OMNISCIA

May 20, 2025

SMART CONTRACT AUDIT REPORT

Stake DAO
Strategies


omniscia.io


info@omniscia.io


Online report: stake-dao-strategies

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-strategies

Strategies Security Audit

Audit Report Revisions

Commit Hash	Date	Audit Report Hash
6a0d6826b4	May 8th 2025	93aad9d444
95f5a12e9f	May 15th 2025	0ea3c14a24
95f5a12e9f	May 20th 2025	e141ae522b

Audit Overview

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

The system is broken down across several modules each responsible for critical functionality of the overall fund disbursement flow a strategy is meant to undertake. Specifically:

- An `Allocator` implementation is responsible for assessing the correct proportions toward several targets to ensure yield-maximizing deposit / withdrawal operations
- A `RewardReceiver` is responsible for relaying any rewards toward the `RewardVault` for distribution
- A `Strategy` implementation is dedicated for handling input funds across the protocols itself supports as well as its sister contracts termed "sidecars"
- A `Sidecar` implementation is meant to accompany a main `Strategy` implementation to introduce additional logic, such as Convex alongside Curve Finance
- An `Accountant` implementation that is meant to track all reward and balance changes across a particular vault, integrating with the relevant strategies where needed
- A `RewardVault` meant to combine the functionality of all aforementioned modules into an **EIP-4626** user-facing vault that users can deposit funds to and withdraw from
- A `Router` implementation meant to streamline access to the underlying `RewardVault` via easy-to-use utility functions

The instance of the codebase we evaluated contained a Curve Finance strategy implementation accompanied by a Convex Finance sidecar alongside `Router` modules that support migrations on top of typical **EIP-4626** functionality.

Over the course of the audit, we identified several significant issues revolving around authorizations, logical flaws, integrations, and arithmetic issues.

Additionally, we observed that any sidecar of a strategy has no fees imposed on it. While we considered this a desirable system trait during the course of the audit, we invite the Stake DAO team to closely evaluate it.

Finally, whilst the system is meant to be highly dynamic and support future strategies we observed certain approaches that would render it incompatible with a significant portion of DeFi protocols.

For example, withdrawals from secondary protocols are expected to be performed without any fees imposed when calculating pending rewards. While the current integration of the system with Curve Finance and Convex Finance does not suffer from this flaw, the system would require a slight overhaul to become compatible with a greater range of DeFi systems.

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 provided a follow-up recommendation in relation to a single exhibit. We advise the Stake DAO team to revisit the following exhibit: `ATN-01M`

We consider all other exhibits within the audit report to have either been properly alleviated or safely acknowledged.

Post-Audit Conclusion (Cont.)

The Stake DAO team evaluated our follow-up recommendation on exhibit `ATN-01M` and opted to retain it as acknowledged.

As such, 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	48	16	0	32
Minor	8	7	0	1
Medium	6	5	0	1
Major	7	7	0	0

During the audit, we filtered and validated a total of **6 findings utilizing static analysis** tools as well as identified a total of **64 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: 6a0d6826b4c4e44de97dc45f6e14f2b9a5bd857f
- Language: Solidity
- Network: Ethereum
- Revisions: **6a0d6826b4**, **95f5a12e9f**

Contracts Assessed

File	Findir
packages/strategies/src/Allocator.sol (ARO)	
packages/strategies/src/Accountant.sol (ATN)	
packages/strategies/src/integrations/curve/CurveFactory.sol (CFY)	
packages/strategies/src/integrations/curve/ConvexSidecar.sol (CSR)	
packages/strategies/src/integrations/curve/CurveStrategy.sol (CSY)	
packages/strategies/src/integrations/curve/CurveAllocator.sol (CAR)	
packages/strategies/src/integrations/curve/ConvexSidecarFactory.sol (CSF)	
packages/strategies/src/Factory.sol (FYR)	
packages/strategies/src/libraries/ImmutableArgsParser.sol (IAP)	
packages/strategies/src/ProtocolContext.sol (PCT)	

packages/strategies/src/ProtocolController.sol (PCR)

packages/strategies/src/Router.sol (RRE)

packages/strategies/src/RewardVault.sol (RVT)

packages/strategies/src/RewardReceiver.sol (RRR)

packages/strategies/src/RouterModules/RouterModuleClaim.sol (RMC)

packages/strategies/src/RouterModules/RouterModuleDeposit.sol (RMD)

packages/strategies/src/RouterModules/RouterModuleWithdraw.sol (RMW)

packages/strategies/src/RouterModules/RouterIdentifierMapping.sol (RIM)

packages/strategies/src/RouterModules/RouterModuleMigrationCurve.sol (RMM)

packages/strategies/src/RouterModules/RouterModuleMigrationYearn.sol (RMY)

packages/strategies/src/RouterModules/RouterModuleMigrationStakeDAOV1.sol (RM

packages/strategies/src/Sidecar.sol (SRA)

packages/strategies/src/Strategy.sol (SYG)

packages/strategies/src/SidecarFactory.sol (SFY)

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 as the `pragma` statements of the contracts are open-ended (`^0.8.20`).

We advise them to be locked to `0.8.28` (`=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 **96 potential issues** within the codebase of which **88 were ruled out to be false positives** or negligible findings.

The remaining **8 issues** were validated and grouped and formalized into the **6 exhibits** that follow:

ID	Severity	Addressed	Title
CAR-01S	Informational	Acknowledged	Inexistent Sanitization of Input Address
CFY-01S	Informational	Acknowledged	Inexistent Sanitization of Input Address
CSY-01S	Informational	Acknowledged	Inexistent Sanitization of Input Address
RVT-01S	Informational	Yes	Literal Equality of `bool` Variable
RMS-01S	Informational	Acknowledged	Multiple Top-Level Declarations
RMY-01S	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 implementations.

As the project at hand implements a complex multi-strategy vault system, 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 **severe ramifications** to its overall operation; for more information, kindly consult the relevant medium and major severity exhibits within the audit report.

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 certain extent, however, we strongly recommend it to be expanded at certain complex points such as the `CurveAllocator::getOptimalLockerBalance` function implementation whose purpose is indeterminate.

A total of **64 findings** were identified over the course of the manual review of which **27 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
ATN-01M	Medium	Acknowledged	Inexistent Validation of Harvest Policy
CSR-01M	Informational	Yes	Misleading Documentation
CSF-01M	Minor	Yes	Inexistent Validation of Argument Length
CAR-01M	Unknown	Yes	Inadequate Documentation of Function
CFY-01M	Minor	Yes	Potentially Unhandled Decoding Error

CFY-02M	Minor	Yes	Potentially Weak Restriction of Token
CFY-03M	Major	Yes	Incorrect Strategy Integration
CSY-01M	Medium	Yes	Incorrect Checkpoint of Liquidity Gauge Rewards
CSY-02M	Medium	Nullified	Potentially Incorrect Curve Integration
RRR-01M	Informational	Yes	Deprecated Approval Operation
RVT-01M	Informational	Yes	Potentially Incorrect Token Name
RVT-02M	Minor	Yes	Incorrect Usage of Mathematical Library
RVT-03M	Minor	Yes	Potentially Incorrect Integration of Reward Evaluation
RVT-04M	Medium	Yes	Incorrect Maximum Limitation Evaluations

RVT-05M	Medium	Yes	Insecure Casting Operations
RVT-06M	Major	Nullified	Incorrect Condition Enforcement
RVT-07M	Major	Yes	Inexistent Validation of Authorization
RRE-01M	Minor	Yes	Inconsistent Module Name Resolution Behaviour
RRE-02M	Minor	Acknowledged	Incorrect Module Enumeration
RMC-01M	Informational	Yes	Misleading Function Restriction
RMC-02M	Medium	Yes	Inexistent Validation of Authorization
RMD-01M	Informational	Yes	Misleading Function Restrictions
RMD-02M	Minor	Yes	Incorrect Payable Attribute

RMM-01M	Major	Yes	Inexistent Validation of Vault Addresses
RMS-01M	Major	Yes	Inexistent Validation of Vault Addresses
RMY-01M	Major	Yes	Inexistent Validation of Vault Addresses
RMW-01M	Major	Yes	Insecure Execution of Withdrawals

Code Style

During the manual portion of the audit, we identified **37 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
ATN-01C	Informational	Acknowledged	Ineffectual Usage of Safe Arithmetics
ATN-02C	Informational	Acknowledged	Non-Descriptive Harvest Policies
ATN-03C	Informational	Acknowledged	Non-Standard Evaluation of New Rewards
ATN-04C	Informational	Yes	Redundant Fee Restriction
ATN-05C	Informational	Acknowledged	Repetitive Value Literal
ARO-01C	Informational	Acknowledged	Redundant Variable
CSR-01C	Informational	Yes	Inefficient Logical Branch
CSR-02C	Informational	Acknowledged	Inefficient Re-Queries of Values
CSR-03C	Informational	Yes	Inefficient Usage of Allowance Increase
CAR-01C	Informational	Acknowledged	Ineffectual Usage of Safe Arithmetics

CAR-02C	Informational	Yes	Inefficient Function Implementation
CAR-03C	Informational	Acknowledged	Repetitive Value Literals
CFY-01C	Informational	Acknowledged	Inefficient ABI Encoding Mechanisms
CFY-02C	Informational	Acknowledged	Inefficient Loop Iterator
CFY-03C	Informational	Acknowledged	Redundant Gauge Validity Flag
CSY-01C	Informational	Acknowledged	Inefficient ABI Encoding Mechanisms
CSY-02C	Informational	Acknowledged	Redundant Code Repetition
FYR-01C	Informational	Acknowledged	Redundant Validation of Protocol Controller
IAP-01C	Informational	Acknowledged	Repetitive Value Literal
PCT-01C	Informational	Acknowledged	Inefficient ABI Encoding of Payload
PCR-01C	Informational	Acknowledged	Inefficient `mapping` Lookups
RVT-01C	Informational	Acknowledged	Ineffectual Usage of Safe Arithmetics
RVT-02C	Informational	Acknowledged	Non-Standard Usages of Library
RVT-03C	Informational	Yes	Redundant Invocations of String Concatenations

RVT-04C	Informational	Acknowledged	Repetitive Value Literal
RRE-01C	Informational	Yes	Inefficient Application of Modifier
RRE-02C	Informational	Acknowledged	Inefficient Loop Iterator
RIM-01C	Informational	Acknowledged	Inefficient Data Type Usage
RMM-01C	Informational	Yes	Inefficient Import of Logic Implementation
RMM-02C	Informational	Acknowledged	Redundant Code Duplication
RMS-01C	Informational	Yes	Inefficient Import of Logic Implementation
RMY-01C	Informational	Yes	Inefficient Import of Logic Implementation
RMY-02C	Informational	Acknowledged	Redundant Code Duplication
SRA-01C	Informational	Acknowledged	Redundant Validation of Argument
SFY-01C	Informational	Acknowledged	Redundant Validation of Argument
SYG-01C	Informational	Yes	Non-Standard Empty Implementation
SYG-02C	Informational	Acknowledged	Non-Standard Usage Statement

CurveAllocator Static Analysis Findings

CAR-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	CurveAllocator.sol:L29-L31

Description:

The linked function accepts an `address` argument yet does not properly sanitize it.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

packages/strategies/src/integrations/curve/CurveAllocator.sol

SOL

```sol
29  constructor(address _locker, address _gateway, address _convexSidecarFactory)
Allocator(_locker, _gateway) {
30      CONVEX_SIDECAR_FACTORY = ISidecarFactory(_convexSidecarFactory);
31  }
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that the `address` specified is non-zero.

Alleviation:

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

CurveFactory Static Analysis Findings

CFY-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	CurveFactory.sol:L38-L49

Description:

The linked function accepts an `address` argument yet does not properly sanitize it.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

packages/strategies/src/integrations/curve/CurveFactory.sol

SOL

```sol
38  constructor(
39      address protocolController,
40      address vaultImplementation,
41      address rewardReceiverImplementation,
42      address locker,
43      address gateway,
44      address convexSidecarFactory
45  )
46      Factory(protocolController, vaultImplementation,
rewardReceiverImplementation, CURVE_PROTOCOL_ID, locker, gateway)
47  {
```

Example (Cont.):

```sol
    CONVEX_SIDECAR_FACTORY = convexSidecarFactory;
}
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that the `address` specified is non-zero.

Alleviation:

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

CurveStrategy Static Analysis Findings

CSY-01S: Inexistent Sanitization of Input Address

Type	Severity	Location
Input Sanitization	Informational	CurveStrategy.sol:L42-L46

Description:

The linked function accepts an `address` argument yet does not properly sanitize it.

Impact:

The presence of zero-value addresses, especially in `constructor` implementations, can cause the contract to be permanently inoperable. These checks are advised as zero-value inputs are a common side-effect of off-chain software related bugs.

Example:

packages/strategies/src/integrations/curve/CurveStrategy.sol

SOL

```sol
42  constructor(address _registry, address _locker, address _gateway, address
_minter)
43      Strategy(_registry, CURVE_PROTOCOL_ID, _locker, _gateway)
44  {
45      MINTER = _minter;
46  }
```

Recommendation:

We advise some basic sanitization to be put in place by ensuring that the `address` specified is non-zero.

Alleviation:

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

RewardVault Static Analysis Findings

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

Type	Severity	Location
Gas Optimization	Informational	RewardVault.sol:L463

Description:

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

Example:

packages/strategies/src/RewardVault.sol

SOL

```
463 require(_isRewardToken(reward) == false, RewardAlreadyExists());
```

Recommendation:

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

Alleviation:

The referenced boolean evaluation was optimized in legibility by negating its result rather than comparing it with `false` via an equality.

RouterModuleMigrationStakeDAOV1 Static Analysis Findings

RMS-01S: Multiple Top-Level Declarations

Type	Severity	Location
Code Style	Informational	RouterModuleMigrationStakeDAOV1.sol:L9, L15

Description:

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

Example:

packages/strategies/src/RouterModules/RouterModuleMigrationStakeDAOV1.sol

SOL

```sol
interface IVault {
    function token() external view returns (address);
    function withdraw(uint256 shares) external;
    function liquidityGauge() external view returns (address);
}

contract RouterModuleMigrationStakeDAOV1 is IRouterModule {
```

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:

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

RouterModuleMigrationYearn Static Analysis Findings

RMY-01S: Multiple Top-Level Declarations

Type	Severity	Location
Code Style	Informational	RouterModuleMigrationYearn.sol:L9, L39

Description:

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

Example:

packages/strategies/src/RouterModules/RouterModuleMigrationYearn.sol

SOL

```sol
9   contract RouterModuleMigrationYearn is IRouterModule {
10      using SafeERC20 for IYearnVault;
11
12      string public constant name = type(RouterModuleMigrationYearn).name;
13      string public constant version = "1.0.0";
14
15      error VaultNotCompatible();
16
17      /// @notice Migrates shares from convex to a reward vault
18      /// @dev The account must have approved the token to the router contract
```

Example (Cont.):

```sol
    /// @param from The address of the old vault
    /// @param to The address of the new reward vault
    /// @param account The address of the account to migrate
    /// @param shares The number of shares to migrate
    function migrate(address from, address to, address account, uint256 shares) external {
        address asset = RewardVault(to).asset();
        require(IYearnVault(from).token() == asset, VaultNotCompatible());

        // 1. Transfer the token of the user to the router contract
        IYearnVault(from).safeTransferFrom(account, address(this), shares);

        // 2. Withdraw the tokens in the old vault
        IYearnVault(from).withdraw(shares);

        // 3. Deposit the tokens in the reward vault
        IYearnVault(asset).safeIncreaseAllowance(to, shares);
        RewardVault(to).deposit(shares, account);
    }
}

interface IYearnVault is IERC20 {
```

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:

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

Accountant Manual Review Findings

ATN-01M: Inexistent Validation of Harvest Policy

Type	Severity	Location
Logical Fault	Medium	Accountant.sol:L495

Description:

The `Accountant::harvest` function permits any gauge and thus strategy to be specified for a manual harvest operation.

This permits a strategy with a harvest policy of `HARVEST` to be manually harvested, thereby incurring a harvest fee incorrectly.

Impact:

It is possible to manipulate the `Accountant` into applying a harvest fee on vaults that normally harvest rewards on all checkpoint operations.

Example:

packages/strategies/src/Accountant.sol

SOL

```
486 // First pass: harvest all gauges and update vault states
487 for (uint256 i; i < _gauges.length; i++) {
488     address gauge = _gauges[i];
489     address vault = PROTOCOL_CONTROLLER.vaults(gauge);
490     require(vault != address(0), InvalidVault());
491
492     VaultData storage _vault = vaults[vault];
493
494     // 1. Pull rewards from the strategy
495     IStrategy.PendingRewards memory pendingRewards =
IStrategy(strategy).harvest(gauge, harvestData[i]);
```

Example (Cont.):

```sol
496
497     // 2. Under-delivery guard (totals must cover soft credits)
498     require(
499         pendingRewards.totalAmount >= _vault.totalAmount
500             && pendingRewards.feeSubjectAmount >= _vault.feeSubjectAmount,
501         InsufficientPendingRewards()
502     );
503
504     /// @dev Guard against fees exceeding the rewards.
505     require(pendingRewards.feeSubjectAmount <= pendingRewards.totalAmount,
FeesExceedRewards());
506
507     // Skip if strategy reports zero
508     if (pendingRewards.totalAmount == 0) continue;
509     // 3. Aggregate global rewards
510     totalRewardsAmount += pendingRewards.totalAmount;
511
512     // 4. Start accounting
513     uint256 protocolFee = _vault.reservedProtocolFee;
514     uint256 harvesterFee = _vault.reservedHarvestFee;
515
516     uint256 netAfterReservedFees = pendingRewards.totalAmount - protocolFee -
harvesterFee;
517     uint256 netCredited = _vault.netCredited;
518
519     // Strategy must at least make LPs whole
520     require(netAfterReservedFees >= netCredited, NetCreditsNotEnough());
521
522     if (netAfterReservedFees > netCredited && _vault.supply > 0) {
523         uint256 newRewards = netAfterReservedFees - netCredited;
```

Example (Cont.):

```sol
524
525         /// Take the fee from the net delta
526         uint256 newHarvesterFee = newRewards.mulDiv(getHarvestFeePercent(), 1e18);
```

Recommendation:

We advise the code to ensure that the `RewardVault::POLICY` of the `vault` is `CHECKPOINT` so as to prevent vaults that harvest on each deposit, withdrawal, and transfer from being processed via the `Accountant::harvest` function.

Alleviation (95f5a12e9f):

The Stake DAO team evaluated this exhibit and clarified that they consider this behaviour intentional.

Specifically, they envision a manual harvest operation to occur solely after a period of significant inactivity and thereby to warrant a fee.

We would like to note that this results in inconsistent behaviour as a holder of the vault might trigger an interaction to capture fees if they detect a manual harvest operation being submitted, effectively "robbing" the manual harvester of fees.

To ensure behaviour remains consistent with the desired specification, we advise a harvest fee to be imposed if a period of inactivity has been observed rather than if a manual harvest has been specified for `HARVEST` policies to prevent a bypass of a manual harvest operation.

Alleviation (95f5a12e9f):

The Stake DAO team considered our recommendation and decided to acknowledge the inconsistency outlined as they do not consider it a significant flaw.

ConvexSidecar Manual Review Findings

CSR-01M: Misleading Documentation

Type	Severity	Location
Logical Fault	Informational	ConvexSidecar.sol:L17-L18

Description:

The referenced documentation indicates that a Convex protocol ID should be established yet the code establishes a Curve protocol ID.

Example:

packages/strategies/src/integrations/curve/ConvexSidecar.sol

SOL

```sol
17  /// @notice The bytes4 ID of the Convex protocol
18  /// @dev Used to identify the Convex protocol in the registry
19  bytes4 private constant CURVE_PROTOCOL_ID = bytes4(keccak256("CURVE"));
```

Recommendation:

We advise either the documentation or the ID generation to be corrected.

Alleviation:

The documentation has been corrected per our recommendation, addressing this exhibit.

ConvexSidecarFactory Manual Review Findings

CSF-01M: Inexistent Validation of Argument Length

Type	Severity	Location
Input Sanitization	Minor	ConvexSidecarFactory.sol:L44

Description:

The `args` parameter for a particular Convex sidecar creation is not validated as having a length of `32` bytes.

Impact:

Any `SidecarCreated` event emitted by the `ConvexSidecar` might have a malformed `args` entry that is not actually reflected by the proxy instance deployed.

Example:

packages/strategies/src/integrations/curve/ConvexSidecarFactory.sol

SOL

```sol
44  function _isValidGauge(address gauge, bytes memory args) internal view override {
45      uint256 pid = abi.decode(args, (uint256));
46
47      // Get the pool info from Convex
48      (,, address curveGauge,,, bool isShutdown) = BOOSTER.poolInfo(pid);
49
50      // Ensure the pool is not shutdown
51      if (isShutdown) revert PoolShutdown();
52
53      // Ensure the gauge matches
```

Example (Cont.):

```sol
54      if (curveGauge != gauge) revert InvalidGauge();
55  }
```

Recommendation:

We advise the code to enforce this limitation, preventing misleading events from being emitted by the `SidecarFactory::create` function.

Alleviation:

The argument length is now properly sanitized in the `ConvexSidecarFactory::_isValidGauge` function, addressing this exhibit in full.

CurveAllocator Manual Review Findings

CAR-01M: Inadequate Documentation of Function

Type	Severity	Location
Logical Fault	Unknown	CurveAllocator.sol:L181-L192

Description:

The referenced function appears to be a legacy utility helper for legacy Stake DAO implementations, however, no documentation exists as to what it is meant to achieve.

Impact:

This exhibit will be replaced by any findings that are identified after the expected behaviour of the function has been outlined.

Example:

packages/strategies/src/integrations/curve/CurveAllocator.sol

SOL

```sol
180 /// @notice Computes the optimal Stake DAO Locker balance (legacy helper).
181 function getOptimalLockerBalance(address gauge) public view returns (uint256
balanceOfLocker) {
182     // 1. Current veBoost weights
183     uint256 veBoostOfLocker =
IBalanceProvider(BOOST_DELEGATION_V3).balanceOf(LOCKER);
184     uint256 veBoostOfConvex =
IBalanceProvider(BOOST_DELEGATION_V3).balanceOf(CONVEX_BOOST_HOLDER);
185
186     // 2. Current Convex LP balance on the gauge
187     uint256 balanceOfConvex =
IBalanceProvider(gauge).balanceOf(CONVEX_BOOST_HOLDER);
188     if (balanceOfConvex == 0 || veBoostOfConvex == 0) return 0;
189
```

Example (Cont.):

```sol
    // 3. Compute the optimal balance for Stake DAO
    balanceOfLocker = balanceOfConvex.mulDiv(veBoostOfLocker, veBoostOfConvex);
}
```

Recommendation:

We advise this function's documentation or the `SPEC.md` file of the project to be expanded to include this implementation.

Alleviation:

The Stake DAO team opted to omit the function implementation altogether, rendering this exhibit indirectly alleviated.

CurveFactory Manual Review Findings

CFY-01M: Potentially Unhandled Decoding Error

Type	Severity	Location
Logical Fault	Minor	CurveFactory.sol:L93

Description:

The referenced `try-catch` clause is meant to evaluate whether a gauge has been killed, however, it might result in an unhandled `revert` if a gauge implements an empty `fallback` function but does not support the `ILiquidityGauge::is_killed` function signature.

In such a case, the code would return with insufficient bytes causing the `bool` decoding to `revert` in an unhandled way.

Impact:

A non-standard liquidity gauge with an empty fallback function will not be able to be added via the `CurveFactory::createVault` function despite being a potentially valid gauge.

Example:

packages/strategies/src/integrations/curve/CurveFactory.sol

SOL

```sol
91  /// Check if the gauge is not killed.
92  /// Not all the pools, but most of them, have this function.
93  try ILiquidityGauge(_gauge).is_killed() returns (bool isKilled) {
94      if (isKilled) return false;
95  } catch {}
```

Recommendation:

To maximize support for gauges, we advise the code to perform a low-level `staticcall` operation and to decode the yielded payload solely if the call was successful and the yielded payload is `32` bytes in length.

Alleviation:

The Stake DAO team evaluated this exhibit and opted to acknowledge it as they do not consider the protocol to interact with non-standard liquidity gauges.

CFY-02M: Potentially Weak Restriction of Token

Type	Severity	Location
Logical Fault	Minor	CurveFactory.sol:L73-L78

Description:

The referenced restriction of a token inclusion to the system is insufficient as the gauge type status of a token can change (i.e. a token might be added as a gauge after it has been accepted into the Stake DAO system).

Impact:

As the gauge state of a particular token can change from inexistent to existing, the Stake DAO team should either acknowledge this risk in in-line documentation or build logic to accommodate for such a scenario.

Example:

packages/strategies/src/integrations/curve/CurveFactory.sol

SOL

```sol
66  function _isValidToken(address _token) internal view virtual override returns
(bool) {
67      /// If the token is not valid, return false.
68      if (!super._isValidToken(_token)) return false;
69
70      /// We already add CVX to the vault by default.
71      if (_token == CVX) return false;
72
73      /// If the token is available as an inflation receiver, it's not valid.
74      try GAUGE_CONTROLLER.gauge_types(_token) {
75          return false;
```

Example (Cont.):

```sol
    } catch {
        return true;
    }
}
```

Recommendation:

We advise this trait to be assessed and potentially counteracted if considered imperative by logic within the `RewardVault` implementation itself.

Alleviation:

The Stake DAO team evaluated this exhibit and consider the edge case outlined to be of no impact to the operation of their protocol as the restriction is a nice-to-have feature rather than a mandatory security trait.

CFY-03M: Incorrect Strategy Integration

Type	Severity	Location
Logical Fault	Major	CurveFactory.sol:L107

Description:

The contract deployed at the `OLD_STRATEGY` address does not support the `IStrategy::isShutdown` function selector rendering the present integration to be invalid.

Impact:

The **`CurveFactory::_isValidDeployment`** integration is presently incorrect and will fail to permit the re-deployment of gauges present in the `OLD_STRATEGY`.

Example:

packages/strategies/src/integrations/curve/CurveFactory.sol

SOL

```sol
103 function _isValidDeployment(address _gauge) internal view virtual override
returns (bool) {
104     /// We check if the gauge is deployed in the old strategy by checking if the
reward distributor is not 0.
105     /// We also check if the gauge is shutdown.
106     return IStrategy(OLD_STRATEGY).rewardDistributors(_gauge) == address(0)
107         || IStrategy(OLD_STRATEGY).isShutdown(_gauge);
108 }
```

Recommendation:

We advise the code to be revised, potentially requiring manual deployment of gauges that were previously present in the `OLD_STRATEGY` by the factory creator.

Alleviation:

The Stake DAO team evaluated this exhibit and expressed that it is of no concern as an upgrade is underway for the contracts to satisfy the relevant interface.

As such, we consider this exhibit alleviated in the sense that the Stake DAO team intends to address the outlined discrepancy in the near future.

CurveStrategy Manual Review Findings

CSY-01M: Incorrect Checkpoint of Liquidity Gauge Rewards

Type	Severity	Location
Logical Fault	Medium	CurveStrategy.sol:L67-L68

Description:

The referenced code will utilize the dynamically evaluated `ILiquidityGauge::integrate_fraction` and `IMinter::minted` values without actually updating those data entries, causing ensuing `CurveStrategy::_checkpointRewards` function invocations to rely on outdated storage data points and thus re-count already-checkpointed rewards.

Impact:

Any rewards that have not been captured in the `ILiquidityGauge` instance will be re-counted every time rewards are checkpointed via the `CurveStrategy::_checkpointRewards` function incorrectly.

Example:

packages/strategies/src/integrations/curve/CurveStrategy.sol

SOL

```sol
65  if (target == LOCKER) {
66      // Calculate pending rewards for the locker by comparing  total earned by
gauge with already minted tokens
67      pendingRewardsAmount =
68          ILiquidityGauge(gauge).integrate_fraction(LOCKER) -
IMinter(MINTER).minted(LOCKER, gauge);
69
70      pendingRewards.feeSubjectAmount += pendingRewardsAmount.toUint128();
71  } else {
```

Recommendation:

We advise rewards to be activated at the `ILiquidityGauge` instance for example by invoking the `ILiquidityGauge::user_checkpoint` function that is available in all liquidity gauge versions.

Alleviation:

The code was updated to properly checkpoint a user's state, ensuring that the storage variables read by the `CurveStrategy` contract are consistently updated after being evaluated.

CSY-02M: Potentially Incorrect Curve Integration

Type	Severity	Location
Logical Fault	Medium	CurveStrategy.sol:L114

Description:

The referenced function (`IMinter::mint_for`) requires approval by the `LOCKER` toward the caller (i.e. `CurveStrategy`) which does not appear implemented in the codebase as an initialization mechanism.

Impact:

The current reward minting mechanism for the L1 scenario (i.e. `LOCKER != GATEWAY`) appears to be incorrect unless several off-chain steps are taken (i.e. approval of the strategy by the `LOCKER` itself).

Example:

packages/strategies/src/integrations/curve/CurveStrategy.sol

SOL

```sol
103 /// @notice Harvests rewards from a Curve gauge
104 /// @param gauge The address of the Curve gauge to harvest from
105 function _harvestLocker(address gauge, bytes memory) internal override returns
(uint256 rewardAmount) {
106     /// 1. Snapshot the balance before minting.
107     uint256 _before = IERC20(REWARD_TOKEN).balanceOf(address(LOCKER));
108
109     /// @dev Locker is deployed on mainnet.
110     /// @dev If the locker is the gateway, we need to mint the rewards via the
gateway
111     /// as it means the strategy is deployed on sidechain.
112     if (LOCKER != GATEWAY) {
```

Example (Cont.):

```sol
        /// 2. Mint the rewards of the gauge to the locker.
        IMinter(MINTER).mint_for(gauge, address(LOCKER));
    } else {
        /// 2. Mint the rewards of the gauge to the locker via the gateway.
        bytes memory data = abi.encodeWithSignature("mint(address)", gauge);
        require(_executeTransaction(MINTER, data), MintFailed());
    }

    /// 3. Calculate the reward amount.
    rewardAmount = IERC20(REWARD_TOKEN).balanceOf(address(LOCKER)) - _before;
}
```

Recommendation:

We advise this approach to be revisited as the `LOCKER != GATEWAY` scenario would still permit the `LOCKER` itself to mint rewards toward itself if the dedicated branching logic was absent.

Alleviation:

The Stake DAO team evaluated this exhibit and clarified that they intend to issue the relevant approvals via off-chain scripts.

As these scripts were already planned, we consider this exhibit to be nullified.

RewardReceiver Manual Review Findings

RRR-01M: Deprecated Approval Operation

Type	Severity	Location
Standard Conformity	Informational	RewardReceiver.sol:L98

Description:

The referenced approval operation has been officially deprecated by OpenZeppelin in favour of `SafeERC20::forceApprove`.

Impact:

Although the `RewardVault::depositRewards` function implementation will presently consume the full allowance provided, we still advise the codebase to utilize up-to-date approval operations as a matter of standardization.

Example:

packages/strategies/src/RewardReceiver.sol

SOL

```sol
function _depositRewards(IERC20 token, uint128 amount) internal {
    /// Check if there's a distribution period in progress.
    if (rewardVault().getPeriodFinish(address(token)) > block.timestamp) return;
    /// Check if the reward receiver is a valid rewards distributor for the reward token.
    if (rewardVault().getRewardsDistributor(address(token)) != address(this)) return;

    /// Approve the reward vault to spend the reward token.
    token.safeIncreaseAllowance(address(rewardVault()), amount);

    // Deposit rewards to the vault
```

Example (Cont.):

```sol
    rewardVault().depositRewards(address(token), amount);
}
```

Recommendation:

We advise the forced approval mechanism to be utilized instead, future-proofing the codebase against reward vaults that might not consume the full rewards provided as input.

Alleviation:

The code was updated to issue a `SafeERC20::forceApprove` operation instead per our recommendation.

RewardVault Manual Review Findings

RVT-01M: Potentially Incorrect Token Name

Type	Severity	Location
Standard Conformity	Informational	RewardVault.sol:L932

Description:

The referenced token name concatenation will not introduce a space character (`" "`) between `Fusion` and the asset's name, creating potentially incorrect token names.

Example:

packages/strategies/src/RewardVault.sol

SOL

```sol
931 function name() public view override(ERC20, IERC20Metadata) returns (string memory) {
932     return string.concat("StakeDAO Fusion", IERC20Metadata(asset()).name(), " Vault");
933 }
```

Recommendation:

We advise the space character to be introduced right after the `Fusion` keyword, optimizing the legibility of generated token names.

Alleviation:

The token's naming convention was updated per our recommendation, optimizing its legibility.

RVT-02M: Incorrect Usage of Mathematical Library

Type	Severity	Location
Mathematical Operations	Minor	RewardVault.sol:L590

Description:

The referenced usage of `Math::mulDiv` is incorrect as it will calculate the multiplication using bounded arithmetic and will perform a multiplication by `1` using unbounded arithmetic which is effectively a no-op.

Impact:

The arithmetic range the referenced calculation is valid for is unnecessarily constrained.

Example:

packages/strategies/src/RewardVault.sol

SOL

```
590 rewardRatePerToken = Math.mulDiv(timeDelta * reward.rewardRate, 1, _totalSupply);
```

Recommendation:

We advise the code to substitute the `1` value literal with `reward.rewardRate`, removing it from the raw multiplication and thus optimizing the arithmetic range of the referenced calculation.

Alleviation:

The library's usage was corrected, ensuring maximum accuracy is obtained for all operations performed.

RVT-03M: Potentially Incorrect Integration of Reward Evaluation

Type	Severity	Location
Logical Fault	Minor	RewardVault.sol:L886, L887

Description:

The `RewardVault::_update` function will issue a zero-value withdrawal on the `Strategy` implementation, however, it will do so with a receiver equal to the strategy itself.

This will result in the `Strategy::withdraw` function logic to actually process zero-value withdrawals with which certain tokens might `revert`.

Impact:

Certain tokens that revert on zero-value transfers are incompatible with the Stake DAO reward vaults due to a requirement for zero-value withdrawals to be processed for rewards to be recorded.

Example:

packages/strategies/src/RewardVault.sol

SOL

```
885 /// Withdraw 0, just to get the pending rewards.
886 /// @dev We pass the strategy as the receiver to avoid the zero address check on
some tokens.
887 IStrategy.PendingRewards memory pendingRewards = strategy().withdraw(allocation,
POLICY, address(strategy()));
```

Recommendation:

We advise the code to avoid any withdrawals if zero funds have been specified or to invoke dedicated reward-related functions f.e. in sidecars to ensure all rewards are captured without any zero-value transfers.

Alleviation:

The code of the `Strategy::withdraw` implementation was updated to perform a withdrawal solely when non-zero amounts have been specified, ensuring that the operation properly harvests rewards and does not result in zero-value transfers that may fail.

RVT-04M: Incorrect Maximum Limitation Evaluations

Type	Severity	Location
Logical Fault	Medium	**RewardVault.sol**: • I-1: **L704**, **L705** • I-2: **L713**, **L714**

Description:

The referenced maximum limit evaluations for deposit / mint operations are incorrect and deviate from the **EIP-4626** standard as they measure the balance of the `receiver` of deposit / mint operations which may not be the party actually supplying the funds to the contract for the deposit / mint operation to take place.

Impact:

The current implementation deviates from the **EIP-4626** standard as it will yield incorrect limitations for all users who do not directly deposit to themselves, potentially breaking **EIP-4626** DeFi integrations.

Example:

packages/strategies/src/RewardVault.sol

SOL

```sol
701 /// @notice Returns the maximum amount of assets that can be deposited.
702 /// @dev The parameter is not used and is included to satisfy the interface. Pass
whatever you want to.
703 /// @return _ The maximum amount of assets that can be deposited.
704 function maxDeposit(address _account) public view returns (uint256) {
705     return IERC20(asset()).balanceOf(_account);
706 }
707
708 /// @notice Returns the maximum amount of shares that can be minted.
709 /// @dev Due to the 1:1 relationship between assets and shares, the max mint
710 ///      is the same as the max deposit.
```

Example (Cont.):

```sol
711 /// @param _account The address of the account to calculate the max mint for.
712 /// @return _ The maximum amount of shares that can be minted.
713 function maxMint(address _account) external view returns (uint256) {
714     return maxDeposit(_account);
715 }
```

Recommendation:

We advise both statements to be revised, effectively yielding a value of `type(uint128).max` as the real limitation of the system is its integration with the `Accountant` implementation.

Alleviation:

The referenced maximum mint deposit and mint limitations have been corrected in compliance with the **EIP-4626** standard, alleviating this exhibit.

RVT-05M: Insecure Casting Operations

Type	Severity	Location
Logical Fault	Medium	**RewardVault.sol**: • I-1: **L890** • I-2: **L910** • I-3: **L925**

Description:

The referenced casting operations are performed insecurely resulting in data loss.

While the `RewardVault::_mint` and `RewardVault::_burn` casting instances are unlikely to manifest as vulnerabilities, the `RewardVault::_update` function can be exploited to cause abnormally large `Transfer` amounts to be performed that effectively result in no balance transfer.

Impact:

An **EIP-20** transfer of `type(uint128).max + 1` will result in `1` wei being transferred yet an event signaling a `type(uint128).max + 1` fund transfer to be emitted.

Example:

packages/strategies/src/RewardVault.sol

SOL

```sol
869 function _update(address from, address to, uint256 amount) internal override {
870     // 1. Update Reward State
871     _checkpoint(from, to);
872
873     /// Get addresses where funds are allocated to.
874     address[] memory targets = allocator().getAllocationTargets(gauge());
875
876     /// Create an allocation struct to pass to the strategy.
877     /// We want to withdraw 0, just to get the pending rewards.
878     IAllocator.Allocation memory allocation = IAllocator.Allocation({
```

Example (Cont.):

```sol
        asset: asset(),
        gauge: gauge(),
        targets: targets,
        amounts: new uint256[](targets.length)
    });

    /// Withdraw 0, just to get the pending rewards.
    /// @dev We pass the strategy as the receiver to avoid the zero address check on some tokens.
    IStrategy.PendingRewards memory pendingRewards = strategy().withdraw(allocation, POLICY, address(strategy()));

    // 2. Update Balances via Accountant
    ACCOUNTANT.checkpoint(gauge(), from, to, uint128(amount), pendingRewards, POLICY);

    // 3. Emit Transfer event
    emit Transfer(from, to, amount);
}
```

Recommendation:

We advise all casting operations to be securely performed, for example by utilizing OpenZeppelin's `SafeCast` library.

Alleviation:

All three referenced casting operations are now performed safely via the `SafeCast` library, alleviating this exhibit.

RVT-06M: Incorrect Condition Enforcement

Type	Severity	Location
Logical Fault	Major	RewardVault.sol:L379-L380

Description:

Despite what the documentation details, funds will sit within the `RewardVault` solely when the relevant gauge **is not shutdown**, meaning that withdrawals under normal operating conditions will not be fulfilled as expected.

Impact:

The `RewardVault` will fail to fulfill withdrawals under normal operating conditions despite what its documentation details.

Example:

packages/strategies/src/RewardVault.sol

SOL

```sol
379 /// @dev If the gauge is shutdown, funds will sit here.
380 if (PROTOCOL_CONTROLLER.isShutdown(gauge())) {
381     // Transfer the assets to the receiver. The 1:1 relationship between assets and shares is maintained.
382     SafeERC20.safeTransfer(IERC20(asset()), receiver, shares);
383 }
```

Recommendation:

We advise the referenced condition to be negated, ensuring that its behaviour matches its documentation.

Alleviation:

The Stake DAO team evaluated this exhibit and clarified that this is intended behaviour as funds will remain within the `RewardVault` if the strategy has been shut down.

As such, we consider this exhibit to be invalid and thus nullified.

RVT-07M: Inexistent Validation of Authorization

Type	Severity	Location
Logical Fault	Major	RewardVault.sol:L257, L310, L320

Description:

The `RewardVault::deposit(address,uint256,address)` function signature permits a deposit to occur with a different provider of funds, claiming that the function should restrict callers to allowed addresses.

This restriction is not imposed, permitting any user's approval toward the `RewardVault` to be consumed maliciously. Furthermore, the `Deposit` **EIP-4626** event emitted within the `RewardVault::_deposit` function is incorrect as a result breaching EIP compliancy.

Impact:

Any user can deposit on behalf of another with a different benefactor, effectively permitting approved funds toward a `RewardVault` to be exploited at will.

Example:

packages/strategies/src/RewardVault.sol

SOL

```
296 function _deposit(address account, address receiver, uint256 assets, uint256
shares, address referrer) internal {
297     // Update the reward state for the receiver
298     _checkpoint(receiver, address(0));
299
300     // Get the address of the allocator contract from the protocol controller
301     // then fetch the recommended deposit allocation from the allocator
302     IAllocator.Allocation memory allocation =
allocator().getDepositAllocation(asset(), gauge(), assets);
303
304     // Get the address of the asset from the clone's immutable args then for each
target recommended by
305     // the allocator, transfer the amount from the account to the target
```

Example (Cont.):

```sol
306     IERC20 _asset = IERC20(asset());
307     for (uint256 i; i < allocation.targets.length; i++) {
308         if (allocation.amounts[i] == 0) continue;
309         require(PROTOCOL_CONTROLLER.isValidAllocationTarget(gauge(),
allocation.targets[i]), TargetNotApproved());
310         SafeERC20.safeTransferFrom(_asset, account, allocation.targets[i],
allocation.amounts[i]);
311     }
312
313     // Get the address of the strategy contract from the protocol controller
314     // then process the deposit of the allocation
315     IStrategy.PendingRewards memory pendingRewards =
strategy().deposit(allocation, POLICY);
316
317     // Mint the shares to the receiver
318     _mint(receiver, shares, pendingRewards, POLICY, referrer);
319
320     emit Deposit(msg.sender, receiver, assets, shares);
321 }
```

Recommendation:

We advise the `RewardVault::_deposit` function implementation to properly validate authorization between the `msg.sender` and the `account` whenever they do not match, either via a signature payload or via an on-chain authorization.

Alleviation:

The `RewardVault::deposit` function was updated to include the `RewardVault::onlyAllowed` function modifier, ensuring that it is invoked by specific contract implementations which are meant to validate allowances and other such relations adequately.

Router Manual Review Findings

RRE-01M: Inconsistent Module Name Resolution Behaviour

Type	Severity	Location
Logical Fault	Minor	Router.sol:L111-L114

Description:

The `Router::getModuleName` function is meant to query the name of a particular module, however, it is impossible to distinguish between a module with an empty name and an inexistent module.

Impact:

The behaviour of an empty name being yielded by the `Router::getModuleName` function is undefined as it may indicate an inexistent module or a module with an empty name.

Example:

packages/strategies/src/Router.sol

SOL

```sol
108 /// @notice Gets the name of the module at the given identifier
109 /// @param identifier The unique identifier of the module
110 /// @return name The name of the module. Returns an empty string if the module is
not set
111 function getModuleName(uint8 identifier) public view returns (string memory name)
{
112     address module = getModule(identifier);
113     if (module != address(0)) name = IRouterModule(module).name();
114 }
```

Recommendation:

We advise the code to `revert` if a module is not found or the registration mechanism of `Router::setModule` to ensure a non-zero `IRouterModule::name` has been defined, either of which we consider an adequate resolution of this exhibit.

Alleviation:

The `Router::setModule` function was updated to ensure that a non-zero name exists for the module being configured, alleviating this exhibit as advised.

RRE-02M: Incorrect Module Enumeration

Type	Severity	Location
Logical Fault	Minor	Router.sol:L125-L133

Description:

The `Router::enumerateModules` function expects modules to have been sequentially introduced to the `Router` implementation, however, the `Router::setModule` function does not enforce this prerequisite.

As such, any gap in the module identifiers will result in an early termination of module enumeration.

Impact:

Module enumeration will fail to function as expected if any module is introduced at an identifier that is not sequential.

Example:

packages/strategies/src/Router.sol

SOL

```
116 /// @notice Convenient function to enumerate the incrementally stored modules
117 /// @dev Never call this function on-chain. It is only meant to be used off-chain
for informational purposes.
118 ///      This function should not replace off-chain indexing of the modules.
119 ///
120 ///      This function stops iterating when it encounters address(0). This means
that
121 ///      if the modules are not stored contiguously, this function will return
only a subset of the modules.
122 /// @return modules The concatenated addresses of the modules in a bytes array.
123 ///                 The length of the returned bytes array is `20 * n`, where `n`
is the number of modules.
124 ///                 Returns an empty bytes array if the first slot is not set.
125 function enumerateModules() external view returns (bytes memory modules) {
```

Example (Cont.):

```sol
    for (uint8 i; i < type(uint8).max; i++) {
        address module = getModule(i);

        if (module == address(0)) break;

        modules = abi.encodePacked(modules, module);
    }
}
```

Recommendation:

We advise the `Router::setModule` function to ensure modules are sequentially introduced by validating that the previous identifier's module is non-zero, ensuring enumeration can be safely performed.

Alleviation:

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

RouterModuleClaim Manual Review Findings

RMC-01M: Misleading Function Restriction

Type	Severity	Location
Logical Fault	Informational	RouterModuleClaim.sol:L18-L19

Description:

The referenced restriction meant to be imposed by the `RouterModuleClaim::claim` function is not done so.

Example:

packages/strategies/src/RouterModules/RouterModuleClaim.sol

SOL

```sol
13  /// @notice Claim rewards from the reward vault
14  /// @param rewardVault The address of the reward vault to call
15  /// @param account The address of the account to claim for
16  /// @param tokens The array of tokens to claim
17  /// @param receiver The address to receive the rewards
18  /// @custom:throws OnlyAllowed if this function is not called using delegatecall
19  ///                from an account authorized by the protocol controller
20  function claim(address rewardVault, address account, address[] calldata tokens,
address receiver)
21      external
22      returns (uint256[] memory amounts)
```

Example (Cont.):

```sol
{
    amounts = IRewardVault(rewardVault).claim(account, tokens, receiver);
}
```

Recommendation:

We advise either the logic or the documentation to be corrected, either of which we consider an adequate resolution to this exhibit.

Alleviation:

The referenced restriction has been omitted from the function's documentation, addressing this exhibit.

RMC-02M: Inexistent Validation of Authorization

Type	Severity	Location
Logical Fault	Medium	RouterModuleClaim.sol:L20, L24

Description:

The `RouterModuleClaim::claim` function will not validate that the `msg.sender` has been authorized to claim on behalf of the `account`, permitting any user's rewards to be claimed maliciously.

Impact:

All rewards a user accumulates in the `RewardVault` instance can be siphoned by a malicious user through the `Router` implementation without any approval as the `Router` is authorized to invoke the `RewardVault::claim(address,address[],address)` function signature.

Example:

packages/strategies/src/RouterModules/RouterModuleClaim.sol

SOL

```sol
20  function claim(address rewardVault, address account, address[] calldata tokens,
address receiver)
21      external
22      returns (uint256[] memory amounts)
23  {
24      amounts = IRewardVault(rewardVault).claim(account, tokens, receiver);
25  }
```

Recommendation:

We advise the code to enforce some sort of approval between the caller of the function and the `account`, preventing unauthorized access to a user's rewards.

Alleviation:

The referenced function no longer accepts an `account` argument, utilizing the `msg.sender` in its place and thus requiring no further authorization validation.

RouterModuleDeposit Manual Review Findings

RMD-01M: Misleading Function Restrictions

Type	Severity	Location
Logical Fault	Informational	**RouterModuleDeposit.sol**: • I-1: **L17-L18** • I-2: **L31-L32**

Description:

The referenced restrictions meant to be imposed by both functions of the `RouterModuleDeposit` contract are not done so.

Example:

packages/strategies/src/RouterModules/RouterModuleDeposit.sol

SOL

```sol
13  /// @notice Deposit assets into the reward vault
14  /// @param rewardVault The address of the reward vault to call
15  /// @param account The address of the account to deposit for
16  /// @param assets The amount of assets to deposit
17  /// @custom:throws OnlyAllowed if this function is not called using delegatecall
18  ///                from an account authorized by the protocol controller
19  /// @custom:throws ZeroAddress if the account is the zero address
20  /// @custom:throws SafeERC20FailedOperation if the account does not have enough
allowance
21  ///                for the reward vault to transfer the assets
22  function deposit(address rewardVault, address account, uint256 assets) external
returns (uint256) {
```

Example (Cont.):

```sol
    return IRewardVault(rewardVault).deposit(account, assets);
}

/// @notice Deposit assets into the reward vault
/// @param rewardVault The address of the reward vault to call
/// @param account The address of the account to deposit for
/// @param assets The amount of assets to deposit
/// @param referrer The address of the referrer
/// @custom:throws OnlyAllowed if this function is not called using delegatecall
///                from an account authorized by the protocol controller
/// @custom:throws ZeroAddress if the account is the zero address
/// @custom:throws SafeERC20FailedOperation if the account does not have enough allowance
///                for the reward vault to transfer the assets
function deposit(address rewardVault, address account, uint256 assets, address referrer)
    public
    payable
    returns (uint256)
{
    return IRewardVault(rewardVault).deposit(account, assets, referrer);
}
```

Recommendation:

We advise either the logic or the documentation to be corrected, either of which we consider an adequate resolution to this exhibit.

Alleviation:

The referenced restrictions have been omitted from each function's documentation, addressing this exhibit.

RMD-02M: Incorrect Payable Attribute

Type	Severity	Location
Logical Fault	Minor	RouterModuleDeposit.sol:L38

Description:

The `RouterModuleDeposit::deposit(address,address,uint256,address)` variant has been incorrectly specified as `payable`.

Impact:

It is presently possible for native funds to be locked within the contract if they are supplied alongside a `RouterModuleDeposit::deposit(address,address,uint256,address)` call.

Example:

packages/strategies/src/RouterModules/RouterModuleDeposit.sol

SOL

```
26  /// @notice Deposit assets into the reward vault
27  /// @param rewardVault The address of the reward vault to call
28  /// @param account The address of the account to deposit for
29  /// @param assets The amount of assets to deposit
30  /// @param referrer The address of the referrer
31  /// @custom:throws OnlyAllowed if this function is not called using delegatecall
32  ///                from an account authorized by the protocol controller
33  /// @custom:throws ZeroAddress if the account is the zero address
34  /// @custom:throws SafeERC20FailedOperation if the account does not have enough
allowance
35  ///                for the reward vault to transfer the assets
```

Example (Cont.):

```sol
function deposit(address rewardVault, address account, uint256 assets, address referrer)
    public
    payable
    returns (uint256)
{
    return IRewardVault(rewardVault).deposit(account, assets, referrer);
}
```

Recommendation:

We advise the `payable` attribute to be omitted as the function does not utilize its native value nor does it forward it to the reward vault.

Alleviation:

The `payable` attribute has been properly removed from the referenced function.

RouterModuleMigrationCurve Manual Review Findings

RMM-01M: Inexistent Validation of Vault Addresses

Type	Severity	Location
Logical Fault	Major	RouterModuleMigrationCurve.sol:L23

Description:

The `RouterModuleMigrationCurve::migrate` function does not validate the `from` and `to` addresses nor any authorization from the `account` address toward the function caller, permitting malicious `from` and `to` members to be supplied so as to extract any approval by an `account` to the `RouterModuleMigrationCurve` implementation (i.e. the `Router` contract).

Impact:

Any funds approved toward the `Router` instance can be siphoned via a malicious migration operation with attacker-supplied `from` and `to` vault implementations.

Example:

packages/strategies/src/RouterModules/RouterModuleMigrationCurve.sol

SOL

```sol
17  /// @notice Migrates shares from a liquidity gauge to a reward vault
18  /// @dev The account must have approved the token to the router contract
19  /// @param from The address of the old vault
20  /// @param to The address of the new reward vault
21  /// @param account The address of the account to migrate
22  /// @param shares The number of shares to migrate
23  function migrate(address from, address to, address account, uint256 shares)
external {
24      address asset = RewardVault(to).asset();
25      require(ILiquidityGauge(from).lp_token() == asset, VaultNotCompatible());
26
```

Example (Cont.):

```sol
    // 1. Transfer the token of the user to the router contract
    ILiquidityGauge(from).safeTransferFrom(account, address(this), shares);

    // 2. Withdraw the tokens in the old vault
    ILiquidityGauge(from).withdraw(shares);

    // 3. Deposit the tokens in the reward vault
    ILiquidityGauge(asset).safeIncreaseAllowance(to, shares);
    RewardVault(to).deposit(shares, account);
}
```

Recommendation:

We advise the `from` and `to` addresses to be validated (f.e. via a factory) and authorization to be enforced between the `msg.sender` and the `acount` whose funds are being migrated.

Alleviation:

The code was updated to no longer accept an arbitrary `account`, rendering supply of malicious `from` and `to` vault implementations to solely impact the caller and thus be of no consequence.

RouterModuleMigrationStakeDAOV1 Manual Review Findings

RMS-01M: Inexistent Validation of Vault Addresses

Type	Severity	Location
Logical Fault	Major	RouterModuleMigrationStakeDAOV1.sol:L29

Description:

The `RouterModuleMigrationStakeDAOV1::migrate` function does not validate the `from` and `to` addresses nor any authorization from the `account` address toward the function caller, permitting malicious `from` and `to` members to be supplied so as to extract any approval by an `account` to the `RouterModuleMigrationStakeDAOV1` implementation (i.e. the `Router` contract).

Impact:

Any funds approved toward the `Router` instance can be siphoned via a malicious migration operation with attacker-supplied `from` and `to` vault implementations.

Example:

packages/strategies/src/RouterModules/RouterModuleMigrationStakeDAOV1.sol

SOL

```sol
23  /// @notice Migrates shares from a liquidity gauge to a reward vault
24  /// @dev The account must have approved the token to the router contract
25  /// @param from The address of the old vault
26  /// @param to The address of the new reward vault
27  /// @param account The address of the account to migrate
28  /// @param shares The number of shares to migrate
29  function migrate(address from, address to, address account, uint256 shares)
external {
30      address asset = RewardVault(to).asset();
31      require(IVault(from).token() == asset, VaultNotCompatible());
32
```

Example (Cont.):

```sol
        // 1. Transfer user's gauge token to the router contract
        IERC20(IVault(from).liquidityGauge()).safeTransferFrom(account,
address(this), shares);

        // 2. Withdraw the shares from the old vault
        IVault(from).withdraw(shares);

        // 3. Deposit the shares in the new reward vault
        IERC20(asset).safeIncreaseAllowance(to, shares);
        RewardVault(to).deposit(shares, account);
    }
```

Recommendation:

We advise the `from` and `to` addresses to be validated (f.e. via a factory) and authorization to be enforced between the `msg.sender` and the `acount` whose funds are being migrated.

Alleviation:

The code was updated to no longer accept an arbitrary `account`, rendering supply of malicious `from` and `to` vault implementations to solely impact the caller and thus be of no consequence.

RouterModuleMigrationYearn Manual Review Findings

RMY-01M: Inexistent Validation of Vault Addresses

Type	Severity	Location
Logical Fault	Major	RouterModuleMigrationYearn.sol:L23

Description:

The `RouterModuleMigrationYearn::migrate` function does not validate the `from` and `to` addresses nor any authorization from the `account` address toward the function caller, permitting malicious `from` and `to` members to be supplied so as to extract any approval by an `account` to the `RouterModuleMigrationYearn` implementation (i.e. the `Router` contract).

Impact:

Any funds approved toward the `Router` instance can be siphoned via a malicious migration operation with attacker-supplied `from` and `to` vault implementations.

Example:

packages/strategies/src/RouterModules/RouterModuleMigrationYearn.sol

SOL

```sol
17  /// @notice Migrates shares from convex to a reward vault
18  /// @dev The account must have approved the token to the router contract
19  /// @param from The address of the old vault
20  /// @param to The address of the new reward vault
21  /// @param account The address of the account to migrate
22  /// @param shares The number of shares to migrate
23  function migrate(address from, address to, address account, uint256 shares)
external {
24      address asset = RewardVault(to).asset();
25      require(IYearnVault(from).token() == asset, VaultNotCompatible());
26
```

Example (Cont.):

```sol
    // 1. Transfer the token of the user to the router contract
    IYearnVault(from).safeTransferFrom(account, address(this), shares);

    // 2. Withdraw the tokens in the old vault
    IYearnVault(from).withdraw(shares);

    // 3. Deposit the tokens in the reward vault
    IYearnVault(asset).safeIncreaseAllowance(to, shares);
    RewardVault(to).deposit(shares, account);
}
```

Recommendation:

We advise the `from` and `to` addresses to be validated (f.e. via a factory) and authorization to be enforced between the `msg.sender` and the `acount` whose funds are being migrated.

Alleviation:

The code was updated to no longer accept an arbitrary `account`, rendering supply of malicious `from` and `to` vault implementations to solely impact the caller and thus be of no consequence.

RouterModuleWithdraw Manual Review Findings

RMW-01M: Insecure Execution of Withdrawals

Type	Severity	Location
Logical Fault	Major	RouterModuleWithdraw.sol:L25

Description:

The `RouterModuleWithdraw::withdraw` function will not validate any authorization between the `msg.sender` and the `owner` of the **EIP-4626** shares, permitting any approval of the `owner` toward the `RouterModuleWithdraw` contract (i.e. the `Router` implementation) to be consumed at will.

Impact:

Any withdrawal operation meant to be performed via the `Router::execute` function can be hijacked to siphon all approved funds as there is no authorization enforced between the `owner` of shares and the executor of the withdrawal.

Example:

packages/strategies/src/RouterModules/RouterModuleWithdraw.sol

SOL

```sol
13  /// @notice Withdraws `assets` from the vault to `receiver` by burning shares
from `owner`.
14  /// @dev `owner` must allow the `rewardVault` to spend the `assets`
15  /// @param rewardVault The address of the reward vault to call
16  /// @param assets The amount of assets to withdraw.
17  /// @param receiver The address to receive the assets. If the receiver is the
zero address, the assets will be sent to the owner.
18  /// @param owner The address to burn shares from.
19  /// @return _ The amount of assets withdrawn.
20  /// @custom:throws NotApproved if the Router is not allowed to withdraw the
assets
21  function withdraw(address rewardVault, uint256 assets, address receiver, address
owner)
22      external
```

Example (Cont.):

```sol
    returns (uint256)
{
    return IERC4626(rewardVault).withdraw(assets, receiver, owner);
}
```

Recommendation:

We advise the code to validate authorization between the `msg.sender` and `owner`, preventing arbitrary consumption of allowances.

Alleviation:

The code was updated to utilize the `msg.sender` in place of the arbitrarily-specified `owner`, preventing approvals of other users to be consumed and thus alleviating this exhibit.

Accountant Code Style Findings

ATN-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	**Accountant.sol**: • I-1: **L523** • I-2: **L531**

Description:

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

Example:

packages/strategies/src/Accountant.sol

SOL

```
522 if (netAfterReservedFees > netCredited && _vault.supply > 0) {
523     uint256 newRewards = netAfterReservedFees - netCredited;
```

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

ATN-02C: Non-Descriptive Harvest Policies

Type	Severity	Location
Code Style	Informational	Accountant.sol:L281, L296

Description:

The referenced harvest policies are non-descriptive and do not actually match their behaviour.

In detail, a harvest policy of `HARVEST` is meant to harvest on each checkpoint operation (i.e. does not require `Accountant::harvest` to be invoked) whereas a harvest policy of `CHECKPOINT` requires rewards to be manually harvested.

Example:

packages/strategies/src/Accountant.sol

SOL

```
280 uint128 totalFees;
281 if (policy == IStrategy.HarvestPolicy.HARVEST && newRewards > 0) {
282     // Calculate total fees in one operation
283     // We charge only protocol fee on the harvested rewards.
284     if (newFeeSubjectAmount > 0) {
285         totalFees = newFeeSubjectAmount.mulDiv(getProtocolFeePercent(),
1e18).toUint128();
286         // Update protocol fees accrued.
287         protocolFeesAccrued += totalFees;
288     }
289
```

Example (Cont.):

```sol
    // Update integral with new rewards per token
    integral += (newRewards - totalFees).mulDiv(SCALING_FACTOR, supply);
}
// If the new rewards are above the minimum meaningful rewards,
// we update the integral and pending rewards.
// Otherwise, we don't update the integral to avoid precision loss. It won't be lost, just delayed.
else if (policy == IStrategy.HarvestPolicy.CHECKPOINT && newRewards >= MIN_MEANINGFUL_REWARDS) {
```

Recommendation:

We advise the policy types to be aptly renamed, for example by renaming `HARVEST` to `ON_CHECKPOINT` and `CHECKPOINT` to `ON_HARVEST`.

Alleviation:

The Stake DAO team has opted to retain the current naming convention as they consider it aligned with their interpretation of the codebase.

ATN-03C: Non-Standard Evaluation of New Rewards

Type	Severity	Location
Code Style	Informational	Accountant.sol:L526

Description:

The referenced `newRewards` calculation is effectively the equivalent of the `Accountant::checkpoint` function calculation albeit with different operands.

The `netAfterReservedFees - netCredited` calculation can be seen as `pendingRewards.totalAmount` minus `_vault.netCredited + protocolFee + harvesterFee`; a value that is equivalent to `_vault.totalAmount`.

Example:

packages/strategies/src/Accountant.sol

SOL

```
516 uint256 netAfterReservedFees = pendingRewards.totalAmount - protocolFee -
harvesterFee;
517 uint256 netCredited = _vault.netCredited;
518
519 // Strategy must at least make LPs whole
520 require(netAfterReservedFees >= netCredited, NetCreditsNotEnough());
521
522 if (netAfterReservedFees > netCredited && _vault.supply > 0) {
523     uint256 newRewards = netAfterReservedFees - netCredited;
```

Recommendation:

We advise the `_vault.totalAmount` value to be utilized for these calculations, standardizing the mathematical formulae implemented in the system.

Alleviation:

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

ATN-04C: Redundant Fee Restriction

Type	Severity	Location
Gas Optimization	Informational	Accountant.sol:L730

Description:

The referenced `require` check is redundant as the same condition is indirectly evaluated by the ensuing `require` statement regarding the `totalFee` of the system.

Example:

packages/strategies/src/Accountant.sol

SOL

```sol
725 /// @notice Updates the protocol fee percentage.
726 /// @param newProtocolFeePercent New protocol fee percentage (scaled by 1e18).
727 /// @custom:throws FeeExceedsMaximum If fee would exceed maximum.
728 function setProtocolFeePercent(uint128 newProtocolFeePercent) external onlyOwner
{
729     // check that the provided protocol fee is valid
730     require(newProtocolFeePercent <= MAX_FEE_PERCENT, FeeExceedsMaximum());
731
732     FeesParams storage currentFees = feesParams;
733
734     // check that the total fee (protocol + harvest) is valid
```

Example (Cont.):

```sol
    uint128 totalFee = newProtocolFeePercent + currentFees.harvestFeePercent;
    require(totalFee <= MAX_FEE_PERCENT, FeeExceedsMaximum());

    // emit the update event
    emit ProtocolFeePercentSet(currentFees.protocolFeePercent,
newProtocolFeePercent);

    // set the new protocol fee percent
    currentFees.protocolFeePercent = newProtocolFeePercent;
}
```

Recommendation:

We advise `newProtocolFeePercent` restriction to be omitted, optimizing the code's gas cost.

Alleviation:

The referenced fee restriction has been omitted, optimizing the code's gas cost.

ATN-05C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	Accountant.sol:L285, L300, L306, L526, L532

Description:

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

Example:

packages/strategies/src/Accountant.sol

SOL

```
285 totalFees = newFeeSubjectAmount.mulDiv(getProtocolFeePercent(), 1e18).toUint128();
```

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

Allocator Code Style Findings

ARO-01C: Redundant Variable

Type	Severity	Location
Gas Optimization	Informational	Allocator.sol:L20, L32

Description:

The `GATEWAY` variable is not referenced in any derivative implementations of the `Allocator` nor on the `Allocator` itself.

Example:

packages/strategies/src/Allocator.sol

SOL

```sol
/// @notice The address of the locker contract
/// @dev This is the contract that holds tokens for staking.
address public immutable LOCKER;

/// @notice The address of the gateway contract
/// @dev This is the contract that handles the actual deposit/withdrawal operations.
address public immutable GATEWAY;

/// @notice Error thrown when the gateway is zero address
error GatewayZeroAddress();
```

Example (Cont.):

```sol
24
25  /// @notice Initializes the Allocator contract
26  /// @param _locker The address of the locker contract (can be address(0) for L2s)
27  /// @param _gateway The address of the gateway contract
28  /// @dev If _locker is address(0), LOCKER will be set to the same address as
GATEWAY
29  constructor(address _locker, address _gateway) {
30      require(_gateway != address(0), GatewayZeroAddress());
31
32      GATEWAY = _gateway;
33      /// In some cases (L2s), the locker is the same as the gateway.
34      LOCKER = _locker == address(0) ? _gateway : _locker;
35  }
```

Recommendation:

We advise it to be omitted, optimizing the code's deployment gas cost.

Alleviation:

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

ConvexSidecar Code Style Findings

CSR-01C: Inefficient Logical Branch

Type	Severity	Location
Gas Optimization	Informational	ConvexSidecar.sol:L136, L139

Description:

The `ConvexSidecar::getRewardTokens` function will inefficiently branch its execution as the `IBaseRewardPool::rewardToken` is evaluated under all execution scenarios.

Example:

packages/strategies/src/integrations/curve/ConvexSidecar.sol

SOL

```sol
132 /// For PIDs greater than 150, the virtual balance pool also has a wrapper.
133 /// So we need to get the token from the wrapper.
134 /// More: https://docs.convexfinance.com/convexfinanceintegration/baserewardpool
135 if (pid() >= 151) {
136     address wrapper = IBaseRewardPool(_token).rewardToken();
137     tokens[i] = IStashTokenWrapper(wrapper).token();
138 } else {
139     tokens[i] = IBaseRewardPool(_token).rewardToken();
140 }
```

Recommendation:

We advise the `tokens[i]` value to be written to the `IBaseRewardPool::rewardToken` value outside the `if-else` branch, the `if` branch to write the `tokens[i].token()` value to itself, and the `else` branch to be omitted.

Alleviation:

The code was optimized in line with its original reference implementation.

CSR-02C: Inefficient Re-Queries of Values

Type	Severity	Location
Gas Optimization	Informational	**ConvexSidecar.sol**: • I-1: **L123**, **L130** • I-2: **L135** • I-3: **L167**, **L174**

Description:

The referenced statements will continuously re-evaluate immutable values of the `ConvexSidecar` bytecode.

Example:

packages/strategies/src/integrations/curve/ConvexSidecar.sol

SOL

```sol
121 function getRewardTokens() public view override returns (address[] memory) {
122     // Check if there is extra rewards
123     uint256 extraRewardsLength = baseRewardPool().extraRewardsLength();
124
125     address[] memory tokens = new address[](extraRewardsLength);
126
127     address _token;
128     for (uint256 i; i < extraRewardsLength;) {
129         /// Get the address of the virtual balance pool.
130         _token = baseRewardPool().extraRewards(i);
```

Example (Cont.):

```sol

        /// For PIDs greater than 150, the virtual balance pool also has a wrapper.
        /// So we need to get the token from the wrapper.
        /// More: https://docs.convexfinance.com/convexfinanceintegration/baserewardpool
        if (pid() >= 151) {
            address wrapper = IBaseRewardPool(_token).rewardToken();
            tokens[i] = IStashTokenWrapper(wrapper).token();
        } else {
            tokens[i] = IBaseRewardPool(_token).rewardToken();
        }

        unchecked {
            ++i;
        }
    }

    return tokens;
}
```

Recommendation:

We advise each function to be invoked once, stored to a local variable, and consequently re-used optimally across all statements.

Alleviation:

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

CSR-03C: Inefficient Usage of Allowance Increase

Type	Severity	Location
Gas Optimization	Informational	ConvexSidecar.sol:L74

Description:

The referenced approval operation is guaranteed to succeed as the code will ensure that the previous approval was `0`.

Example:

packages/strategies/src/integrations/curve/ConvexSidecar.sol

SOL

```sol
71  function _initialize() internal override {
72      require(asset().allowance(address(this), address(BOOSTER)) == 0,
AlreadyInitialized());
73
74      asset().safeIncreaseAllowance(address(BOOSTER), type(uint256).max);
75  }
```

Recommendation:

We advise a direct `IERC20::approve` operation to be performed, optimizing the code's gas cost.

Alleviation:

The code was updated to issue a `SafeERC20::forceApprove` call instead which we consider more optimal than the increase operation and thus a proper remediation to this exhibit.

CurveAllocator Code Style Findings

CAR-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	**CurveAllocator.sol**: • I-1: **L68** • I-2: **L73** • I-3: **L114** • I-4: **L120** • I-5: **L124** • I-6: **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:

packages/strategies/src/integrations/curve/CurveAllocator.sol

SOL

```
68  uint256 toLocker = optimalLocker > balanceOfLocker ? optimalLocker -
balanceOfLocker : 0;
```

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

CAR-02C: Inefficient Function Implementation

Type	Severity	Location
Gas Optimization	Informational	CurveAllocator.sol:L213-L217

Description:

The `CurveAllocator::_getVeBoosts` function is meant to be utilized to maintain the same percentage ratio of vote-escrowed balance for the `LOCKER` implementation in comparison to the Convex system and is presently inefficient as it will yield the vote-escrowed balances which are never utilized as-is and are instead utilized for a proportion calculation (i.e. multiplication and division).

Example:

packages/strategies/src/integrations/curve/CurveAllocator.sol

SOL

```
64  (uint256 veLocker, uint256 veTotal) = _getVeBoosts();
65  uint256 optimalLocker = total.mulDiv(veLocker, veTotal);
```

Recommendation:

We advise the function to accept an input `uint256` argument, converting it to the necessary proportion and yielding it to its caller in an optimal way.

Alleviation:

The code was optimized per our recommendation, reducing its gas cost whilst increasing its legibility.

CAR-03C: Repetitive Value Literals

Type	Severity	Location
Code Style	Informational	CurveAllocator.sol:L72, L73, L108, L109, L135, L136, L166, L

Description:

The linked value literals are repeated across the codebase multiple times.

Example:

packages/strategies/src/integrations/curve/CurveAllocator.sol

SOL

```sol
72  alloc.amounts[1] = toLocker; // to Locker
```

Recommendation:

We advise each to be set to its dedicated `constant` variable instead, optimizing the legibility of the codebase.

In case some of the `constant` declarations have already been introduced, we advise them to be properly re-used across the code.

Alleviation:

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

CurveFactory Code Style Findings

CFY-01C: Inefficient ABI Encoding Mechanisms

Type	Severity	Location
Gas Optimization	Informational	**CurveFactory.sol**: • I-1: **L122** • I-2: **L150** • I-3: **L157**

Description:

The referenced statements will create an ABI encoded transaction utilizing `string` literals rather than function selectors.

Example:

packages/strategies/src/integrations/curve/CurveFactory.sol

SOL

```sol
150 bytes memory data = abi.encodeWithSignature("set_rewards_receiver(address)",
_rewardReceiver);
```

Recommendation:

We advise the referenced function signatures to be introduced to an `interface` that can be consequently imported by the `CurveFactory` contract and utilized via its `selector` syntax (i.e. `ILiquidityGauge::set_rewards_receiver`).

This approach will permit the `abi.encodeWithSelector` built-in function to be utilized, optimizing the code's gas cost.

Alleviation:

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

CFY-02C: Inefficient Loop Iterator

Type	Severity	Location
Gas Optimization	Informational	CurveFactory.sol:L129

Description:

The referenced `for` loop will utilize a `uint8` iterator which is inefficient.

Example:

packages/strategies/src/integrations/curve/CurveFactory.sol

SOL

```
129 for (uint8 i = 0; i < 8; i++) {
```

Recommendation:

We advise the iterator to be updated to the `uint256` data type, optimizing the code's gas cost.

Alleviation:

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

CFY-03C: Redundant Gauge Validity Flag

Type	Severity	Location
Gas Optimization	Informational	CurveFactory.sol:L82, L86, L95, L98

Description:

The referenced variable is redundant as the `CurveFactory::_isValidGauge` function will either yield `false` immediately or `true` at its end.

Example:

packages/strategies/src/integrations/curve/CurveFactory.sol

SOL

```sol
81  function _isValidGauge(address _gauge) internal view virtual override returns
(bool) {
82      bool isValid;
83      /// Check if the gauge is a valid candidate and available as an inflation
receiver.
84      /// This call always reverts if the gauge is not valid.
85      try GAUGE_CONTROLLER.gauge_types(_gauge) {
86          isValid = true;
87      } catch {
88          return false;
89      }
90
```

Example (Cont.):

```sol
    /// Check if the gauge is not killed.
    /// Not all the pools, but most of them, have this function.
    try ILiquidityGauge(_gauge).is_killed() returns (bool isKilled) {
        if (isKilled) return false;
    } catch {}

    /// If the gauge doesn't support the is_killed function, but is unofficially killed, it can be deployed.
    return isValid;
}
```

Recommendation:

We advise the code to simply yield `true` at the end, optimizing the code's gas cost.

Alleviation:

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

CurveStrategy Code Style Findings

CSY-01C: Inefficient ABI Encoding Mechanisms

Type	Severity	Location
Gas Optimization	Informational	**CurveStrategy.sol**: • I-1: **L85** • I-2: **L95** • I-3: **L99** • I-4: **L117**

Description:

The referenced statements will create an ABI encoded transaction utilizing `string` literals rather than function selectors.

Example:

packages/strategies/src/integrations/curve/CurveStrategy.sol

SOL

```
117 bytes memory data = abi.encodeWithSignature("mint(address)", gauge);
```

Recommendation:

We advise the referenced function signatures to be introduced to an `interface` that can be consequently imported by the `CurveStrategy` contract and utilized via its `selector` syntax (i.e. `IMinter::mint`).

This approach will permit the `abi.encodeWithSelector` built-in function to be utilized, optimizing the code's gas cost.

Alleviation:

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

CSY-02C: Redundant Code Repetition

Type	Severity	Location
Code Style	Informational	CurveStrategy.sol:L68

Description:

The referenced calculation can be accessed via the `ILiquidityGauge::claimable_tokens` implementation.

Example:

packages/strategies/src/integrations/curve/CurveStrategy.sol

SOL

```
66  // Calculate pending rewards for the locker by comparing  total earned by gauge with already minted tokens
67  pendingRewardsAmount =
68      ILiquidityGauge(gauge).integrate_fraction(LOCKER) - IMinter(MINTER).minted(LOCKER, gauge);
```

Recommendation:

We advise it to be invoked, optimizing the code's legibility.

Alleviation:

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

Factory Code Style Findings

FYR-01C: Redundant Validation of Protocol Controller

Type	Severity	Location
Gas Optimization	Informational	Factory.sol:L79

Description:

The `_protocolController` argument is validated locally in the `Factory::constructor` as well as in the `ProtocolContext::constructor` implementations redundantly.

Example:

packages/strategies/src/Factory.sol

SOL

```sol
63  /// @notice Initializes the factory with protocol controller, reward token, and
vault implementation
64  /// @param _protocolController Address of the protocol controller
65  /// @param _vaultImplementation Address of the reward vault implementation
66  /// @param _rewardReceiverImplementation Address of the reward receiver
implementation
67  /// @param _protocolId Protocol identifier
68  /// @param _locker Address of the locker
69  /// @param _gateway Address of the gateway
70  constructor(
71      address _protocolController,
72      address _vaultImplementation,
```

Example (Cont.):

```sol
    address _rewardReceiverImplementation,
    bytes4 _protocolId,
    address _locker,
    address _gateway
) ProtocolContext(_protocolId, _protocolController, _locker, _gateway) {
    require(
        _protocolController != address(0) && _vaultImplementation != address(0)
            && _rewardReceiverImplementation != address(0),
        ZeroAddress()
    );

    REWARD_VAULT_IMPLEMENTATION = _vaultImplementation;
    REWARD_RECEIVER_IMPLEMENTATION = _rewardReceiverImplementation;
}
```

Recommendation:

We advise the validation within the `Factory::constructor` to be removed, optimizing the code's gas cost.

Alleviation:

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

ImmutableArgsParser Code Style Findings

IAP-01C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	ImmutableArgsParser.sol:L16, L25

Description:

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

Example:

packages/strategies/src/libraries/ImmutableArgsParser.sol

SOL

```
16  result := shr(96, mload(add(add(args, 0x20), offset)))
```

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

ProtocolContext Code Style Findings

PCT-01C: Inefficient ABI Encoding of Payload

Type	Severity	Location
Gas Optimization	Informational	ProtocolContext.sol:L92

Description:

The referenced ABI encoding mechanism is inefficient as it utilizes a `string` literal to denote the `IGateway::execute` function signature.

Example:

packages/strategies/src/ProtocolContext.sol

SOL

```sol
success = IModuleManager(GATEWAY).execTransactionFromModule(
    LOCKER,
    0,
    abi.encodeWithSignature("execute(address,uint256,bytes)", target, 0, data),
    IModuleManager.Operation.Call
);
```

Recommendation:

We advise a proper `interface` to be declared and imported for it, permitting the `IGateway.execute.selector` syntax to be utilized in conjunction with the `abi.encodeWithSelector` implementation in an optimal way.

Alleviation:

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

ProtocolController Code Style Findings

PCR-01C: Inefficient `mapping` Lookups

Type	Severity	Location
Gas Optimization	Informational	**ProtocolController.sol**: • I-1: **L236**, **L238** • I-2: **L288**, **L290** • I-3: **L298**, **L300** • I-4: **L307**, **L310** • I-5: **L318**, **L320** • I-6: **L327**, **L329**

Description:

The linked statements perform key-based lookup operations on `mapping` declarations from storage multiple times for the same key redundantly.

Example:

packages/strategies/src/ProtocolController.sol

SOL

```sol
236 require(_protocolComponents[protocolId].accountant == address(0),
AccountantAlreadySet());
237
238 _protocolComponents[protocolId].accountant = _accountant;
```

Recommendation:

As the lookups internally perform an expensive `keccak256` operation, we advise the lookups to be cached wherever possible to a single local declaration that either holds the value of the `mapping` in case of primitive types or holds a `storage` pointer to the `struct` contained.

As the compiler's optimizations may take care of these caching operations automatically at-times, we advise the optimization to be selectively applied, tested, and then fully adopted to ensure that the proposed caching model indeed leads to a reduction in gas costs.

Alleviation:

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

RewardVault Code Style Findings

RVT-01C: Ineffectual Usage of Safe Arithmetics

Type	Severity	Location
Language Specific	Informational	RewardVault.sol:L491

Description:

The linked mathematical operation is guaranteed to be performed safely by logical inference, such as surrounding conditionals evaluated in `require` checks or `if-else` constructs.

Example:

packages/strategies/src/RewardVault.sol

SOL

```
488 if (currentTime >= periodFinish) {
489     newRewardRate = Math.mulDiv(amount, 1e18,
DEFAULT_REWARDS_DURATION).toUint128();
490 } else {
491     uint32 remainingTime = periodFinish - currentTime;
```

Recommendation:

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

Alleviation:

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

RVT-02C: Non-Standard Usages of Library

Type	Severity	Location
Code Style	Informational	**RewardVault.sol**: • I-1: **L310** • I-2: **L382** • I-3: **L445**

Description:

The referenced `SafeERC20` library statements are non-standard as they access the relevant functions from the library instance itself.

Example:

packages/strategies/src/RewardVault.sol

SOL

```
310 SafeERC20.safeTransferFrom(_asset, account, allocation.targets[i],
allocation.amounts[i]);
```

Recommendation:

We advise the `SafeERC20` library functions to be accessed from the `IERC20` data type directly akin to the `RewardVault::depositRewards` function.

Alleviation:

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

RVT-03C: Redundant Invocations of String Concatenations

Type	Severity	Location
Gas Optimization	Informational	RewardVault.sol:L178

Description:

The `RewardVault::constructor` will invoke the built-in `string.concat` functions with single-argument payloads which effectively act as no-ops.

Example:

packages/strategies/src/RewardVault.sol

SOL

```
178 ERC20(string.concat("StakeDAO Vault"), string.concat("sd-vault"))
```

Recommendation:

We advise this to be corrected by specifying the strings directly, optimizing the contract's deployment cost.

Alleviation:

The referenced `string.concat` operations have been omitted as advised, optimizing the code's gas cost.

RVT-04C: Repetitive Value Literal

Type	Severity	Location
Code Style	Informational	RewardVault.sol:L489, L492, L493, L608

Description:

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

Example:

packages/strategies/src/RewardVault.sol

SOL

```
489 newRewardRate = Math.mulDiv(amount, 1e18, DEFAULT_REWARDS_DURATION).toUint128();
```

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

Router Code Style Findings

RRE-01C: Inefficient Application of Modifier

Type	Severity	Location
Gas Optimization	Informational	Router.sol:L77, L92

Description:

The `Ownable::onlyOwner` modifier is redundantly applied twice during a `Router::safeSetModule` call; once at the top-level call and once at the inner `Router::setModule` call.

Example:

packages/strategies/src/Router.sol

SOL

```
66  /// @notice Sets a module
67  /// @dev The module is set at the storage slot `buffer + identifier`
68  ///
69  ///      While not enforced by the code, developers are expected to use
70  ///      incremental identifiers when setting modules.
71  ///      This allows modules to be enumerated using the `enumerateModules`
helper.
72  ///      Note that this is just a convention, and modules should be indexed off-
chain for
73  ///      efficiency and correctness.
74  /// @param identifier The unique identifier of the module
75  /// @param module The address of the module
```

Example (Cont.):

```sol
/// @custom:throws OwnableUnauthorizedAccount if the caller is not the owner
function setModule(uint8 identifier, address module) public onlyOwner {
    bytes32 buffer = getStorageBuffer();
    assembly ("memory-safe") {
        sstore(add(buffer, identifier), module)
    }

    emit ModuleSet(identifier, module, IRouterModule(module).name());
}

/// @notice Sets a module in safe mode
/// @dev The module can be set to address(0) to erase it
/// @param identifier The unique identifier of the module
/// @param module The address of the module
/// @custom:throws OwnableUnauthorizedAccount if the caller is not the owner
/// @custom:throws IdentifierAlreadyUsed if the identifier is already set
function safeSetModule(uint8 identifier, address module) external onlyOwner {
    require(getModule(identifier) == address(0),
IdentifierAlreadyUsed(identifier));

    setModule(identifier, module);
}
```

Recommendation:

We advise it to be applied once by omitting it from the `Router::safeSetModule` function call, optimizing the code's gas cost.

Alleviation:

The redundant application of the modifier has been omitted as advised.

RRE-02C: Inefficient Loop Iterator

Type	Severity	Location
Gas Optimization	Informational	Router.sol:L126

Description:

The referenced `for` loop will utilize a `uint8` iterator which is inefficient.

Example:

packages/strategies/src/Router.sol

SOL

```
126 for (uint8 i; i < type(uint8).max; i++) {
```

Recommendation:

We advise the iterator to be updated to the `uint256` data type, optimizing the code's gas cost.

Alleviation:

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

RouterIdentifierMapping Code Style Findings

RIM-01C: Inefficient Data Type Usage

Type	Severity	Location
Code Style	Informational	RouterIdentifierMapping.sol:L5-L10

Description:

The referenced `constant` declarations represent various identifiers that the router is meant to distinguish entries by, however, they are declared as `uint8` variables.

Example:

packages/strategies/src/RouterModules/RouterIdentifierMapping.sol

SOL

```sol
5   uint8 internal constant DEPOSIT = 0x00;
6   uint8 internal constant WITHDRAW = 0x01;
7   uint8 internal constant CLAIM = 0x02;
8   uint8 internal constant MIGRATION_STAKE_DAO_V1 = 0x03;
9   uint8 internal constant MIGRATION_CURVE = 0x04;
10  uint8 internal constant MIGRATION_YEARN = 0x05;
```

Recommendation:

We advise the code to either utilize an `enum` declaration or the data types to be upgraded to the `uint256` data type as the former recommendation would increase the codebase's legibility whilst the latter would decrease its gas cost.

Alleviation:

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

RouterModuleMigrationCurve Code Style Findings

RMM-01C: Inefficient Import of Logic Implementation

Type	Severity	Location
Gas Optimization	Informational	RouterModuleMigrationCurve.sol:L7

Description:

The `RouterModuleMigrationCurve` contract will import the full `RewardVault` logic implementation even though it is meant to be utilized as an `interface`.

Example:

packages/strategies/src/RouterModules/RouterModuleMigrationCurve.sol

SOL

```
7   import {RewardVault} from "src/RewardVault.sol";
```

Recommendation:

We advise a proper `interface` to be introduced and imported to the contract, optimizing the code's generated bytecode.

Alleviation:

An `IERC4626` interface is now imported in place of the `RewardVault` implementation, addressing this exhibit.

RMM-02C: Redundant Code Duplication

Type	Severity	Location
Code Style	Informational	RouterModuleMigrationCurve.sol:L23-L36

Description:

The `RouterModuleMigrationCurve::migrate` function is identical to the `RouterModuleMigrationYearn::migrate` function with the sole difference being the asset evaluated for compatibility.

Example:

packages/strategies/src/RouterModules/RouterModuleMigrationCurve.sol

SOL

```sol
23  function migrate(address from, address to, address account, uint256 shares)
external {
24      address asset = RewardVault(to).asset();
25      require(ILiquidityGauge(from).lp_token() == asset, VaultNotCompatible());
26
27      // 1. Transfer the token of the user to the router contract
28      ILiquidityGauge(from).safeTransferFrom(account, address(this), shares);
29
30      // 2. Withdraw the tokens in the old vault
31      ILiquidityGauge(from).withdraw(shares);
32
```

Example (Cont.):

```sol
    // 3. Deposit the tokens in the reward vault
    ILiquidityGauge(asset).safeIncreaseAllowance(to, shares);
    RewardVault(to).deposit(shares, account);
}
```

Recommendation:

We advise the code of both implementations to be relocated to a "base" implementation inherited by both, optimizing the code's structure and gas cost.

Alleviation:

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

RouterModuleMigrationStakeDAOV1 Code Style Findings

RMS-01C: Inefficient Import of Logic Implementation

Type	Severity	Location
Gas Optimization	Informational	RouterModuleMigrationStakeDAOV1.sol:L7

Description:

The `RouterModuleMigrationStakeDAOV1` contract will import the full `RewardVault` logic implementation even though it is meant to be utilized as an `interface`.

Example:

packages/strategies/src/RouterModules/RouterModuleMigrationStakeDAOV1.sol

SOL

```sol
7   import {RewardVault} from "src/RewardVault.sol";
```

Recommendation:

We advise a proper `interface` to be introduced and imported to the contract, optimizing the code's generated bytecode.

Alleviation:

An `IERC4626` interface is now imported in place of the `RewardVault` implementation, addressing this exhibit.

RouterModuleMigrationYearn Code Style Findings

RMY-01C: Inefficient Import of Logic Implementation

Type	Severity	Location
Gas Optimization	Informational	RouterModuleMigrationYearn.sol:L7

Description:

The `RouterModuleMigrationYearn` contract will import the full `RewardVault` logic implementation even though it is meant to be utilized as an `interface`.

Example:

packages/strategies/src/RouterModules/RouterModuleMigrationYearn.sol

SOL

```
7   import {RewardVault} from "src/RewardVault.sol";
```

Recommendation:

We advise a proper `interface` to be introduced and imported to the contract, optimizing the code's generated bytecode.

Alleviation:

An `IERC4626` interface is now imported in place of the `RewardVault` implementation, addressing this exhibit.

RMY-02C: Redundant Code Duplication

Type	Severity	Location
Code Style	Informational	RouterModuleMigrationYearn.sol:L23-L36

Description:

The `RouterModuleMigrationYearn::migrate` function is identical to the `RouterModuleMigrationStakeDAOV1::migrate` function with the sole difference being the asset transferred inward.

Example:

packages/strategies/src/RouterModules/RouterModuleMigrationYearn.sol

SOL

```
17  /// @notice Migrates shares from convex to a reward vault
18  /// @dev The account must have approved the token to the router contract
19  /// @param from The address of the old vault
20  /// @param to The address of the new reward vault
21  /// @param account The address of the account to migrate
22  /// @param shares The number of shares to migrate
23  function migrate(address from, address to, address account, uint256 shares)
external {
24      address asset = RewardVault(to).asset();
25      require(IYearnVault(from).token() == asset, VaultNotCompatible());
26
```

Example (Cont.):

```sol
    // 1. Transfer the token of the user to the router contract
    IYearnVault(from).safeTransferFrom(account, address(this), shares);

    // 2. Withdraw the tokens in the old vault
    IYearnVault(from).withdraw(shares);

    // 3. Deposit the tokens in the reward vault
    IYearnVault(asset).safeIncreaseAllowance(to, shares);
    RewardVault(to).deposit(shares, account);
}
```

Recommendation:

We advise the code of both implementations to be relocated to a "base" implementation inherited by both, optimizing the code's structure and gas cost.

Alleviation:

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

Sidecar Code Style Findings

SRA-01C: Redundant Validation of Argument

Type	Severity	Location
Gas Optimization	Informational	Sidecar.sol:L81

Description:

The `_accountant` argument is validated as a non-zero address yet a function call is made to it within the same code block.

Example:

packages/strategies/src/Sidecar.sol

SOL

```sol
79  constructor(bytes4 _protocolId, address _accountant, address _protocolController)
{
80      require(_protocolId != bytes4(0), InvalidProtocolId());
81      require(_accountant != address(0) && _protocolController != address(0),
ZeroAddress());
82
83      PROTOCOL_ID = _protocolId;
84      ACCOUNTANT = _accountant;
85      PROTOCOL_CONTROLLER = IProtocolController(_protocolController);
86      REWARD_TOKEN = IERC20(IAccountant(_accountant).REWARD_TOKEN());
87
88      _initialized = true;
```

Example (Cont.):

```sol
89  }
```

Recommendation:

We advise the explicit validation to be omitted as the function would fail in the ensuing getter function invocations on the `_accountant`.

Alleviation:

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

SidecarFactory Code Style Findings

SFY-01C: Redundant Validation of Argument

Type	Severity	Location
Gas Optimization	Informational	SidecarFactory.sol:L59

Description:

The `_protocolController` argument is validated as a non-zero address yet a function call is made to it within the same code block.

Example:

packages/strategies/src/SidecarFactory.sol

SOL

```sol
58  constructor(bytes4 _protocolId, address _implementation, address
_protocolController) {
59      require(_implementation != address(0) && _protocolController != address(0),
ZeroAddress());
60      require(_protocolId != bytes4(0), InvalidProtocolId());
61
62      IMPLEMENTATION = _implementation;
63      PROTOCOL_ID = _protocolId;
64      PROTOCOL_CONTROLLER = IProtocolController(_protocolController);
65
66      ACCOUNTANT = PROTOCOL_CONTROLLER.accountant(PROTOCOL_ID);
67      REWARD_TOKEN = IAccountant(ACCOUNTANT).REWARD_TOKEN();
```

Example (Cont.):

```sol
    STRATEGY = PROTOCOL_CONTROLLER.strategy(PROTOCOL_ID);
}
```

Recommendation:

We advise the explicit validation to be omitted as the function would fail in the ensuing getter function invocations on the `PROTOCOL_CONTROLLER`.

Alleviation:

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

Strategy Code Style Findings

SYG-01C: Non-Standard Empty Implementation

Type	Severity	Location
Standard Conformity	Informational	Strategy.sol:L445

Description:

The referenced implementation needs to be overridden by derivative implementations yet is declared as an empty block.

Example:

packages/strategies/src/Strategy.sol

SOL

```sol
419 /// @notice Synchronizes state of pending rewards.
420 /// @dev Must be implemented by derived strategies to handle protocol-specific reward collection
421 /// @param gauge The gauge to synchronize
422 /// @return Pending rewards collected during synchronization
423 function _checkpointRewards(address gauge) internal virtual returns (PendingRewards memory);
424
425 /// @notice Harvests rewards from the locker
426 /// @dev Must be implemented by derived strategies to handle protocol-specific reward collection
427 /// @param gauge The gauge to harvest rewards from
428 /// @param extraData Additional data needed for harvesting (protocol-specific)
```

Example (Cont.):

```sol
/// @return pendingRewards The pending rewards collected during harvesting
function _harvestLocker(address gauge, bytes memory extraData) internal virtual returns (uint256 pendingRewards);

/// @notice Deposits assets into a specific target
/// @dev Must be implemented by derived strategies to handle protocol-specific deposits
/// @param asset The asset to deposit
/// @param gauge The gauge to deposit into
/// @param amount The amount to deposit
function _deposit(address asset, address gauge, uint256 amount) internal virtual;

/// @notice Withdraws assets from a specific target
/// @dev Must be implemented by derived strategies to handle protocol-specific withdrawals
/// @param asset The asset to withdraw
/// @param gauge The gauge to withdraw from
/// @param amount The amount to withdraw
/// @param receiver The address to receive the withdrawn assets
function _withdraw(address asset, address gauge, uint256 amount, address receiver) internal virtual {}
```

Recommendation:

We advise the empty block to be omitted akin to the other functions declared within the contract.

Alleviation:

The referenced function was updated to contain a line-terminating character (`;`) instead of an empty function body in line with the rest of the codebase.

SYG-02C: Non-Standard Usage Statement

Type	Severity	Location
Standard Conformity	Informational	Strategy.sol:L24

Description:

The referenced library usage statement is non-specific causing the `TransientSlot` functions to be exposed by several native types.

Example:

packages/strategies/src/Strategy.sol

SOL

```sol
using TransientSlot for *;

/// @dev Slot for the flush amount in transient storage
bytes32 internal constant FLUSH_AMOUNT_SLOT = keccak256("strategy.flushAmount");
```

Recommendation:

We advise the library usage statement to be re-purposed for the `bytes32` data type as the `TransientSlot` functions are solely utilized on the `FLUSH_AMOUNT_SLOT` data type.

Alleviation:

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

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.