (1:1 ratio)
48  ///        - While the ERC20 tokens are held (even as collateral in lending markets), users can claim both main protocol rewards and extra rewards
49  ///        - Handles the edge case where the main reward token is also listed as an extra reward
50  ///        - Integrates with Stake DAO's reward and checkpointing logic to ensure users always receive the correct rewards, regardless of the custody
51  /// @author Stake DAO
52  /// @custom:contact contact@stakedao.org
```

Example (Cont.):

```sol
53 /// @custom:github https://github.com/stake-dao/contracts-monorepo
54 contract StrategyWrapper is ERC20, IStrategyWrapper, Ownable2Step,
ReentrancyGuardTransient {
```

Recommendation:

We advise all highlighted top-level declarations to be split into their respective code files, avoiding unnecessary imports as well as increasing the legibility of the codebase.

Alleviation (987d25d3c37124c40c5b9422b05cc12522201ffd):

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

StrategyWrapper Manual Review Findings

SWR-01M: Cross-Contract Security Assumption

Type	Severity	Location
Logical Fault	Unknown	**StrategyWrapper.sol**: • I-1: **L317** • I-2: **L439**

Description:

The `StrategyWrapper` implementation relies on the security assumption that it is not possible to claim on behalf of another user through the `RewardVault::claim` and `Accountant::claim` function implementations whereas both permit such functionality through proper authorization of system contracts.

Presently, such functionality is not possible as the only module that interacts with claims is the `RouterModuleClaim::claim` implementation, however, the code must never introduce a claim-on-behalf-of-another functionality in the future for the `StrategyWrapper` to function as expected.

Impact:

This exhibit outlines a latent vulnerability that can manifest in a future iteration and cannot have its severity properly assessed.

Example:

packages/strategies/src/wrappers/StrategyWrapper.sol

SOL

```solidity
438 // 2. Claim the rewards from the RewardVault
439 uint256[] memory amounts = REWARD_VAULT.claim(tokens, address(this));
440
441 // 3. Update the stored rewards for each extra reward token
442 for (uint256 i; i < tokens.length; i++) {
443     uint256 amount = amounts[i];
444     if (amount > 0) extraRewardPerToken[tokens[i]] += Math.mulDiv(amount, 1e18,
supply);
445 }
```

Recommendation:

We advise this dependency to be reflected in the integrated contracts through disclaimers.

Alleviation (987d25d3c37124c40c5b9422b05cc12522201ffd):

The documentation of the contract was updated to properly illustrate this critical security assumption, ensuring that future development efforts do not breach it.

As such, we consider this exhibit properly addressed.

SWR-02M: Inexistent Association of Liquidator & Victim

Type	Severity	Location
Logical Fault	Minor	StrategyWrapper.sol:L512-L543

Description:

The `StrategyWrapper::claimLiquidation` function can be invoked on behalf of any liquidator and victim and no association exists between those parties.

This permits a Denial-of-Service attack vector to manifest whereby a single liquidator splits their liquidation among several victims and thus causes full re-coup attempts to fail.

Impact:

It is possible to DoS liquidation claims by claiming a miniscule amount from victims of other liquidators.

Example:

packages/strategies/src/wrappers/StrategyWrapper.sol

SOL

```
512 function claimLiquidation(address liquidator, address victim, uint256
liquidatedAmount) external nonReentrant {
513     require(liquidator != address(0) && victim != address(0), ZeroAddress());
514     require(liquidatedAmount > 0, ZeroAmount());
515
516     // 1. Check that the liquidator own the claimed amount and that it's off-
track
517     //    Because token is untransferable (except by the lending protocol), the
only way
518     //    to have an off-track balance is to get it from liquidation executed by
the lending protocol.
519     //    We have to be sure the claimed amount is off-track, otherwise any
holders will be able to liquidate any positions.
520     require(balanceOf(liquidator) >= userCheckpoints[liquidator].balance +
liquidatedAmount, InvalidLiquidation());
521
```

Example (Cont.):

```sol
522     // 2. Check that the victim has enough tokens registered internally
523     uint256 internalVictimBalance = userCheckpoints[victim].balance;
524     require(internalVictimBalance >= liquidatedAmount, InvalidLiquidation());
525
526     // 3. Check that the victim as a real holding hole of at least the claimed
amount
527     //    This is done by summing up the collateral supplied by the victim with
his balance.
528     //    Because token is untransferable (except by the lending protocol), the
only way
529     //    to have an off-track balance is to get liquidated.
530     uint256 victimBalance =
531         balanceOf(victim) +
IMorpho(LENDING_PROTOCOL).position(Id.wrap(lendingMarketId), victim).collateral;
532     require(internalVictimBalance - victimBalance >= liquidatedAmount,
InvalidLiquidation());
533
534     // 4. Claim the accrued rewards for the victim and reduce his internal
balance
535     _claimAllRewards(victim);
536     userCheckpoints[victim].balance -= liquidatedAmount;
537
538     // 5. Burn the liquidated amount and transfer the underlying RewardVault
shares back to the liquidator
539     _burn(liquidator, liquidatedAmount);
540     SafeERC20.safeTransfer(IERC20(address(REWARD_VAULT)), liquidator,
liquidatedAmount);
541
542     emit Liquidated(liquidator, victim, liquidatedAmount);
543 }
```

Recommendation:

The vulnerability described is difficult to counter-act as it would require an association between liquidators and victims in the Morpho system to be reflected in the Stake DAO system.

Our recommended approach would be for liquidations to flow through the `StrategyWrapper` itself which would permit the liquidation to be claimed immediately as well.

Another approach could be to track liquidation fund batches received and to force liquidation claims to occur in the amounts received via liquidations rather than any subset amounts, minimizing the potential impact of this issue to same-value liquidations.

Alleviation (987d25d3c37124c40c5b9422b05cc12522201ffd):

The Stake DAO team evaluated this exhibit and analyzed the potential likelihood it would manifest in depth, ultimately opting to acknowledge it.

We concur with the **analysis chapter** of the Stake DAO team and have thus downgraded this exhibit to minor with an acknowledged rating.

SWR-03M: Potential Self-Liquidation

Type	Severity	Location
Logical Fault	Medium	StrategyWrapper.sol:L512-L543

Description:

The `StrategyWrapper::claimLiquidation` function does not validate against a self-liquidation event, permitting a user who has received liquidated funds from another to liquidate themselves.

Combined with the absence of access control in the `StrategyWrapper::claimLiquidation` function, it is possible for such a user to have their funds withdrawn in an unauthorized manner.

Impact:

A self-liquidation is possible if a user has an active balance in the `StrategyWrapper` and has received liquidation proceeds.

This state can be exploited by malicious actors to repetitively attempt to "liquidate" the extra balance from the liquidator themselves, resulting in these actions being processed as normal withdrawals and thus resulting in a Denial-of-Service for the liquidator.

Example:

packages/strategies/src/wrappers/StrategyWrapper.sol

SOL

```
512 function claimLiquidation(address liquidator, address victim, uint256
liquidatedAmount) external nonReentrant {
513     require(liquidator != address(0) && victim != address(0), ZeroAddress());
514     require(liquidatedAmount > 0, ZeroAmount());
515
516     // 1. Check that the liquidator own the claimed amount and that it's off-
track
517     //    Because token is untransferable (except by the lending protocol), the
only way
518     //    to have an off-track balance is to get it from liquidation executed by
the lending protocol.
519     //    We have to be sure the claimed amount is off-track, otherwise any
holders will be able to liquidate any positions.
520     require(balanceOf(liquidator) >= userCheckpoints[liquidator].balance +
liquidatedAmount, InvalidLiquidation());
521
```

Example (Cont.):

```sol
522     // 2. Check that the victim has enough tokens registered internally
523     uint256 internalVictimBalance = userCheckpoints[victim].balance;
524     require(internalVictimBalance >= liquidatedAmount, InvalidLiquidation());
525
526     // 3. Check that the victim as a real holding hole of at least the claimed
amount
527     //    This is done by summing up the collateral supplied by the victim with
his balance.
528     //    Because token is untransferable (except by the lending protocol), the
only way
529     //    to have an off-track balance is to get liquidated.
530     uint256 victimBalance =
531         balanceOf(victim) +
IMorpho(LENDING_PROTOCOL).position(Id.wrap(lendingMarketId), victim).collateral;
532     require(internalVictimBalance - victimBalance >= liquidatedAmount,
InvalidLiquidation());
533
534     // 4. Claim the accrued rewards for the victim and reduce his internal
balance
535     _claimAllRewards(victim);
536     userCheckpoints[victim].balance -= liquidatedAmount;
537
538     // 5. Burn the liquidated amount and transfer the underlying RewardVault
shares back to the liquidator
539     _burn(liquidator, liquidatedAmount);
540     SafeERC20.safeTransfer(IERC20(address(REWARD_VAULT)), liquidator,
liquidatedAmount);
541
542     emit Liquidated(liquidator, victim, liquidatedAmount);
543 }
```

Recommendation:

We advise the system to prevent a self-liquidation event as it will result in an exploitable Denial-of-Service attack vector.

Alleviation (987d25d3c3):

The Stake DAO team evaluated this exhibit and specified that they consider it an impossible case due to the restrictions that the `StrategyWrapper::claimLiquidation` enforces.

Specifically, the last condition relevant to collateral would fail if a self-liquidation would be attempted due to evaluating the dynamic balance of the victim rather than the recorded balance.

We would like to note that this assessment paves the way to another issue; liquidation proceeds can be utilized to "recoup" liquidated amounts. This can break the `liquidator` <-> `victim` that the Morpho system would emit via events, and it is a risk we consider to be minor but able to be acknowledged.

Alleviation (d31b9e5b82):

The Stake DAO team assessed the ramifications of this issue and has opted to acknowledge it as they do not envision any significant or non-rectifiable impact to arise from the exploitation of this disassociation.

StrategyWrapper Code Style Findings

SWR-01C: Inefficient Address Evaluations

Type	Severity	Location
Gas Optimization	Informational	StrategyWrapper.sol:L142

Description:

Two of the referenced address evaluations are redundant; the `_owner` is evaluated by the `Ownable` dependency directly whereas the `rewardVault` has functions invoked on it which prevent a zero-address from being defined.

Example:

packages/strategies/src/wrappers/StrategyWrapper.sol

SOL

```
140 constructor(IRewardVault rewardVault, address lendingProtocol, address _owner)
ERC20("", "") Ownable(_owner) {
141     require(
142         address(rewardVault) != address(0) && lendingProtocol != address(0) &&
_owner != address(0), ZeroAddress()
143     );
144
145     IAccountant accountant = IAccountant(address(rewardVault.ACCOUNTANT()));
146
147     // Store the immutable variables
148     GAUGE = rewardVault.gauge();
149     ACCOUNTANT = accountant;
```

Example (Cont.):

```sol
150     REWARD_VAULT = rewardVault;
151     MAIN_REWARD_TOKEN = IERC20(accountant.REWARD_TOKEN());
152     ACCOUNTANT_SCALING_FACTOR = accountant.SCALING_FACTOR();
153     LENDING_PROTOCOL = lendingProtocol;
154
155     // Approve the asset token to the reward vault and the wrapped token to the
lending protocol
156     SafeERC20.forceApprove(IERC20(rewardVault.asset()), address(rewardVault),
type(uint256).max);
157     _approve(address(this), lendingProtocol, type(uint256).max, true);
158 }
```

Recommendation:

We advise those two evaluations to be omitted, optimizing the contract's deployment gas cost.

Alleviation (987d25d3c3):

The system continues to evaluate a non-zero `rewardVault` inefficiently, rendering this exhibit partially addressed.

Alleviation (d31b9e5b82):

The `rewardVault` address evaluation has been omitted, further optimizing the code's gas cost.

SWR-02C: Inefficient Execution of Duplicate Operations

Type	Severity	Location
Gas Optimization	Informational	StrategyWrapper.sol:L386, L390, L394

Description:

The `StrategyWrapper::_updateUserCheckpoint` function will be significantly inefficient in case a user has a non-zero balance; a scenario that represents the highest percentage of transaction executions.

Example:

packages/strategies/src/wrappers/StrategyWrapper.sol

SOL

```
383 function _updateUserCheckpoint(address account, uint256 amount) internal {
384     // 1. Update the internal account checkpoint
385     UserCheckpoint storage checkpoint = userCheckpoints[account];
386     if (checkpoint.balance != 0) _claimAllRewards(account);
387     checkpoint.balance += amount;
388 
389     // 2. Keep track of the Main reward token checkpoint
390     checkpoint.rewardPerTokenPaid[MAIN_REWARD_TOKEN_SLOT] = _getGlobalIntegral();
391 
392     // 3. Keep track of the Extra reward tokens checkpoints at deposit time
```

Example (Cont.):

```sol
393     address[] memory rewardTokens = REWARD_VAULT.getRewardTokens();
394     _updateExtraRewardState(rewardTokens);
395     for (uint256 i; i < rewardTokens.length; i++) {
396         checkpoint.rewardPerTokenPaid[rewardTokens[i]] =
extraRewardPerToken[rewardTokens[i]];
397     }
398 }
```

Recommendation:

We advise the code to configure the `MAIN_REWARD_TOKEN_SLOT` and update the extra reward state solely when the user's original balance was `0`, optimizing the code greatly.

To note, this optimization would require the `userCheckpoint.rewardPerTOkenPaid` value of the `MAIN_REWARD_TOKEN_SLOT` to be set regardless of whether an amount was claimed in the **`StrategyWrapper::_claim`** function.

Alleviation (987d25d3c37124c40c5b9422b05cc12522201ffd):

The code was revised per our recommendation, greatly optimizing how user checkpoints are recorded.

SWR-03C: Non-Standard Usage of Library

Type	Severity	Location
Code Style	Informational	**StrategyWrapper.sol**: • I-1: **L156** • I-2: **L210** • I-3: **L237** • I-4: **L286** • I-5: **L540**

Description:

The referenced statements will utilize the `SafeERC20` library via direct invocations which is non-standard.

Example:

packages/strategies/src/wrappers/StrategyWrapper.sol

SOL

```
156 SafeERC20.forceApprove(IERC20(rewardVault.asset()), address(rewardVault),
type(uint256).max);
```

Recommendation:

We advise the `using SafeERC20 for IERC20` statement to be introduced to the `StrategyWrapper` contract, permitting the relevant `SafeERC20` functions to be exposed directly through an `IERC20` data type variable.

Alleviation (987d25d3c3):

The Stake DAO team consider the current invocation approach to be more legible.

We would like to note that invoking the library directly instead of through a data type may result in a `delegatecall` operation instead of local code execution thereby increasing the code's gas cost.

Alleviation (d31b9e5b82):

All library invocations have been replaced via function invocations as exposed by the relevant library data type `IERC20`, optimizing the code's gas cost as well as legibility.

SWR-04C: Redundant Forceful Approval

Type	Severity	Location
Gas Optimization	Informational	StrategyWrapper.sol:L156

Description:

The referenced operation will invoke the `SafeERC20::forceApprove` function even though the approval operation is located in the constructor of a newly deployed contract.

Example:

packages/strategies/src/wrappers/StrategyWrapper.sol

SOL

```
156 SafeERC20.forceApprove(IERC20(rewardVault.asset()), address(rewardVault), type(uint256).max);
```

Recommendation:

We advise a direct approve operation to be performed, optimizing the code's gas cost.

Alleviation (987d25d3c37124c40c5b9422b05cc12522201ffd):

The code was updated to utilize a direct approve operation efficiently, addressing this exhibit.

SWR-05C: Redundant Imposition of Separate Checks

Type	Severity	Location
Gas Optimization	Informational	StrategyWrapper.sol:L524, L532

Description:

The first referenced `require` check is guaranteed by the second referenced `require` check, rendering the former redundant.

Example:

packages/strategies/src/wrappers/StrategyWrapper.sol

```sol
522 // 2. Check that the victim has enough tokens registered internally
523 uint256 internalVictimBalance = userCheckpoints[victim].balance;
524 require(internalVictimBalance >= liquidatedAmount, InvalidLiquidation());
525
526 // 3. Check that the victim as a real holding hole of at least the claimed amount
527 //    This is done by summing up the collateral supplied by the victim with his
balance.
528 //    Because token is untransferable (except by the lending protocol), the only
way
529 //    to have an off-track balance is to get liquidated.
530 uint256 victimBalance =
531     balanceOf(victim) +
IMorpho(LENDING_PROTOCOL).position(Id.wrap(lendingMarketId), victim).collateral;
```

Example (Cont.):

```sol
532 require(internalVictimBalance - victimBalance >= liquidatedAmount,
InvalidLiquidation());
```

Recommendation:

We advise only the latter of the two checks to be imposed, optimizing the code's gas cost.

Alleviation (987d25d3c37124c40c5b9422b05cc12522201ffd):

The Stake DAO team has opted to acknowledge this optimization as they consider the additional documentation and extra cost incurred by redundancy to outweigh the benefits provided to this critical function of the system.

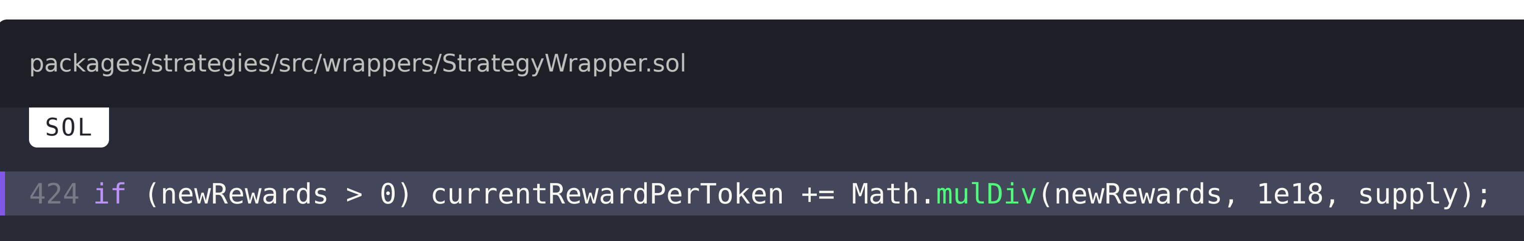
SWR-06C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	StrategyWrapper.sol:L424, L427, L444

Description:

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

Example:



packages/strategies/src/wrappers/StrategyWrapper.sol

```sol
424 if (newRewards > 0) currentRewardPerToken += Math.mulDiv(newRewards, 1e18, supply);
```

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 (987d25d3c37124c40c5b9422b05cc12522201ffd):

The referenced value literal `1e18` has been properly relocated to a contract-level `constant` declaration labelled `PRECISION`, optimizing the code's legibility.

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.