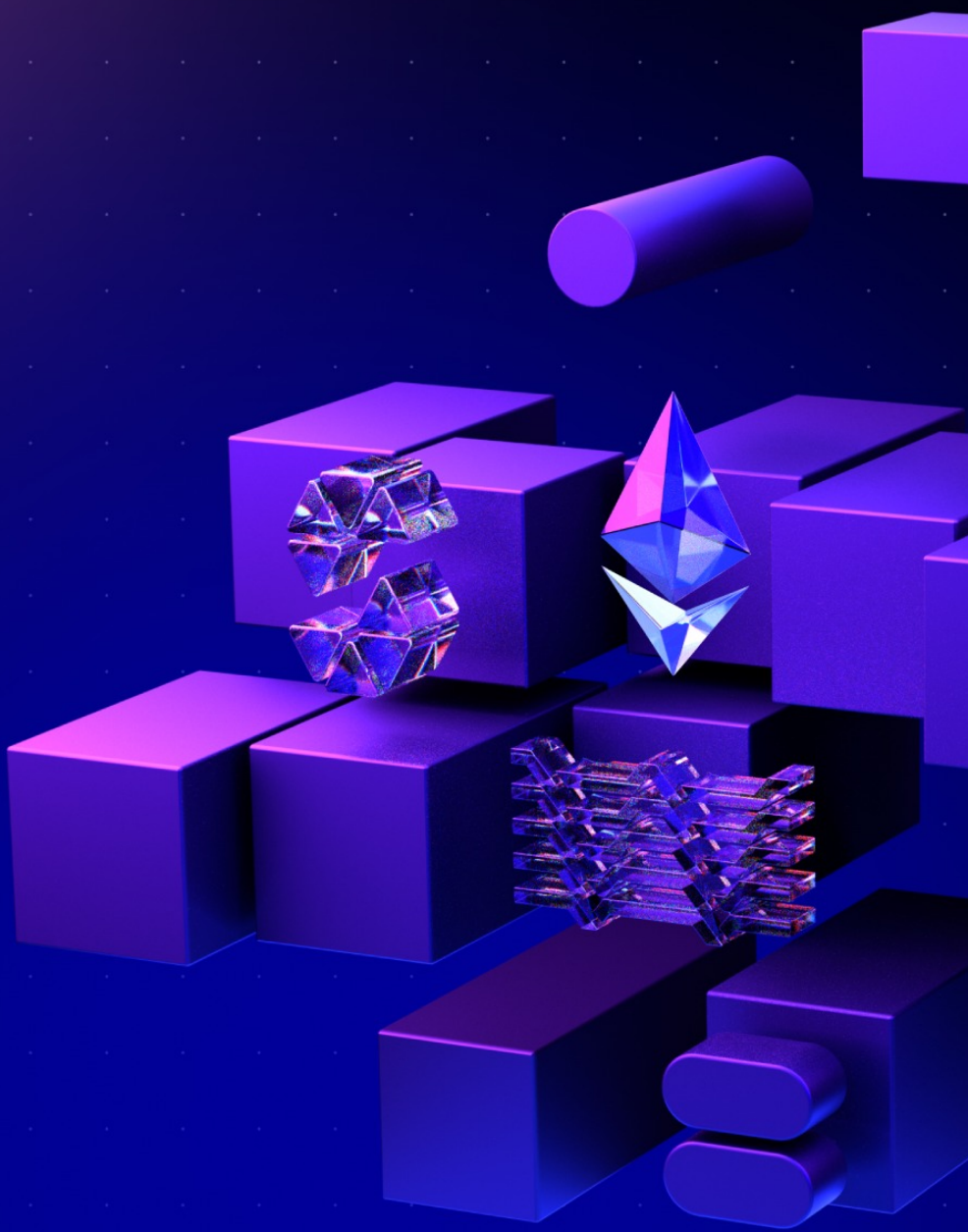
ackee
blockchain security

Lido

Vault Wrapper

February 6, 2026



Audit Report

Contents

1. Document Revisions . . . 3
2. Overview . . . 4
   2.1. Ackee Blockchain Security . . . 4
   2.2. Audit Methodology . . . 5
   2.3. Finding Classification . . . 6
   2.4. Review Team . . . 8
   2.5. Disclaimer . . . 8
3. Executive Summary . . . 9
   Revision 1.0 . . . 9
   Revision 1.1 . . . 11
   Revision 1.2 . . . 11
4. Findings Summary . . . 13
Report Revision 1.0 . . . 15
   Revision Team . . . 15
   System Overview . . . 15
   Trust Model . . . 16
   Fuzzing . . . 17
   Findings . . . 17
Appendix A: How to cite . . . 43
Appendix B: Wake Findings . . . 44
   B.1. Fuzzing . . . 44
   B.2. Detectors . . . 49
Appendix C: Wake Arena Findings . . . 51
   C.1. Discovered Findings . . . 51

1. Document Revisions

1.0-draft	Draft Report	December 5, 2025
1.1	Final Report	January 27, 2026
1.2	Deployment Verification	February 6, 2026

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain Security

Ackee Blockchain Security is an in-house team of security researchers performing security audits focusing on manual code reviews with extensive fuzz testing for Ethereum and Solana. Ackee is trusted by top-tier organizations in web3, securing protocols including Lido, Safe, and Axelar.

We develop open-source security and developer tooling Wake for Ethereum and Trident for Solana, supported by grants from Coinbase and the Solana Foundation. Wake and Trident help auditors in the manual review process to discover hardly recognizable edge-case vulnerabilities.

Our team teaches about blockchain security at the Czech Technical University in Prague, led by our co-founder and CEO, Josef Gattermayer, Ph.D. As the official educational partners of the Solana Foundation, we run the School of Solana and the Solana Auditors Bootcamp.

Ackee's mission is to build a stronger blockchain community by sharing our knowledge.

Ackee Blockchain a.s.
Rohanske nabrezi 717/4
186 00 Prague, Czech Republic
https://ackee.xyz
hello@ackee.xyz

2.2. Audit Methodology

1. **Verification of technical specification**

   The audit scope is confirmed with the client, and auditors are onboarded to the project. Provided documentation is reviewed and compared to the audited system.

2. **Tool-based analysis**

   A deep check with Solidity static analysis tool Wake in companion with Solidity (Wake) extension is performed, flagging potential vulnerabilities for further analysis early in the process.

3. **Manual code review**

   Auditors manually check the code line by line, identifying vulnerabilities and code quality issues. The main focus is on recognizing potential edge cases and project-specific risks.

4. **Local deployment and hacking**

   Contracts are deployed in a local Wake environment, where targeted attempts to exploit vulnerabilities are made. The contracts' resilience against various attack vectors is evaluated.

5. **Unit and fuzz testing**

   Unit tests are run to verify expected system behavior. Additional unit or fuzz tests may be written using Wake framework if any coverage gaps are identified. The goal is to verify the system's stability under real-world conditions and ensure robustness against both expected and unexpected inputs.

6. **Wake Arena assisted vulnerability discovery**

   As the last step, the scope is checked against Wake Arena, an LLM-powered audit tool, to identify potentially missed vulnerabilities. This step is executed at the end of the audit process to avoid distracting auditors from manual review.

2.3. Finding Classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in *configuration* (system settings or parameters, such as deployment scripts, compiler configurations, using multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**N/A**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Low	-
	Low	Medium	Low	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or *configuration*, but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or *configuration* was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review Team

The following table lists all contributors to this report. For authors of the specific revision, see the “Revision team” section in the respective “Report revision” chapter.

Member’s Name	Position
Dmytro Khimchenko	Lead Auditor
Naoki Yoshida	Auditor
Jan Kalivoda	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We’ve put our best effort to find all vulnerabilities in the system, however our findings shouldn’t be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

Revision 1.0

Lido engaged Ackee Blockchain Security to perform a security review of Lido Vault Wrapper with a total time donation of 26 engineering days in a period between November 13 and December 5, 2025, with Dmytro Khimchenko as the lead auditor.

The audit was performed on the commit `ac004a8`[1] in the contracts repository and the scope was the following:

- `src/factories/*.sol`
- `src/proxy/*.sol`
- `src/strategy/*.sol`
- `src/utils/FeaturePausable.sol`
- `src/Allowlist.sol`
- `src/Distributor.sol`
- `src/Factory.sol`
- `src/StvPool.sol`
- `src/StvStETHPool.sol`
- `src/WithdrawalQueue.sol`

We began our review by preparing and executing a manually-guided differential fuzz test in the Wake testing framework. The implemented fuzzing flows and invariants are available in Appendix B.

Then we took a deep dive into the in-scope contracts starting from the `StvPool` contract and its integration with the logic implemented in staking vaults. After that, we moved to the `StvStETHPool` contract and its integration

with other systems. During the time of analyzing the pools, we also analyzed withdrawal from `StvPool` and `StvStETHPool` via `WithdrawalQueue` contract. Finally, we moved to the `GGVStrategy` contract and its integration pools and staking vaults contracts.

During the review, we paid special attention to:

- ensuring the arithmetic operations involving stv tokens are correct;
- ensuring that rebalancing mechanism works correctly;
- checking that for every important state change call the freshness of the oracle report is checked;
- confirming accounting is implemented correctly;
- ensuring rounding mistakes are not made or handled correctly;
- checking finalization mechanism of the queue works correctly and cannot be DoSed; and
- looking for common issues such as data validation.

At the end of the review, we engaged the Wake Arena, which discovered the following issues I7, I8, I9.

Our review resulted in 14 findings, ranging from Info to Low severity. The most severe ones are L1 and L2 and are discovered during manually-guided fuzzing.

The codebase is well-written and thoroughly documented. It includes several emergency and recovery mechanisms and correctly manages accounting of stETH, wstETH and stv tokens. Most of the findings in the latest commit are minor issues or optional recommendations for further improvements.

Ackee Blockchain Security recommends Lido:

- read and review the complete audit report; and

- address all identified issues.

See Report Revision 1.0 for the system overview and trust model.

Revision 1.1

Lido engaged Ackee Blockchain Security to perform a fix review of the findings from the previous revision.

Lido provided a pull request PR #94 with the fixes and added one more PR #102 with fixed versions of the OpenZeppelin contracts and provided comments to most of the acknowledged findings.

The review was performed between January 26 and January 29, 2026 on the commit `f35fe13`[2].

From the reported 14 findings:

- 13 issues were fixed; and
- 1 issues were acknowledged;

No new findings were reported.

Revision 1.2

Lido engaged Ackee Blockchain Security to perform deployment verification of Vault Wrapper on the Ethereum mainnet. The verification was performed on the same commit as in the previous revision, `f35fe13`[3].

The verification concluded successfully with an exact bytecode match and reasonable initialization values for all of the following contracts on the Ethereum mainnet:

- StvPoolFactory: 0x5Def7fBC0211351139B928f307EDC794af845Bde

- StvStETHPoolFactory: 0x671978CEEa7DAf405fA08E930E1047d1b7b21a69
- WithdrawalQueueFactory: 0xB011531857B6006479627e776feB6c0cEA5fc74a
- DistributorFactory: 0x9FD67B2D5b88BeBaC741EE50510cf808B4854a5F
- TimelockFactory: 0x15f2C9ea98e5564d25A46eE39D19704476998786
- Factory: 0x3f221b8E5bC098cC6C23611BEeacaeCfD77e1587
- DummyImplementation: 0x468029A88b6f75Eb1D13BB291fC3B82fc2C0232F

[1] full commit hash: `ac004a859f1fb2c33b0e5767ffbffc78651130fe`, link to commit

[2] full commit hash: `f35fe13ddca2084d7b848172b2698feb4fa88025`, link to commit

[3] full commit hash: `f35fe13ddca2084d7b848172b2698feb4fa88025`, link to commit

4. Findings Summary

The following section summarizes findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- *Description*
- *Exploit scenario* (if severity is low or higher)
- *Recommendation*
- *Fix* (if applicable).

Summary of findings:

Critical	High	Medium	Low	Warning	Info	Total
0	0	0	2	3	9	**14**

Table 2. Findings Count by Severity

Findings in detail:

Finding title	Severity	Reported	Status
L1: SocializedLoss event emits incorrect asset amount	Low	1.0	Fixed
L2: Rebalance preview calculation inconsistency	Low	1.0	Fixed
W1: Shares cannot be burnt when exceeding liability exists on pool and liability on vault is low	Warning	1.0	Fixed
W2: Possible type mismatch during calling `BORING_QUEUE.requestOnChainWithdraw`	Warning	1.0	Fixed

Finding title	Severity	Reported	Status
W3: Unsafe transfer in `recoverERC20` function	Warning	1.0	Fixed
I1: Misleading error name in `GGVStrategy`	Info	1.0	Fixed
I2: Potentially misleading `reserveRatioBp` variable name	Info	1.0	Fixed
I3: Incomplete documentation for checkpoint hint function	Info	1.0	Fixed
I4: Unused interfaces and libraries	Info	1.0	Acknowledged
I5: Unused errors	Info	1.0	Fixed
I6: Public functions can be declared as external	Info	1.0	Fixed
I7: Role constants read from Dashboard implementation instead of proxy	Info	1.0	Fixed
I8: createPoolFinish forwards full `msg.value` instead of exact connect deposit potentially causing loss of overpaid ETH	Info	1.0	Fixed
I9: calculateCurrentStethShareRate documentation contradicts implementation precision	Info	1.0	Fixed

Table 3. Table of Findings

Report Revision 1.0

Revision Team

Member's Name	Position
Dmytro Khimchenko	Lead Auditor
Naoki Yoshida	Auditor
Jan Kalivoda	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

System Overview

Lido Vault Wrapper is a capital-efficient staking vault system that allows users to deposit ETH and mint liquid staking tokens (stETH and wstETH) while receiving staking rewards.

The system lets users to deposit ETH and receive STV tokens, which represent their share in the vault. Users can leverage their deposits by minting stETH or wstETH against their STV holdings, subject to configurable reserve ratios enforced by the protocol.

The core functionality is implemented through four main contracts. The `StvPool` contract provides the base ERC20 token functionality and handles ETH deposits, asset accounting, and basic withdrawal operations. The `StvStETHPool` contract extends this with stETH minting capabilities, liability tracking, and forced rebalancing mechanisms to maintain protocol health.

The `WithdrawalQueue` contract manages the withdrawal lifecycle, implementing a queuing system with minimum delays, oracle-based pricing at finalization, and optional stETH liability rebalancing. The system supports discounting mechanisms when the STV price decreases between request and finalization.

The `GGVStrategy` contract provides an integration layer for the Boring Vault protocol, allowing users to deposit minted wstETH into external yield strategies while maintaining liability tracking through the vault system. The strategy uses a call forwarder pattern to maintain user-specific accounting while interacting with external protocols.

Trust Model

The project trusts the admin of the Staking Vault contracts to correctly manage the staking process and the withdrawal queue. If the admin is compromised, the system can corrupt and users can lose their funds.

The system relies on Lido's stETH protocol for staking operations and share rate calculations, and uses a LazyOracle for timestamp-based freshness checks for accurate pricing during deposits and withdrawals. The VaultHub manages vault connections and reserve ratio parameters that are synchronized to the pool.

User operations such as deposits, minting stETH against collateral, requesting withdrawals, and claiming finalized withdrawals are permissionless. Force rebalancing of undercollateralized positions is also permissionless when health thresholds are breached, helping to maintain stability of the protocol without relying on privileged actors.

Administrative operations are role-restricted. The `FINALIZE_ROLE` is required to finalize withdrawal requests and set gas cost coverage. The `LOSS_SOCIALIZER_ROLE` can force rebalance undercollateralized accounts with loss socialization. The `DEFAULT_ADMIN_ROLE` manages other roles and configures the maximum allowed loss socialization percentage. Feature-specific pause and resume operations require dedicated roles for deposits, minting, withdrawals, and finalization.

The GGVStrategy contract uses a call forwarder pattern where each user

receives a dedicated proxy contract. Users does not have full control over their forwarder and uses it via the strategy contract. However, users has ability to easily interact with the strategies and pools. Besides, there is ability to recover tokens from the forwarder.

Fuzzing

A manually-guided differential stateful fuzz test was developed during the review to test the correctness and robustness of the system. The fuzz test employs fork testing technique to test the system with external contracts exactly as they are deployed in the deployment environment. This is crucial to detect any potential integration issues. The differential fuzz test keeps its own Python state according to the system's specification. Assertions are used to verify the Python state against the on-chain state in contracts.

The list of all implemented execution flows and invariants is available in Appendix B.

Findings

The following section presents the list of findings discovered in this revision. For the complete list of all findings, Go back to Findings Summary

L1: SocializedLoss event emits incorrect asset amount

Low severity issue

Impact:	Low	Likelihood:	Medium
Target:	StvStETHPool.sol	Type:	Logic error

Description

Listing 1. Excerpt from StvStETHPool._rebalanceMintedStethShares

```
673 if (remainingStethShares > 0)
    DASHBOARD.rebalanceVaultWithShares(remainingStethShares);
674
675 if (stvToBurn > _maxStvToBurn) {
676     _checkAllowedLossSocializationPortion(stvToBurn, _maxStvToBurn);
677
678     emit SocializedLoss(
679         stvToBurn - _maxStvToBurn,
680         ethToRebalance - _convertToAssets(_maxStvToBurn),
681         _getStvStETHPoolStorage().maxLossSocializationBP
682     );
```

The `assets` parameter in the `SocializedLoss` event returns a larger value than expected.

The function `_convertToAssets(_maxStvToBurn)` calculates `_maxStvToBurn * totalAssets / totalSupply` in the `StvStETHPool` contract.

However, `DASHBOARD.rebalanceVaultWithShares` rebalances by decreasing the asset of the vault to cover the user's share, so `totalAssets` decreases before the event is emitted.

Exploit scenario

Alice, a protocol operator, monitors on-chain events to track ETH shortfalls.

Bob, a user, triggers a rebalance that causes a socialized loss. The `SocializedLoss` event emits with the mutated `totalAssets` value rather than the pre-rebalance value. Alice's off-chain accounting system records an incorrect delta, leading to discrepancies in loss tracking.

Recommendation

Calculate the corresponding asset value before calling `rebalanceVaultWithShares`.

Fix 1.1

The finding was fixed by calculating the corresponding asset value before calling `rebalanceVaultWithShares`.

Go back to Findings Summary

L2: Rebalance preview calculation inconsistency

Low severity issue

Impact:	Low	Likelihood:	Medium
Target:	StvStETHPool.sol	Type:	Logic error

Description

Listing 2. Excerpt from StvStETHPool._rebalanceMintedStethShares

```
670 uint256 ethToRebalance = _getPooledEthBySharesRoundUp(_stethShares);
671 stvToBurn = _convertToStv(ethToRebalance, Math.Rounding.Ceil);
```

The function `previewForceRebalance` computes the burn amount as `stvRequired = _convertToStv(stethToRebalance, Ceil)`, but execution burns via an stETH shares round-trip:

1. `stEthShares = Lido.getSharesByPooledEth(stethToRebalance)`
2. `ethToRebalance = Lido.getPooledEthbysharesRoundUp(stEthShares)`
3. `stvToBurn = _convertToStv(ethToRebalance, Ceil)`

Therefore, the rounding returns a different value. The value `stvToBurn` is equal to or smaller than `stvRequired` because `getSharesByPooledEth` rounds down first.

The value returns correctly when socialization happens, as it burns all of the user's STV tokens.

Exploit scenario

Alice, an integrator, attempts to rebalance her position and estimates the required STV balance using `previewForceRebalance`. However, the preview returns a larger amount than the actual required value. This discrepancy

confuses or breaks applications that expect the preview to match the actual execution, such as off-chain planners or integrators.

Recommendation

Add the rounding calculation in the `previewForceRebalance` function to match the execution path.

Fix 1.1

The finding was fixed by changing the calculation of `stv` in the `previewForceRebalance` function to match the calculation of `stvToBurn` in the `StvStETHPool._rebalanceMintedStethShares` function.

Go back to Findings Summary

W1: Shares cannot be burnt when exceeding liability exists on pool and liability on vault is low

Impact:	Warning	Likelihood:	N/A
Target:	StvStETHPool.sol	Type:	Logic error

Description

During direct staking vault rebalancing, the `liabilityShares` field in the `VaultRecord _record` structure is decreased. However, this change does not affect the `mintedStethShares` mapping in the `StvETHPool` contract, which is used during execution of the `burnStethShares` and `burnWsteth` functions. This discrepancy prevents users from burning shares to repay their debt even when the liability has been reduced through rebalancing.

Listing 3. Excerpt from StvStETHPool.burnWsteth

```
338 function burnWsteth(uint256 _wsteth) external {
339     /// @dev Simulate conversions during unwrapping to account for possible
    reduction due to rounding errors
340     uint256 unwrappedSteth = _getPooledEthByShares(_wsteth);
341     uint256 unwrappedStethShares = _getSharesByPooledEth(unwrappedSteth);
342     _decreaseMintedStethShares(msg.sender, unwrappedStethShares);
343
344     // Transfer on WSTETH contract always return true or revert
345     assert(WSTETH.transferFrom(msg.sender, address(this), _wsteth));
346     DASHBOARD.burnWstETH(_wsteth);
347 }
```

Listing 4. Excerpt from StvStETHPool.burnStethShares

```
353 function burnStethShares(uint256 _stethShares) external {
354     _decreaseMintedStethShares(msg.sender, _stethShares);
355     STETH.transferSharesFrom(msg.sender, address(this), _stethShares);
356     DASHBOARD.burnShares(_stethShares);
```

The impact is restricted to users not being able to repay their debt using the

`burnStethShares` and `burnWsteth` functions. In rare cases, a user position can become unhealthy when `_record.liabilityShares` reaches 0 due to rebalancing, making the position subject to force rebalancing. However, there is a possibility to use the `WithdrawalQueue` contract to withdraw the remaining stETH shares and repay the debt.

Exploit scenario

Consider the following scenario with `reserveRatioBP = 20_00`:

1. Alice, a user, deposits 100 ETH and mints 60 stETH.
2. Bob, another user, deposits 50 ETH and mints 30 stETH.
3. The liability in the staking vault reaches approximately 90 ETH.
4. The protocol calls the `rebalanceVaultWithShares` function on the Dashboard with `_shares=30` stETH, which decreases `_record.liabilityShares` to 60 stETH.
5. Alice calls `burnStethShares` to burn her 60 stETH shares to repay the debt fully, and the transaction succeeds.
6. Bob calls `burnStethShares` to burn his 30 stETH shares, but the transaction reverts with `InsufficientSharesToBurn`, despite the fact that Bob should be able to repay his debt.

Recommendation

Implement the same logic as used in the `_rebalanceMintedStethShares` function. Check for exceeding shares and only burn the remaining stETH shares via the vault hub when the debt has been reduced through rebalancing.

Fix 1.1

The finding was fixed by adding

`StvStETHPool.rebalanceExceedingMintedStethShares` as recommended.

Go back to Findings Summary

W2: Possible type mismatch during calling `BORING_QUEUE.requestOnChainWithdraw`

Impact:	Warning	Likelihood:	N/A
Target:	GGVStrategy.sol	Type:	Data validation

Description

The function `GGVStrategy.requestExitByWsteth` calls `BORING_QUEUE.requestOnChainWithdraw` with an incorrect parameter type. The function passes `ggvShares.toInt256()` as the second argument, converting a `uint256` value to `int256`, while the `requestOnChainWithdraw` function signature expects `uint128 amountOfShares`.

Listing 5. Excerpt from GGVQueueMock

```
99 function requestOnChainWithdraw(
100     address _assetOut,
101     uint128 amountOfShares,
102     uint16 discount,
103     uint24 secondsToDeadline
104 ) external returns (bytes32 requestId) {
```

Listing 6. Excerpt from GGVStrategy

```
199 bytes memory data = callForwarder.doCall(
200     address(BORING_QUEUE),
201     abi.encodeWithSelector(
202         BORING_QUEUE.requestOnChainWithdraw.selector,
203         address(WSTETH),
204         ggvShares.toInt256(),
205         params.discount,
206         params.secondsToDeadline
207     )
208 );
```

The type mismatch could cause transaction failures when the ABI encoding does not match the expected function signature, or potentially result in

unexpected behavior if the contract accepts the call with incorrect encoding. The assumption is based on the fact that the `GGVQueueMock` contract implements the `requestOnChainWithdraw` function similarly to the production contract.

Recommendation

Change the type conversion from `.toInt256()` to `.toUint128()` when calling `BORING_QUEUE.requestOnChainWithdraw`.

Fix 1.1

The finding was fixed by changing the type conversion from `.toInt256()` to `.toUint128()` when calling `BORING_QUEUE.requestOnChainWithdraw`.

Go back to Findings Summary

W3: Unsafe transfer in `recoverERC20` function

Impact:	Warning	Likelihood:	N/A
Target:	GGVStrategy.sol	Type:	Data validation

Description

The `recoverERC20` function in the `GGVStrategy` contract uses the `IERC20.transfer` selector without checking the return value or handling tokens that do not conform to the ERC20 standard.

Listing 7. Excerpt from GGVStrategy.recoverERC20

```
366 function recoverERC20(address _token, address _recipient, uint256 _amount)
    external {
367     if (_token == address(0)) revert ZeroArgument("_token");
368     if (_recipient == address(0)) revert ZeroArgument("_recipient");
369     if (_amount == 0) revert ZeroArgument("_amount");
370
371     IStrategyCallForwarder callForwarder =
    _getOrCreateCallForwarder(msg.sender);
372     callForwarder.doCall(_token,
    abi.encodeWithSelector(IERC20.transfer.selector, _recipient, _amount));
373 }
```

Some ERC20 token implementations do not return a boolean value on transfer operations, such as USDT, while others return `false` instead of reverting on failure. The `doCall` function executes the transfer through a low-level call without verifying the result, which means:

- tokens that return `false` on failure will silently fail without reverting the transaction; and
- tokens that do not return any value may cause unexpected behavior during ABI decoding.

This could result in the function appearing to succeed while the tokens remain in the call forwarder contract, leading users to believe the recovery

was successful when it actually failed.

Recommendation

Use `SafeERC20` library wrapper around the transfer call. Replace `IERC20.transfer` with `SafeERC20.safeTransfer`, which handles both standard and non-standard ERC20 implementations by checking return values and ensuring the call succeeds.

Fix 1.1

The finding was fixed by using `SafeERC20.safeTransfer` instead of `IERC20.transfer` and function was renamed to `safeTransferERC20`.

Go back to Findings Summary

I1: Misleading error name in `GGVStrategy`

Impact:	Info	Likelihood:	N/A
Target:	GGVStrategy.sol	Type:	Code quality

Description

The error name `NothingToExit` is misleading, as it suggests that there are no tokens to be withdrawn. However, the reason for the error is that the user tries to withdraw more tokens than he has requested.

Listing 8. Excerpt from GGVStrategy.requestExitByWsteth

```
190 if (_wsteth > totalWstethFromGGV) revert NothingToExit();
```

Recommendation

Rename the error name from `NothingToExit` to better reflect the actual reason for the error.

Fix 1.1

The finding was fixed by renaming the error name from `NothingToExit` to `InsufficientWsteth`, which better reflects the actual reason for the error.

Go back to Findings Summary

I2: Potentially misleading `reserveRatioBp` variable name

Impact:	Info	Likelihood:	N/A
Target:	StvStETHPool.sol	Type:	Code quality

Description

Listing 9. Excerpt from StvStETHPool.forcedRebalanceThresholdBP

```
483 uint16 newReserveRatioBP = uint16(Math.min(connection.reserveRatioBP +
    RESERVE_RATIO_GAP_BP, maxReserveRatioBP));
484 uint16 newForcedRebalanceThresholdBP = uint16(
485     Math.min(connection.forcedRebalanceThresholdBP + RESERVE_RATIO_GAP_BP,
    maxForcedRebalanceThresholdBP)
486 );
487
488 StvStETHPoolStorage storage $ = _getStvStETHPoolStorage();
489
490 if (newReserveRatioBP == $.reserveRatioBP && newForcedRebalanceThresholdBP
    == $.forcedRebalanceThresholdBP) {
491     return;
492 }
493
494 $.reserveRatioBP = newReserveRatioBP;
495 $.forcedRebalanceThresholdBP = newForcedRebalanceThresholdBP;
```

The effective `reserveRatioBp` for the vault user in `StvStETHPool` is `ReservationGap` + `VaultConnection.reserveRatioBp`.

However, the same variable name is used in both contexts, which can cause confusion about its actual meaning.

Recommendation

Use a different variable name for the `StvStETHPool` user context.

Fix 1.1

The finding was fixed by renaming the variable from `reserveRatioBp` to `poolReserveRatioBP`.

Go back to Findings Summary

I3: Incomplete documentation for checkpoint hint function

Impact:	Info	Likelihood:	N/A
Target:	WithdrawalQueue.sol	Type:	N/A

Description

Listing 10. Excerpt from WithdrawalQueue

```
758 * @notice View function to find a checkpoint hint to use in
    `claimWithdrawalBatch()` and `getClaimableEther()`
```

The documentation states that the function is for use with `claimWithdrawalBatch` and `getClaimableEther`. However, this omits other relevant functions. The single functions `claimWithdrawal` and `getClaimableEther` call this automatically, while the batch functions `claimWithdrawalBatch` and `getClaimableEtherBatch` require this view function to be called manually.

Recommendation

Fix the comment to include all required functions.

Fix 1.1

The finding was fixed by updating the comment to include all required functions.

Go back to Findings Summary

I4: Unused interfaces and libraries

Impact:	Info	Likelihood:	N/A
Target:	ILido.sol, IOperatorGrid.sol, IOssifiableProxy.sol, IBoringSolver.sol, IVaultHub.sol	Type:	Code quality

Description

The following interfaces are defined but never used:

Listing 11. Excerpt from ILido

```
8 interface ILido is IStETH {
```

Listing 12. Excerpt from IOperatorGrid

```
6 interface IOperatorGrid is IAccessControlEnumerable {
```

Listing 13. Excerpt from IOssifiableProxy

```
7 interface IOssifiableProxy {
```

Listing 14. Excerpt from IBoringSolver

```
6 interface IBoringSolver {
```

The following library is never used:

Listing 15. Excerpt from IVaultHub

```
8 library DoubleRefSlotCache {
```

Recommendation

Remove the unused interfaces and libraries.

Acknowledgment 1.1

The finding was acknowledged by the team with the comment:

> *Acknowledged.*
>
> *IBoringSolver, IOperatorGrid, ILido, IOperatorGrid - used in tests - we'd like more to keep core/ggv interfaces altougether*
>
> *IOssifiableProxy - used in deploy scripts, makes sense to keep it as separate from OssifiableProxy entity*
>
> *DoubleRefSlotCache - removed*
>
> — Lido Team

Go back to Findings Summary

I5: Unused errors

Impact:	Info	Likelihood:	N/A
Target:	StvStETHPool.sol, Factory.sol	Type:	Code quality

Description

The following errors are defined but never used:

Listing 16. Excerpt from StvStETHPool

```
29 error ArraysLengthMismatch(uint256 firstArrayLength, uint256
   secondArrayLength);
```

Listing 17. Excerpt from Factory

```
221 error StringTooLong(string str);
```

Recommendation

Remove the unused errors.

Fix 1.1

The finding was fixed by removing the unused errors.

Go back to Findings Summary

I6: Public functions can be declared as external

Impact:	Info	Likelihood:	N/A
Target:	StvStETHPool.sol	Type:	Code quality

Description

The following functions can be declared as external to save gas on deployment:

Listing 18. Excerpt from StvStETHPool

```
564 function forceRebalance(address _account) public returns (uint256 stvBurned)
    {
```

Listing 19. Excerpt from StvStETHPool

```
580 function forceRebalanceAndSocializeLoss(address _account) public returns
    (uint256 stvBurned) {
```

Recommendation

Declare the functions as external to save gas on deployment.

Fix 1.1

The finding was fixed by declaring the functions as `external` instead of `public`.

Go back to Findings Summary

I7: Role constants read from Dashboard implementation instead of proxy

Impact:	Info	Likelihood:	N/A
Target:	Factory.sol	Type:	Logic error

Description

The function `Factory.createPoolFinish` fetches role identifiers from the global implementation returned by `VAULT_FACTORY.DASHBOARD_IMPL` and applies them to the specific Dashboard instance created during the start phase. If `DASHBOARD_IMPL` changes between `createPoolStart` and `createPoolFinish`, the role constants returned by the new implementation may not match the constants compiled into the implementation used by the already-deployed Dashboard. The `Factory` contract would then grant the wrong `bytes32` roles, leading to missing privileges. This misalignment can break flows such as `FUND_ROLE`, `REBALANCE_ROLE`, `WITHDRAW_ROLE`, `MINT_ROLE`, `BURN_ROLE`, or deposit pause controls.

Listing 20. Excerpt from Factory.createPoolFinish

```
647 IDashboard dashboardImpl = IDashboard(
    payable(VAULT_FACTORY.DASHBOARD_IMPL()));
648
649 dashboard.grantRole(dashboardImpl.FUND_ROLE(), _intermediate.poolProxy);
650 dashboard.grantRole(dashboardImpl.REBALANCE_ROLE(),
    _intermediate.poolProxy);
651 dashboard.grantRole(dashboardImpl.WITHDRAW_ROLE(),
    _intermediate.withdrawalQueueProxy);
652 if (_auxiliaryConfig.mintingEnabled) {
653     dashboard.grantRole(dashboardImpl.MINT_ROLE(), _intermediate.poolProxy);
654     dashboard.grantRole(dashboardImpl.BURN_ROLE(), _intermediate.poolProxy);
655 }
```

Recommendation

Read role constants from the already deployed Dashboard instance instead

of reading from the configured implementation.

Fix 1.1

The finding was fixed by reading role constants from the already deployed Dashboard instance instead of reading from the configured implementation.

Go back to Findings Summary

I8: createPoolFinish forwards full `msg.value` instead of exact connect deposit potentially causing loss of overpaid ETH

Impact:	Info	Likelihood:	N/A
Target:	Factory.sol	Type:	Data validation

Description

During the final phase of pool deployment, the function `Factory.createPoolFinish` forwards the entire `msg.value` to the `Dashboard.connectToVaultHub` function instead of the exact `CONNECT_DEPOSIT` amount. This can potentially cause loss of overpaid ETH if the user pays more than the required amount and is not informed about this behavior.

Listing 21. Excerpt from Factory.createPoolFinish

```
564 if (msg.value < VAULT_HUB.CONNECT_DEPOSIT()) {
565     revert InsufficientConnectDeposit(msg.value,
    VAULT_HUB.CONNECT_DEPOSIT());
566 }
567
568 bytes32 deploymentHash = _hashDeploymentConfiguration(
569     msg.sender,
570     _vaultConfig,
571     _commonPoolConfig,
572     _auxiliaryConfig,
573     _timelockConfig,
574     _strategyFactory,
575     _strategyDeployBytes,
576     _intermediate
577 );
578 uint256 finishDeadline = intermediateState[deploymentHash];
579 if (finishDeadline == 0) {
580     revert InvalidConfiguration("deploy not started");
581 } else if (finishDeadline == DEPLOY_FINISHED) {
582     revert InvalidConfiguration("deploy already finished");
583 }
584 if (block.timestamp > finishDeadline) {
585     revert InvalidConfiguration("deploy finish deadline passed");
```

```
586 }
587 intermediateState[deploymentHash] = DEPLOY_FINISHED;
588
589 address tempAdmin = address(this);
590
591 IDashboard dashboard = IDashboard(payable(_intermediate.dashboard));
592
593 dashboard.connectToVaultHub{value: msg.value}();
```

Recommendation

Add information about the behavior to the documentation or send exactly `CONNECT_DEPOSIT` amount to the `Dashboard.connectToVaultHub` function.

Fix 1.1

The finding was fixed by adding an information about the behavior of the function to the documentation.

Go back to Findings Summary

I9: calculateCurrentStethShareRate documentation contradicts implementation precision

Impact:	Info	Likelihood:	N/A
Target:	WithdrawalQueue.sol	Type:	Code quality

Description

The NatSpec for `calculateCurrentStethShareRate` claims a 1e27-precision return value, but the implementation returns a 1e18-precision ETH amount per 1e27 stETH shares. The rest of the codebase, including checkpoint storage and arithmetic, also assumes a 1e18-scaled value. This inconsistency can mislead off-chain integrators.

Listing 22. Excerpt from WithdrawalQueue

```
654 /**
655  * @notice Calculate current stETH share rate
656  * @return stethShareRate Current stETH share rate (1e27 precision)
657  */
658 function calculateCurrentStethShareRate() public view returns (uint256
    stethShareRate) {
659     stethShareRate = _getPooledEthBySharesRoundUp(E27_PRECISION_BASE);
660 }
```

Recommendation

Update the NatSpec for `calculateCurrentStethShareRate` to reflect the correct 1e18-precision ETH amount per 1e27 stETH shares.

Fix 1.1

The finding was fixed by updating the NatSpec to reflect the correct 1e18-precision ETH amount per 1e27 stETH shares.

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain Security, Audit Report | Lido: Vault Wrapper, February 6, 2026.

Appendix B: Wake Findings

This section lists the outputs from the Wake framework used for testing and static analysis during the audit.

B.1. Fuzzing

The following table lists all implemented execution flows in the Wake fuzzing framework.

The fuzzing is implemented in 4 files:

- `tests/test_vaults.py` tests the round state of the vaults contract;
- `tests/test_pool.py` overrides the above tests and adds additional functionality of the Vault Wrapper with `StETHPool`;
- `tests/test_stethpool.py` overrides the above tests and fuzzes additional functionality of `StvStETHPool`; and
- `tests/test_strategy_pool.py` overrides the above tests and fuzzes additional functionality of `GGVStrategy`.

Each derived test class can be run individually to test individual features.

ID	Flow	Added
F1	Creation of new staking vaults with dashboard	1.0
F2	Vault funding by vault owner with ETH	1.0
F3	Vault withdrawal by vault owner	1.0
F4	Minting stETH shares on vault	1.0
F5	Burning stETH shares on vault	1.0
F6	Rebalancing vault with ether or shares by owner	1.0
F7	Force rebalancing vault at VaultHub	1.0

ID	Flow	Added
F8	Report submission to LazyOracle with quarantine handling	1.0
F9	stETH value changes	1.0
F10	ETH deposit to pool with STV token minting	1.0
F11	Withdrawal request creation for STV tokens	1.0
F12	Finalization of withdrawal requests with checkpoint creation	1.0
F13	Setting finalization gas cost coverage	1.0
F14	Claiming finalized withdrawals	1.0
F15	Minting stETH shares for pool users	1.0
F16	Burning stETH shares or wstETH for pool users	1.0
F17	Force rebalancing user positions	1.0
F18	Force rebalancing with loss socialization for undercollateralized users	1.0
F19	Rebalancing unassigned liability	1.0
F20	Syncing vault parameters from VaultHub	1.0
F21	ETH deposit through GGV strategy with wstETH minting	1.0
F22	Withdrawal request through strategy	1.0
F23	Solving on-chain withdrawals from GGV vault	1.0
F24	Burning wstETH through strategy	1.0

Table 4. Wake fuzzing flows

The following table lists the invariants checked after each flow.

ID	Invariant	Added	Status
IV1	Transactions do not revert except where explicitly expected	1.0	Success

ID	Invariant	Added	Status
IV2	Event parameter values match expected calculations	1.0	**Fail** (L1)
IV3	Native ETH balances match tracked state for all accounts	1.0	Success
IV4	stETH share balances tracked in Python are never negative	1.0	Success
IV5	stETH and wstETH balances match on-chain state	1.0	Success
IV6	Vault total value equals report total value adjusted by in/out delta	1.0	Success
IV7	Latest report data matches tracked vault state	1.0	Success
IV8	LazyOracle latest timestamp matches tracked state	1.0	Success
IV9	Vault record in/out delta cache matches tracked state	1.0	Success
IV10	Report freshness status is correctly determined	1.0	Success
IV11	Vault obligations match tracked state	1.0	Success
IV12	Max lockable value at VaultHub is correctly calculated	1.0	Success
IV13	Settled growth matches dashboard state	1.0	Success
IV14	Accrued fee is correctly calculated based on growth and fee rate	1.0	Success
IV15	Max lockable value accounts for accrued fees	1.0	Success
IV16	Liability shares match dashboard state	1.0	Success

ID	Invariant	Added	Status
IV17	Dashboard minting capacity is correctly calculated with reserve ratio	1.0	Success
IV18	Vault health status is correctly determined	1.0	Success
IV19	Health shortfall shares are correctly calculated	1.0	Success
IV20	Native balance for withdrawal queue and staking vault entities match	1.0	Success
IV21	Pool token balances match tracked state for all accounts	1.0	Success
IV22	Asset values derived from pool balances are correct	1.0	Success
IV23	Withdrawal request counts match queue state	1.0	Success
IV24	Gas coverage setting matches tracked state	1.0	Success
IV25	Withdrawal request status matches on-chain state	1.0	Success
IV26	Pool liability shares and unassigned liability are zero for StvPool	1.0	Success
IV27	Pool total supply matches tracked state	1.0	Success
IV28	Pool total nominal assets are correctly calculated	1.0	Success
IV29	Pool total assets equals nominal assets minus unassigned liability	1.0	Success
IV30	Unfinalized withdrawal amounts match tracked state	1.0	Success
IV31	Queue state (last finalized and last request IDs) is consistent	1.0	Success

ID	Invariant	Added	Status
IV32	stETH balance for pool matches on-chain state	1.0	Success
IV33	User minted stETH shares match pool mintedStethSharesOf	1.0	Success
IV34	User health status is correctly determined based on collateralization	1.0	Success
IV35	Preview force rebalance returns correct values for breach detection	1.0	Success
IV36	Preview function results match actual execution values	1.0	**Fail** (L2)
IV37	User minting stETH capability is correctly calculated	1.0	Success
IV38	Assets to lock for stETH shares calculation is correct	1.0	Success
IV39	Reserve ratio BP matches tracked state	1.0	Success
IV40	Forced rebalance threshold BP matches tracked state	1.0	Success
IV41	Total minted stETH shares match sum of user shares	1.0	Success
IV42	Total exceeding minted stETH shares are correctly tracked	1.0	Success
IV43	Pool total liability shares match sum of user minted shares	1.0	Success
IV44	Pool liability shares include strategy call forwarder accounts	1.0	Success
IV45	wstETH balances for strategy-related accounts match on-chain state	1.0	Success

ID	Invariant	Added	Status
IV46	Native balances for strategy-related accounts match tracked state	1.0	Success
IV47	GGV vault balances for call forwarders match tracked state	1.0	Success

Table 5. Wake fuzzing invariants

B.2. Detectors

```
wake detect unused-error

[INFO][HIGH] Unused error [unused-error]
  26     error InsufficientMintedShares();
  27     error InsufficientStv();
  28     error ZeroArgument();
❱ 29     error ArraysLengthMismatch(uint256 firstArrayLength, uint256 secondArrayLength);
  30     error CannotRebalanceWithdrawalQueue();
  31     error UndercollateralizedAccount();
  32     error CollateralizedAccount();
src/StvStETHPool.sol
```

Figure 1. Unused error in Factory contract

```
wake detect unused-error

[INFO][HIGH] Unused error [unused-error]
  26     error InsufficientMintedShares();
  27     error InsufficientStv();
  28     error ZeroArgument();
❱ 29     error ArraysLengthMismatch(uint256 firstArrayLength, uint256 secondArrayLength);
  30     error CannotRebalanceWithdrawalQueue();
  31     error UndercollateralizedAccount();
  32     error CollateralizedAccount();
src/StvStETHPool.sol
```

Figure 2. Unused error in StvStETHPool contract

```
wake detect unused-contract

[INFO][HIGH] Interface not used [unused-contract]
  3
  4 import {IBoringOnChainQueue} from "./IBoringOnChainQueue.sol";
  5
❱ 6 interface IBoringSolver {
  7     enum SolveType {
  8         BORING_REDEEM, // Fill multiple user requests with a single transaction.
  9         BORING_REDEEM_MINT // Fill multiple user requests to redeem shares and mint new shares.
src/interfaces/ggv/IBoringSolver.sol
```

Figure 3. Unused IBoringSolver interface

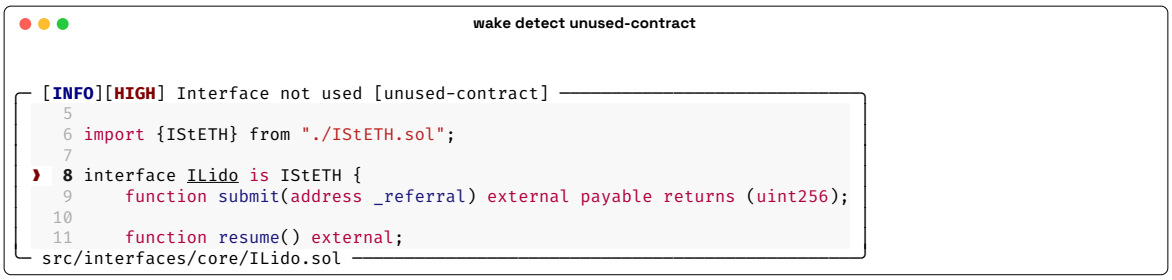


Figure 4. Unused ILido interface

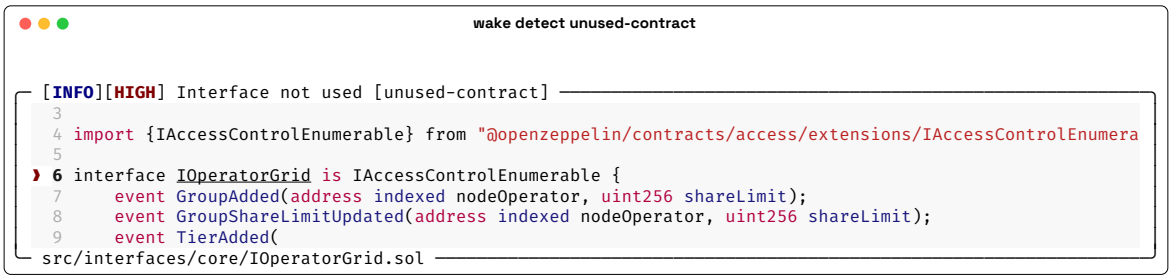


Figure 5. Unused IOperatorGrid interface

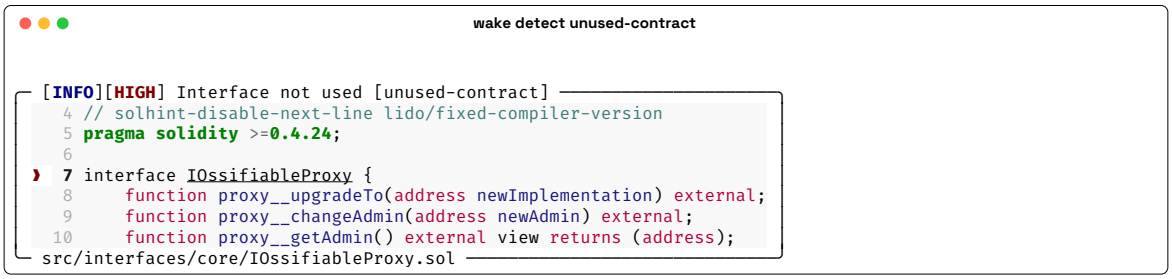


Figure 6. Unused IOssifiableProxy interface

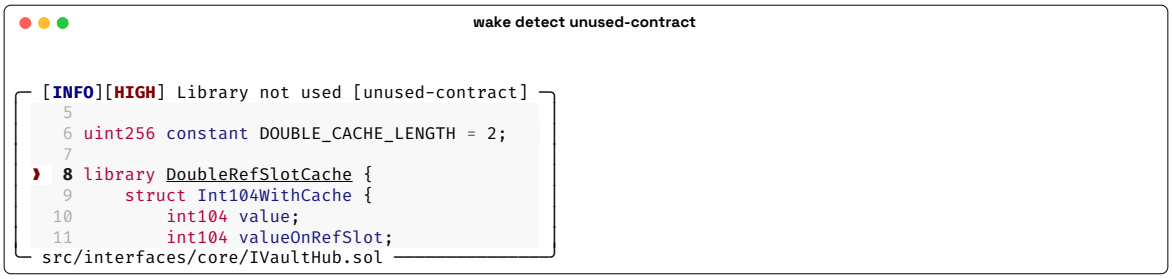


Figure 7. Unused IVaultHub library

Appendix C: Wake Arena Findings

This section lists vulnerabilities identified by Wake Arena, an LLM-powered audit tool used for AI-assisted vulnerability discovery during the audit. Wake Arena leverages large language models to understand code context and reason about complex contract behavior, complementing manual review.

C.1. Discovered Findings

The following table contains true-positive findings identified by Wake Arena. These findings are included regardless of whether they were also discovered independently by auditors during manual review.

Finding title	Severity	Reported	Discoverer
I7: Role constants read from Dashboard implementation instead of proxy	Info	1.0	Wake Arena
I8: createPoolFinish forwards full `msg.value` instead of exact connect deposit potentially causing loss of overpaid ETH	Info	1.0	Wake Arena
I9: calculateCurrentStethShareRate documentation contradicts implementation precision	Info	1.0	Wake Arena

Table 6. Table of findings identified by Wake Arena

ackee
blockchain security

Thank You

Ackee Blockchain a.s.

Rohanske nabrezi 717/4
186 00 Prague
Czech Republic

hello@ackee.xyz