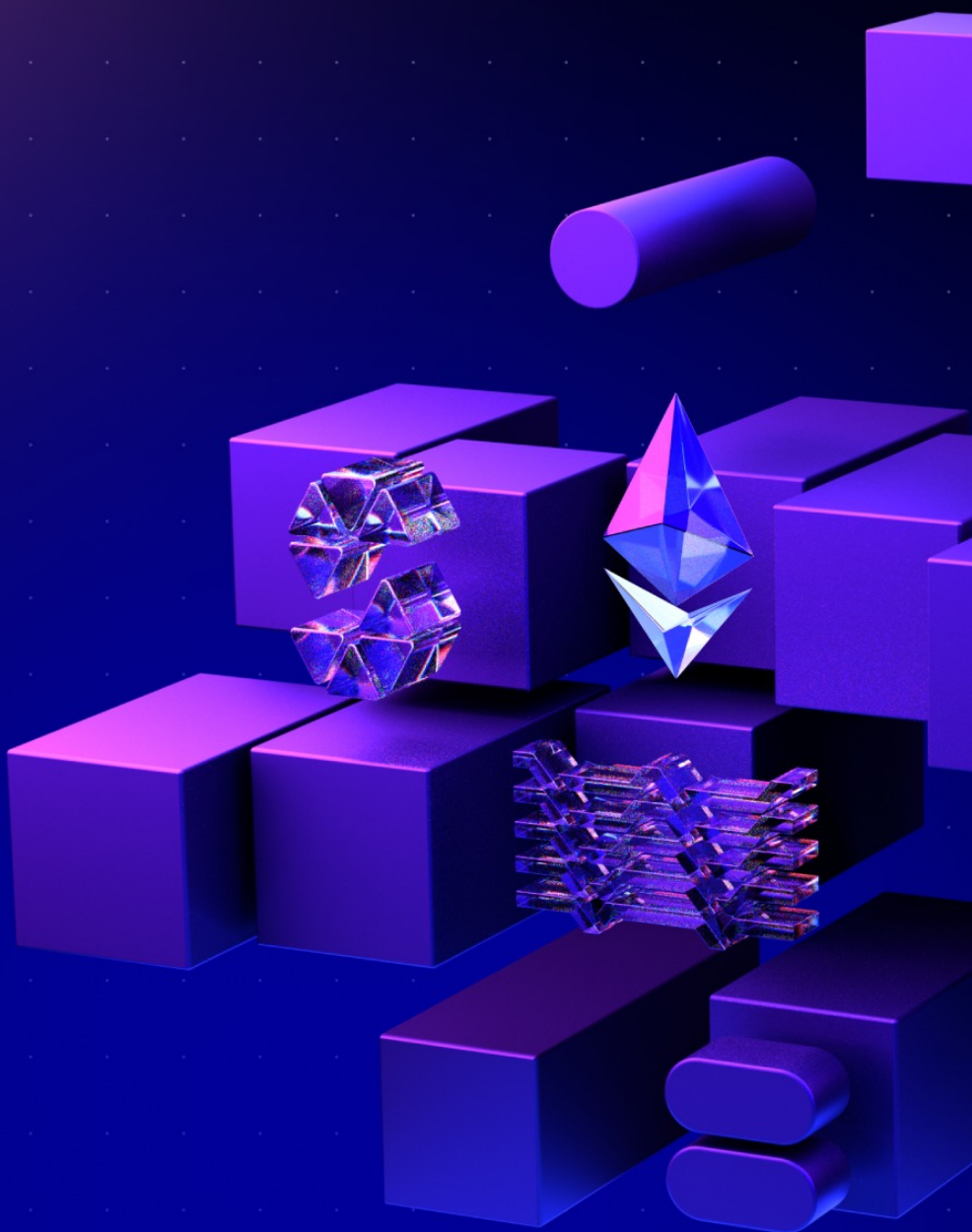
ackee
blockchain security

Lido

Stonks 2.0

12.12.2025



Audit Report

Contents

1. Document Revisions

1.0-draft	Draft Report	02.12.2025
1.0	Final Report	08.12.2025
1.1	Final Report	12.12.2025

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-AI assisted vulnerability discovery**
   As the last step, the scope is checked against Wake AI, 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
Michal Převrátil	Lead Auditor
David Lapuník	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

Lido Stonks 2.0 is the next major revision of Stonks, an ERC-1271 based solution for token exchanges using the CoW protocol.

The project uses Chainlink data feeds to obtain real-time expected output amounts for token swaps and manages order fulfillment based on the configured parameters.

The major upgrade introduces any-to-any pricing either through a common price denomination or through bridging over the ETH/USD price. Additionally, emergency mechanisms are improved and partial-fill functionality is added.

Revision 1.0

Lido engaged Ackee Blockchain Security to perform a security review of Lido Stonks 2.0 with a total time donation of 15 engineering days in a period between November 17 and December 2, 2025, with Michal Převrátil as the lead auditor. 4 engineering days were dedicated to manually-guided fuzzing using the Wake testing framework.

The audit was initially performed on the commit `0c17c1d`[1] in the stonks repository, but due to the M1 finding preventing testing of a major part of the protocol, the commit was changed 2 days after the start of the audit to `325bfa6`[2].

The scope of the audit was all Solidity contracts in the repository except for the `contracts/stubs` and `contracts/test` directories.

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, especially focusing on

the following areas:

- token accounting is performed correctly, yielding expected token outputs;
- reentrancy and front-running attacks are prevented;
- emergency functionalities work as intended;
- token accounting cannot be manipulated through donation attacks;
- Chainlink integration is correct; and
- there are no common issues, including data validation.

At the end of the review, we engaged the Wake AI tool, which discovered the following issues: M1, L2, W5, I9.

Our review resulted in 17 findings, ranging from Info to Medium severity. The most severe one M1, present in the original commit revision, prevents swapping of pairs tokens with mixed ETH/USD denominations. This poses a denial of service for users trying to swap such pairs.

Next most severe findings L1 and L2 introduce a temporary blockage for the execution of orders with strict parameters and a risk of unpausing existing orders after the `killSwitch` is engaged, respectively.

The codebase overall shows good quality with excellent documentation. The code contains multiple emergency mechanisms and properly handles atypical tokens, such as rebasing tokens. Most of the findings in the latest commit revision present minor oversights or additional suggestions for improvements.

Ackee Blockchain Security recommends Lido:

- ensure that the `AGENT` address can receive ERC-721 and ERC-1155 tokens through the `AssetRecoverer` contract;
- deploy the `OracleRouter` contract with large enough `PRICE_DECIMALS`

parameter (such as 18) to avoid precision loss;

- note that ERC-20 tokens without the `decimals()` function implemented are not supported; and
- review and address the reported findings.

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 #25 with the fixes and additional comments regarding the code changes.

The review was performed between December 11 and December 12, 2025 on the commit `0669c4a`[3].

All of the findings from the previous revision were fixed.

No new findings were discovered.

[1] full commit hash: `0c17c1ddd4a8e67e6ecf0f4838c731c84cb0352b`, link to commit

[2] full commit hash: `325bfa6b87d081ad4d1369bb92c0b6de8e0af89f`, link to commit

[3] full commit hash: `0669c4a50660912785e18861d9ef8108a86a3552`, 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	1	2	5	9	**17**

Table 2. Findings Count by Severity

Findings in detail:

Finding title	Severity	Reported	Status
M1: Flawed ETH/USD routing	Medium	1.0	Fixed
L1: Order rounding errors	Low	1.0	Fixed
L2: Stonks can be unpaused after `killSwitch`	Low	1.0	Fixed
W1: `AmountConverter` in the `Stonks` contract may not allow `TOKEN_FROM` and `TOKEN_TO`	Warning	1.0	Fixed
W2: Token decimals used as configured flag	Warning	1.0	Fixed
W3: Missing ETH/USD feed configuration check	Warning	1.0	Fixed

Finding title	Severity	Reported	Status
W4: `PRICE_UNIT` and `PRICE_SCALE` discrepancy	Warning	1.0	Fixed
W5: Misleading use of errors	Warning	1.0	Fixed
I1: Inconsistency in allowing manager to be zero address	Info	1.0	Fixed
I2: The `KillEngaged` should not be emitted if the the `Stonks` contract was already killed	Info	1.0	Fixed
I3: `assertQuotable` inefficiency	Info	1.0	Fixed
I4: Missing public getters	Info	1.0	Fixed
I5: Redundant `MAX_DECIMALS` checks	Info	1.0	Fixed
I6: Missing event emissions	Info	1.0	Fixed
I7: Inefficient variable packing	Info	1.0	Fixed
I8: Missing `TokenNotConfigured` check	Info	1.0	Fixed
I9: Misleading event parameter name	Info	1.0	Fixed

Table 3. Table of Findings

Report Revision 1.0

Revision Team

Member's Name	Position
Michal Převrátil	Lead Auditor
David Lapuník	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

System Overview

Lido Stonks 2.0 is an MEV-resistant token exchange solution built on top of the CoW Swap protocol.

The project uses Chainlink data feeds to estimate the expected output amounts for token swaps and manages fulfillment of orders based on the real-time data and configured parameters.

Stonks is designed to allow any-to-any token swaps through dedicated `AmountConverter` contracts, defining the allowed token pairs and the common price denomination used for the price calculations.

The central component is the `OracleRouter` contract, which is responsible for caching the Chainlink aggregator metadata, resynchronizing the price feeds upon changes, and providing the price data for the token swaps.

Each token pair has a dedicated deployed instance of the `Stonks` contract, which is designed for placing new orders, effectively creating new `Order` contracts. `Order` contracts then manage the placed limit order through the ERC-1271 signature scheme.

Trust Model

The project trusts Chainlink data feeds to obtain real-time price data for token swaps.

Deployment of new `AmountConverter` and `Stonks` instances is permissionless but sets the ownership of the contracts to the predefined `AGENT` address. Recovery of funds from expired orders is also permissionless. All other operations are restricted to the `AGENT` address or an optionally set manager address.

The `AGENT` address is expected to be implementing ERC-721 and ERC-1155 token callbacks, or to be EOA or an upgradeable contract in order to be able to recover lost tokens through the `AssetRecoverer` contract.

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.

The full source code of all fuzz tests is available at https://github.com/Ackee-Blockchain/tests-lido-stonks.

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

M1: Flawed ETH/USD routing

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	OracleRouter	Type:	Denial of service

Description

The `AmountConverter` contract defines the `getExpectedOut` function to estimate the expected amount of received tokens for a given amount of sold tokens. The function first chooses the common quote denomination based on the `USE_ETH_ANCHOR` deployment parameter. Both token prices estimation is then delegated to the `OracleRouter.getPricesAndDecimals` function.

Listing 1. Excerpt from AmountConverter.getExpectedOut

```
if (USE_ETH_ANCHOR) {
    quote = IOracleRouter.QuoteDenomination.ETH;
} else {
    quote = IOracleRouter.QuoteDenomination.USD;
}

(priceFrom, priceTo, decimalsOfSellToken, decimalsOfBuyToken) = ORACLE_ROUTER
    .getPricesAndDecimals(tokenFrom_, tokenTo_, quote);
```

However, due to a flawed logic, the `getPricesAndDecimals` function reverts when either of the tokens configured denomation does not match the requested common denomination.

Listing 2. Excerpt from OracleRouter.getPricesAndDecimals

```
if (baseConfig.primaryQuote != quote_) {
    if (quote_ == IOracleRouter.QuoteDenomination.USD) {
        revert TokenNotUsdQuoted(baseToken_);
    } else {
        revert TokenNotEthQuoted(baseToken_);
    }
```

```
7 }
```

Listing 3. Excerpt from OracleRouter.getPricesAndDecimals

```
1 if (quoteConfig.primaryQuote != quote_) {
2     if (quote_ == IOracleRouter.QuoteDenomination.USD) {
3         revert TokenNotUsdQuoted(quoteToken_);
4     } else {
5         revert TokenNotEthQuoted(quoteToken_);
6     }
7 }
```

Exploit scenario

A user wants to sell 100 stETH tokens, configured with USD denomination, for LDO tokens, configured with ETH denomination. Due to the flawed logic, creation of the order reverts which results in a denial of service for the user.

Recommendation

Fix the flawed logic in the `OracleRouter.getPricesAndDecimals` function to use the ETH/USD bridge in the cases where both tokens are configured with different denominations.

Fix 1.1

The commit introducing the flaw was reverted. The original code implements the ETH/USD bridge usage correctly.

Go back to Findings Summary

L1: Order rounding errors

Low severity issue

Impact:	Medium	Likelihood:	Low
Target:	Order	Type:	Arithmetics

Description

The `isValidSignature` function in the `Order` contract confirms or rejects order swaps under the current time and market conditions. The function first fetches the current sell token balance and calculates the estimated buy token amount. The ratio between the two values is used as the current execution price. The contract additionally holds the original buy amount and sell amount of the order. The ratio between the original amounts is the original price of the order.

Listing 4. Excerpt from Order.isValidSignature

```
226 uint256 availableBalance = IERC20(tokenFrom).balanceOf(address(this));
```

Listing 5. Excerpt from Order.isValidSignature

```
235 uint256 basisSellAmount = allowPartialFill
236     ? (availableBalance < sellAmount ? availableBalance : sellAmount)
237     : sellAmount;
```

Listing 6. Excerpt from Order.isValidSignature

```
243 uint256 currentEstimatedBuyAmount =
    stonksContract.estimateTradeOutput(basisSellAmount);
```

Listing 7. Excerpt from Order.isValidSignature

```
250 uint256 baselineBuyAmount = Math.mulDiv(buyAmount, basisSellAmount,
    sellAmount);
```

```
251
252 // Fast path: exact amount match avoids rounding issues in price ratio
    comparison
253 if (currentEstimatedBuyAmount == baselineBuyAmount) {
254     return ERC1271_MAGIC_VALUE;
255 }
256
257 // Compute prices scaled to 1e18 for ratio comparison
258 uint256 originalLimitPrice = Math.mulDiv(buyAmount, PRICE_SCALE,
    sellAmount);
259 uint256 currentExecutionPrice = Math.mulDiv(
260     currentEstimatedBuyAmount,
261     PRICE_SCALE,
262     basisSellAmount
263 );
```

By comparison of the two prices, one of the following scenarios can occur:

- The prices are equal — the order swap is valid.
- The current price is higher than the original price (the market moved in favor of the order) — the validity of the order swap is determined by the maximum improvement parameter in basis points.
- The current price is lower than the original price (the market moved against the order) — the validity of the order swap is determined by the price tolerance parameter in basis points.

Both maximum improvement and price tolerance parameters can be set to zero, i.e. strict mode. No price changes are allowed in this case.

Due to rounding errors in the `isValidSignature` calculations, the current execution price value may be evaluated as a different value than the original price by changing the current balance of the sell token. This may happen one of the following cases:

- The order was partially filled; i.e. a partial amount of the sell token was transferred from the contract.

- A malicious user donated a small amount of the sell token to the contract.

Exploit scenario

A partially fillable order to sell 214,576 YFI tokens for stETH tokens in the strict tolerance mode is created. The estimated output amount is 276,559 stETH tokens. The raw execution price is evaluated as 1288862687346208336.

The order is first partially filled with an exact trade output amount match, selling 55,936 YFI tokens.

However, the subsequent `isValidSignature` call reverts with the `PriceShortfallExceedsTolerance(minAcceptableBuyAmount=204465, actualBuyAmount=204464)` error. This is because the remaining sell token amount is 158,640 YFI tokens and the new estimated baseline output amount is 204,464 stETH tokens. This results in a different raw execution price of 1288855269793242561.

Due to the rounding errors, the order execution may become temporarily blocked after a partial fill or a small donation of the sell token. The described scenario is most likely to happen when either the maximum improvement or price tolerance parameters are set to zero, but it is also reproducible when the parameters are set close to zero.

Recommendation

Consider allowing off-by-two rounding errors when comparing the original buy amount with the current estimated buy amount.

Fix 1.1

The finding was fixed by allowing off-by-two rounding errors when comparing the original buy amount with the current estimated buy amount. This is also true when comparing the minimum acceptable buy amount (based on the configured tolerance parameter) with the current estimated buy

amount.

Go back to Findings Summary

L2: Stonks can be unpaused after `killSwitch`

Low severity issue

Impact:	Medium	Likelihood:	Low
Target:	Stonks	Type:	Logic error

Description

The `Stonks` contract implements multiple emergency mechanisms to stop the contract and orders created by it:

- `pauseCreation` to pause creation of new orders;
- `pauseSignatures` to pause filling of existing orders created by the contract;
- `killSwitch` to irreversibly pause the contract and orders created by it.

However, both `unpauseCreation` and `unpauseSignatures` functions can still be called after the `killSwitch` is engaged.

Listing 8. Excerpt from Stonks

```
302 function unpauseCreation() external onlyAgentOrManager {
303     _unpause();
304 }
```

Listing 9. Excerpt from Stonks

```
322 function unpauseSignatures() external onlyAgentOrManager {
323     if (!_signaturesPaused) {
324         return;
325     }
326
327     _signaturesPaused = false;
328
329     emit SignaturesUnpaused(msg.sender);
330 }
```

Exploit scenario

Calling `unpauseCreation` after `killSwitch` does not pose a significant risk as both order creation functions `placeOrder` and `placeOrderWithAmount` are protected by the `notKilled` modifier. The only impact is an inconsistency in the contract state.

Calling `unpauseSignatures` after `killSwitch` introduces a higher risk as it allows filling of orders created by the contract again. This is conflict with the `killSwitch` purpose and can lead to a situation where the contract is not fully stopped.

Recommendation

Add the `whenNotKilled` modifier to the `unpauseCreation` and `unpauseSignatures` functions to prevent them from being called after the `killSwitch` is engaged.

Fix 1.1

The finding was fixed by following the recommendation.

Go back to Findings Summary

W1: `AmountConverter` in the `Stonks` contract may not allow `TOKEN_FROM` and `TOKEN_TO`

Impact:	Warning	Likelihood:	N/A
Target:	Stonks	Type:	Code quality

Description

In the `Stonks` constructor, there is no check whether `TOKEN_FROM` and `TOKEN_TO` are allowed in `AmountConverter`.

Listing 10. Excerpt from Stonks.constructor

```
170 AMOUNT_CONVERTER = initParams_.amountConverter;
171 TOKEN_FROM = initParams_.tokenFrom;
172 TOKEN_TO = initParams_.tokenTo;
```

If they are not allowed, this will cause `getExpectedOut` on `AmountConverter` to revert, thus `estimateTradeOutput` will revert on `Stonks` contract and it will not be possible to place orders.

Listing 11. Excerpt from AmountConverter.getExpectedOut

```
152 if (!allowedTokensToSell[tokenFrom_]) {
153     revert SellTokenNotAllowed(tokenFrom_);
154 }
155
156 if (!allowedTokensToBuy[tokenTo_]) {
157     revert BuyTokenNotAllowed(tokenTo_);
158 }
```

Recommendation

Ensure the `AmountConverter` allows `TOKEN_FROM` and `TOKEN_TO` during construction.

Fix 1.1

The `Stonks` contract now checks whether `TOKEN_FROM` and `TOKEN_TO` are allowed in `AmountConverter` during construction.

Go back to Findings Summary

W2: Token decimals used as configured flag

Impact:	Warning	Likelihood:	N/A
Target:	OracleRouter	Type:	Logic error

Description

The `OracleRouter` contract explicitly forbids ERC-20 tokens with zero decimals and uses the `tokenDecimals` field as a flag whether the token is configured.

Listing 12. Excerpt from OracleRouter._setTokenFeed

```
623 if (erc20Decimals == 0 || erc20Decimals > MAX_DECIMALS) {
624     revert InvalidTokenDecimals();
625 }
```

Listing 13. Excerpt from OracleRouter.setTokenActive

```
216 if (config.tokenDecimals == 0) {
217     revert TokenNotConfigured(token_);
218 }
```

Note that the similar `tokenDecimals` zero checks are used in multiple places in the contract.

As per ERC-20, tokens with zero decimals are technically valid and can be supported in Stonks with adequate modifications.

Recommendation

Allow ERC-20 tokens with zero decimals to be configured. Use the `primaryFeed.aggregator` field instead of the `tokenDecimals` field as a flag whether the token is configured.

Fix 1.1

The finding was fixed by using the `primaryFeed.aggregator` field instead of the `tokenDecimals` field as a flag whether the token is configured.

Tokens with zero decimals were intentionally decided as unsupported in the Stonks protocol.

Go back to Findings Summary

W3: Missing ETH/USD feed configuration check

Impact:	Warning	Likelihood:	N/A
Target:	OracleRouter	Type:	Data validation

Description

The `OracleRouter.syncEthUsdBridge` function can be used to fetch the latest ETH/USD feed metadata while preserving the current maximum staleness threshold. The function can be called even if the ETH/USD feed was not configured yet. However, since the maximum staleness threshold is not updated, any calculations using the ETH/USD feed will revert due to the stale error. This may cause confusion if the `syncEthUsdBridge` function is called before the `setEthUsdBridge` function by mistake.

Listing 14. Excerpt from OracleRouter

```
159 function syncEthUsdBridge() external onlyAgentOrManager {
160     _updateEthUsdBridge(ethUsdBridge.maxStalenessSeconds);
161 }
```

Recommendation

Check if the ETH/USD feed is configured and revert otherwise in the `syncEthUsdBridge` function.

Fix 1.1

The finding was fixed by introducing the `ethUsdBridge.aggregator == address(0)` check in the `syncEthUsdBridge` function.

Go back to Findings Summary

W4: `PRICE_UNIT` and `PRICE_SCALE` discrepancy

Impact:	Warning	Likelihood:	N/A
Target:	Order	Type:	Configuration

Description

The `Order` contract defines the `PRICE_SCALE` constant as a precision multiplier for price calculations.

Listing 15. Excerpt from Order

```
46 /// @notice Price scaling factor for ratio calculations (1e18 matches router
   precision).
47 uint256 private constant PRICE_SCALE = 1e18;
```

As the documentation states, the `PRICE_SCALE` constant should be equal to the `PRICE_UNIT` value defined in the `OracleRouter` contract. However, the value of `PRICE_UNIT` is configurable by the deployer of the `OracleRouter` contract.

Recommendation

Either:

- ensure that the `OracleRouter` contract is deployed with `PRICE_DECIMALS = 18`; or
- pass the `PRICE_UNIT` value from the `OracleRouter` contract to the `Order` contract as a parameter; or
- update the in-code documentation.

Fix 1.1

The finding was resolved by fixing the `PRICE_DECIMALS` value to 18 in the `OracleRouter` contract.

Go back to Findings Summary

W5: Misleading use of errors

Impact:	Warning	Likelihood:	N/A
Target:	AmountConverter, Stonks	Type:	Code quality

Description

The `AmountConverter` contract defines the `AmountFromTooLarge` error to prevent the `amountFrom_` parameter in the `getExpectedOut` function from overflowing the `uint128` type.

Listing 16. Excerpt from AmountConverter.getExpectedOut

```
148 if (amountFrom_ > type(uint128).max) {
149     revert AmountFromTooLarge(amountFrom_);
150 }
```

However, the same error definition is later used in the same function when scaling the `amountFrom_` parameter to account for the decimal difference between the tokens.

Listing 17. Excerpt from AmountConverter.getExpectedOut

```
222 // Scale the input first to avoid overflow on multiplication by 10**diff.
223 uint256 pow10 = 10 ** decimalsDiff;
224 uint256 maxAmountFromBeforeScale = type(uint256).max / pow10;
225
226 if (amountFrom_ > maxAmountFromBeforeScale) {
227     revert AmountFromTooLarge(amountFrom_);
228 }
```

Although the error name still reflects the in-code usage, it can be misleading to the function caller as both error usages are unrelated and applied in different contexts.

Additionally, the `Stonks` contract uses the `MarginOverflowsAllowedLimit` error in validation of the `maxImprovementInBasisPoints_` parameter. This is misleading

because the error is defined for the `marginInBasisPoints_` parameter.

Listing 18. Excerpt from Stonks._validateBps

```
489 if (
490     maxImprovementInBasisPoints_ != type(uint256).max &&
491     maxImprovementInBasisPoints_ > BASIS_POINTS_PARAMETERS_LIMIT
492 ) {
493     revert MarginOverflowsAllowedLimit(
494         BASIS_POINTS_PARAMETERS_LIMIT,
495         maxImprovementInBasisPoints_
496     );
497 }
```

Recommendation

Consider introducing a new error definition in the `AmountConverter` contract for the decimal scaling error.

Introduce a new error definition in the `Stonks` contract for the `maxImprovementInBasisPoints_` parameter validation.

Fix 1.1

New error definitions were introduced in the `AmountConverter` and `Stonks` contracts by following the recommendation.

The original `AmountFromTooLarge` error and the corresponding logic were removed.

Go back to Findings Summary

I1: Inconsistency in allowing manager to be zero address

Impact:	Info	Likelihood:	N/A
Target:	Ownable, Stonks	Type:	Data validation

Description

The `Stonks` contract inherits the `Ownable` contract. In the `Stonks` constructor, it is not possible to set the manager to the zero address, as the `_validateAddresses` function reverts in this case.

Listing 19. Excerpt from Stonks._validateAddresses

```
431 if (manager_ == address(0)) {
432     revert InvalidManagerAddress(manager_);
433 }
```

However, the `Ownable` contract contains a `setManager` function that allows the manager to be set to the zero address.

Listing 20. Excerpt from Ownable

```
71 function setManager(address manager_) external onlyAgent {
72     manager = manager_;
73
74     emit ManagerSet(manager_);
75 }
```

This creates an inconsistency where during construction the manager cannot be the zero address, but after construction it is possible to set the manager to the zero address.

Recommendation

Ensure this inconsistency is intended, or resolve it by either reverting on zero

address in `setManager` or allowing zero address in the constructor when manager is set to zero address.

Fix 1.1

The `manager` address in the `Stonks` contract is no longer required to be set to a non-zero address in the constructor.

Go back to Findings Summary

I2: The `KillEngaged` should not be emitted if the the `Stonks` contract was already killed

Impact:	Info	Likelihood:	N/A
Target:	Stonks	Type:	Code quality

Description

When the `killSwitch` function is called multiple times, the `KillEngaged` event is emitted every time. In contrast, pausing creation or pausing signatures emits events only when a state change occurs.

Listing 21. Excerpt from Stonks

```
335 function killSwitch() external onlyAgentOrManager {
336     // Set signatures paused if not already, emit telemetry when it changes
337     if (!_signaturesPaused) {
338         _signaturesPaused = true;
339 
340         emit SignaturesPaused(msg.sender);
341     }
342     // Pause creation if not already paused
343     if (!paused()) {
344         _pause();
345     }
346 
347     // Mark killed (irreversible)
348     if (!_killed) {
349         _killed = true;
350     }
351 
352     emit KillEngaged(msg.sender);
353 }
```

Recommendation

Ensure the event is emitted only on state change, meaning the first time `killSwitch` is called.

Fix 1.1

The finding was fixed by following the recommendation.

Go back to Findings Summary

I3: `assertQuotable` inefficiency

Impact:	Info	Likelihood:	N/A
Target:	Order	Type:	Gas optimization

Description

The `Stonks.assertQuotable` function is used as a fail-fast mechanism to prevent placing orders with misconfigured tokens or oracles.

Listing 22. Excerpt from Order.initialize

```
146 // Fail-fast if either side lacks a valid oracle route (prevents stranded
    approvals/funds).
147 stonksContract.assertQuotable();
```

However, the `assertQuotable` function call is followed by a `Stonks.estimateTradeOutput` function call that performs almost the same checks internally without reusing already fetched data. This makes the `assertQuotable` call redundant and inefficient.

Additionally, the `assertQuotable` call does not serve as a fail-fast mechanism in the edge case scenario where both buy and sell tokens are configured as USD-denominated, but the linked `AmountConverter` contract instance has `USE_ETH_ANCHOR` set to `true`. In this case, the `assertQuotable` logic does not verify the ETH/USD feed configuration, but `estimateTradeOutput` logic uses the ETH/USD feed and reverts if the feed is not configured correctly.

Recommendation

Remove the redundant `assertQuotable` call.

Fix 1.1

The finding was fixed by following the recommendation.

Go back to Findings Summary

I4: Missing public getters

Impact:	Info	Likelihood:	N/A
Target:	Order	Type:	Code quality

Description

The `Order` contract defines multiple state variables. Most of them can be accessed either through the `getOrderDetails` function or directly through public getters. However, the `allowPartialFill` and `cancelled` state variables are missing public getters.

Listing 23. Excerpt from Order

```
65 /// @notice Whether this order allows partial fills (cached from Stonks to
   avoid external calls).
66 bool private allowPartialFill;
67 /// @notice Order cancellation flag.
68 bool private cancelled;
```

While the value of `allowPartialFill` can be derived from the `Stonks` contract, the `cancelled` state variable is not accessible publicly and can only be fetched through low-level storage deconstruction.

Recommendation

Consider marking both state variables as public.

Fix 1.1

The finding was fixed by marking both state variables as public.

Go back to Findings Summary

I5: Redundant `MAX_DECIMALS` checks

Impact:	Info	Likelihood:	N/A
Target:	AmountConverter, OracleRouter	Type:	Code quality

Description

The `getExpectedOut` function in the `AmountConverter` contract computes the difference between the sell token and buy token decimals and compares it to the `ROUTER_MAX_DECIMALS` parameter.

Listing 24. Excerpt from AmountConverter.getExpectedOut

```
195 if (sellHasMoreOrEqualDecimals) {
196     decimalsDiff = decimalsOfSellToken - decimalsOfBuyToken;
197 } else {
198     decimalsDiff = decimalsOfBuyToken - decimalsOfSellToken;
199 }
200
201 if (decimalsDiff > ROUTER_MAX_DECIMALS) {
202     revert InvalidDecimalsDifference(decimalsDiff);
203 }
```

However, the comparison is redundant because the `OracleRouter` contract (used during the `getExpectedOut` execution) already verifies that each configured token has at most `MAX_DECIMALS` decimals.

Listing 25. Excerpt from OracleRouter._setTokenFeed

```
623 if (erc20Decimals == 0 || erc20Decimals > MAX_DECIMALS) {
624     revert InvalidTokenDecimals();
625 }
```

Similarly, the `upDiff` and `downDiff` values in the `OracleRouter._computeScaleFactors` function do not need to be checked against the `MAX_DECIMALS` parameter.

Listing 26. Excerpt from OracleRouter._computeScaleFactors

```
723 uint8 upDiff = PRICE_DECIMALS - feedDecimals_;
724
725 if (upDiff > MAX_DECIMALS) {
726     revert InvalidAggregatorDecimals();
727 }
```

Listing 27. Excerpt from OracleRouter._computeScaleFactors

```
734 uint8 downDiff = feedDecimals_ - PRICE_DECIMALS;
735
736 if (downDiff > MAX_DECIMALS) {
737     revert InvalidAggregatorDecimals();
738 }
```

This is because both `PRICE_DECIMALS` and `feedDecimals_` values are already earlier checked against the `MAX_DECIMALS` parameter.

Listing 28. Excerpt from OracleRouter.constructor

```
125 if (unitDecimals_ == 0 || unitDecimals_ > MAX_DECIMALS) {
126     revert InvalidUnitDecimals();
127 }
```

Listing 29. Excerpt from OracleRouter._resolveFeedAndScale

```
696 decimals = registry.decimals(baseToken_, quoteToken_);
697
698 if (decimals == 0 || decimals > MAX_DECIMALS) {
699     revert InvalidAggregatorDecimals();
700 }
```

Recommendation

Consider removing the redundant checks in the `AmountConverter` and `OracleRouter` contracts.

Fix 1.1

All redundant checks were removed by following the recommendation.

Go back to Findings Summary

I6: Missing event emissions

Impact:	Info	Likelihood:	N/A
Target:	Stonks, Order	Type:	Logging

Description

The `Stonks` contract fails to emit events for the `maxImprovementInBasisPoints` and `allowPartialFill` constructor input parameters.

Additionally, the `Order` contract does not emit the `ManagerSet` event in the `initialize` function.

Both occurrences are inconsistencies with respect to the rest of the codebase.

Recommendation

Introduce new event definitions in the `Stonks` contract and add the missing event emissions to both contracts.

Fix 1.1

All of the missing event emissions were introduced by following the recommendation.

Go back to Findings Summary

I7: Inefficient variable packing

Impact:	Info	Likelihood:	N/A
Target:	OracleRouter	Type:	Gas optimization

Description

The `tokenConfig` mapping in the `OracleRouter` holds the configuration for each token.

Listing 30. Excerpt from OracleRouter

```
65 mapping(address => TokenConfig) public tokenConfig;
```

Each token configuration is stored in the `TokenConfig` struct that inlines the `FeedConfig` struct.

Listing 31. Excerpt from OracleRouter

```
41 struct FeedConfig {
42     address aggregator;
43     uint128 scaleNumerator;
44     uint128 scaleDenominator;
45     uint32 maxStalenessSeconds;
46     uint8 aggregatorDecimals;
47 }
48
49 struct TokenConfig {
50     IOracleRouter.QuoteDenomination primaryQuote;
51     FeedConfig primaryFeed;
52     uint8 tokenDecimals;
53     bool isActive;
54     uint32 ethUsdMaxStalenessOverrideSeconds; // 0 => use global bridge
   staleness
55 }
```

The fields of both structs are sorted inefficiently that they occupy more storage slots than necessary.

Recommendation

Consider restructuring both struct fields as follows:

```
struct FeedConfig {
    uint128 scaleNumerator;
    uint128 scaleDenominator;
    address aggregator;
    uint32 maxStalenessSeconds;
    uint8 aggregatorDecimals;
}

struct TokenConfig {
    FeedConfig primaryFeed;
    uint32 ethUsdMaxStalenessOverrideSeconds;
    uint8 tokenDecimals;
    IOracleRouter.QuoteDenomination primaryQuote;
    bool isActive;
}
```

Fix 1.1

The finding was fixed by following the recommendation.

Go back to Findings Summary

I8: Missing `TokenNotConfigured` check

Impact:	Info	Likelihood:	N/A
Target:	OracleRouter	Type:	Data validation

Description

The `setTokenEthUsdStalenessOverride` function in the `OracleRouter` contract allows to override the default staleness of the ETH/USD bridge for a given token. The function checks if the token address is non-zero, but fails to verify if the token is configured in the `tokenConfig` mapping.

Listing 32. Excerpt from OracleRouter

```
186 function setTokenEthUsdStalenessOverride(
187     address token_,
188     uint32 overrideSeconds_
189 ) external onlyAgentOrManager {
190     if (token_ == address(0)) {
191         revert InvalidTokenAddress(token_);
192     }
```

A later call to `setTokenFeed` overwrites any existing staleness override value with zero, which may lead to unexpected behavior.

Listing 33. Excerpt from OracleRouter._setTokenFeed

```
648 tokenConfig[token_] = TokenConfig({
649     primaryQuote: primaryQuote_,
650     primaryFeed: FeedConfig({
651         aggregator: aggregator,
652         maxStalenessSeconds: maxStalenessSeconds_,
653         aggregatorDecimals: feedDecimals,
654         scaleNumerator: scaleNumerator,
655         scaleDenominator: scaleDenominator
656     }),
657     tokenDecimals: erc20Decimals,
658     isActive: isActive_,
659     ethUsdMaxStalenessOverrideSeconds: 0
```

```
660 });
```

Recommendation

Replace the token zero address check with token-configured check.

Fix 1.1

The finding was fixed by introducing the `config.primaryFeed.aggregator == address(0)` check and reverting with the `TokenNotConfigured` error if the condition is met.

Go back to Findings Summary

I9: Misleading event parameter name

Impact:	Info	Likelihood:	N/A
Target:	AmountConverterFactory	Type:	Code quality

Description

The `AmountConverterFactory` contract defines the `AmountConverterDeployed` event with the `allowedStableTokensToBuy` parameter name.

Listing 34. Excerpt from AmountConverterFactory

```
19 event AmountConverterDeployed(
20     address indexed amountConverterAddress,
21     address oracleRouter,
22     address[] allowedTokensToSell,
23     address[] allowedStableTokensToBuy,
24     bool useEthAnchor
25 );
```

However, the protocol is designed to support buying any token, not just stable tokens. This discrepancy can lead to a confusion.

Recommendation

Change the parameter name to `allowedTokensToBuy`.

Fix 1.1

The finding was fixed by following the recommendation.

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

> Ackee Blockchain Security, Audit Report | Lido: Stonks 2.0, 12.12.2025.

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.

ID	Flow	Added
F1	Change of a Chainlink feed address and decimals	1.0
F2	Change of ETH/USD feed address and decimals	1.0
F3	Change of a Chainlink feed decimals	1.0
F4	Change of ETH/USD feed decimals	1.0
F5	Deployment of a new `AmountConverter` contract	1.0
F6	Deployment of a new `Stonks` contract	1.0
F7	Configuration of an ERC-20 token feed	1.0
F8	Configuration of token-specific ETH/USD max staleness parameter	1.0
F9	Configuration of token active/inactive status	1.0
F10	Transfer of random buy/sell token amount to random `Stonks` or `Order`	1.0
F11	Placement of a new order from `Stonks` balance	1.0
F12	Placement of a new order with defined sell amount	1.0
F13	Filling a random order (possibly partially)	1.0
F14	Recovery of tokens from `Order` after expiration	1.0
F15	Emergency cancellation of an `Order`	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	**Fail** (M1)
IV2	Feed is reported as not in-sync after changing the feed	1.0	Success
IV3	ETH/USD feed is reported as not in-sync after changing the feed	1.0	Success
IV4	Feed is reported as not in-sync after changing the feed decimals	1.0	Success
IV5	ETH/USD feed is reported as not in-sync after changing the feed decimals	1.0	Success
IV6	`Order` remains valid after donating sell tokens	1.0	**Fail** (L1)
IV7	`Order` remains valid after partially filling it	1.0	**Fail** (L1)
IV8	`Order` details match the expected values	1.0	Success
IV9	Token balances (shares) of important accounts match the expected values	1.0	Success
IV10	Estimated buy token amount of matches the expected value	1.0	Success

Table 5. Wake fuzzing invariants

Appendix C: Wake AI Findings

This section lists vulnerabilities identified by Wake AI, an LLM-powered audit tool used for AI-assisted vulnerability discovery during the audit. Wake AI 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 AI. These findings are included regardless of whether they were also discovered independently by auditors during manual review.

Finding title	Severity	Reported	Discoverer
M1: Flawed ETH/USD routing	Medium	1.0	Wake AI, Auditor
L2: Stonks can be unpaused after `killSwitch`	Low	1.0	Wake AI, Auditor
W5: Misleading use of errors	Warning	1.0	Wake AI, Auditor
I9: Misleading event parameter name	Info	1.0	Wake AI

Table 6. Table of findings identified by Wake AI

ackee
blockchain security

Thank You

Ackee Blockchain a.s.

Rohanske nabrezi 717/4
186 00 Prague
Czech Republic

hello@ackee.xyz