ackee | blockchain security

Lido Finance

Stonks

by Ackee Blockchain

18.3.2024

ackee | blockchain security

Contents

1. Document Revisions

1.0	Final report	5.2.2024
1.1	Fix review	8.2.2024
1.2	Deployment verification	18.3.2024

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run free certification courses School of Solana, Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, RockawayX.

2.2. Audit Methodology

1. **Technical specification/documentation** - a brief overview of the system is requested from the client and the scope of the audit is defined.
2. **Tool-based analysis** - deep check with automated Solidity analysis tools and Wake is performed.
3. **Manual code review** - the code is checked line by line for common vulnerabilities, code duplication, best practices and the code architecture is reviewed.
4. **Local deployment + hacking** - the contracts are deployed locally and we try to attack the system and break it.
5. **Unit and fuzz testing** - run unit tests to ensure that the system works as expected, potentially write missing unit or fuzz tests.

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 (such as deployment scripts, compiler configuration, use of 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**	**-**
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 (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), 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 (see above) 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

Member's Name	Position
Štěpán Šonský	Lead Auditor
Michal Převrátil	Auditor, Fuzz Tester
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 Finance engaged Ackee Blockchain to perform a security review of the Lido Finance protocol with a total time donation of 7 engineering days in a period between January 28 and February 5, 2024, with Štěpán Šonský as the lead auditor.

We began our review using static analysis tools, including Wake. We then took a deep dive into the logic of the contracts. In parallel, we have involved Wake testing framework for implementing a differential fuzz test with fork testing, including Chainlink simulations. See Appendix C for more details. During the review, we paid special attention to:

- the correctness of the Chainlink integration,
- compliance with EIP-1271 standard,
- ensuring the arithmetic of the system is correct,
- detecting possible reentrancies in the code,
- ensuring access controls are not too relaxed or too strict,
- looking for common issues such as data validations,
- searching for possible gas optimizations.

Our review resulted in 9 findings, ranging from Info to Medium severity. The most severe one breaks the protocol when working with non-standard ERC-20 tokens like USDT (see M1). The overall code quality is solid, follows best practices, and contains proper data validations. The code is fully covered with detailed NatSpec documentation, and developers provided a comprehensive specification for onboarding.

Ackee Blockchain recommends Lido Finance:

- use `SafeERC20.forceApprove` or `SafeERC20.safeIncreaseAllowance` instead of `ERC20.approve`,
- be aware of Chainlink data feed limitations (deviation threshold, typical heartbeat) that may be the limiting factor for fairness evaluation,
- avoid assuming stablecoins are always valued at 1 USD,
- address all other reported issues.

See Revision 1.0 for the system overview of the codebase.

Revision 1.1

The fix review was done on the given commit: `4c51f22`. The issues M1, W1, W3, I1, I2, I3, and I4 were fixed according our recommendations. Warnings W2 and W4 were acknowledged as unlikely, regarding the client's intentions.

Revision 1.2

Lido Finance engaged Ackee Blockchain to verify that contracts deployed on the Ethereum mainnet correspond to the reviewed codebase. The verification was performed on the commit `40af0bb` [1] with minor changes since the last review for the scoped contracts:

- Stonks
- StonksFactory
- AmountConverter
- AmountConverterFactory
- Order

See Revision 1.2 for the description of the changes.

The compiler optimizer was enabled since the last review, so the fuzz testing performed in the Revision 1.0 cannot guarantee the absence of issues originating from compiler optimizer bugs.

The verification concluded successfully with an exact bytecode match achieved for all scoped contracts at the following addresses on the Ethereum mainnet:

- Stonks: 0x3e2D251275A92a8169A3B17A2C49016e2de492a7
- StonksFactory: 0x1d64f763a9d9dC7d86b0EC17471FF1B13a5345c8
- AmountConverter: 0x12cc60eea45F705f069B43095FbF2Fb3c7f874c1
- AmountConverterFactory: 0x589757ad5A90e2092d353Ccb05C91D5704696DD7
- Order: 0x2cB20C0223a159DD861c76eCEB419C6030a4c9c2

See Appendix C for more detailed information about the verification process.

[1] full commit hash: 40af0bb3ede47b7666d402280b96f05e6c897546

4. Summary of Findings

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

- a *Description*,
- an *Exploit scenario*,
- a *Recommendation* and if applicable
- a *Fix*.

There might often be multiple ways to solve or alleviate the issue, with varying requirements regarding the necessary changes to the codebase. In that case, we will try to enumerate them all, clarifying which solves the underlying issue better (albeit possibly only with architectural changes) than others.

	Severity	Reported	Status
M1: `SafeERC20` not used for `approve`	Medium	1.0	Fixed
W1: `MIN_POSSIBLE_BALANCE` inadequate logic	Warning	1.0	Fixed
W2: Chainlink feed registry mainnet-only	Warning	1.0	Acknowledged
W3: `InvalidExpectedOutAmount` reverts	Warning	1.0	Fixed
W4: Stablecoins assumed stable	Warning	1.0	Acknowledged
I1: Unused code	Info	1.0	Fixed

	Severity	Reported	Status
I2: Missing event	Info	1.0	Fixed
I3: Duplicated code	Info	1.0	Fixed
I4: Typos and documentation	Info	1.0	Fixed

Table 2. Table of Findings

5. Report revision 1.0

5.1. System Overview

This section contains an outline of the audited contracts. Note that this is meant for understandability purposes and does not replace project documentation.

Contracts

Contracts we find important for better understanding are described in the following section.

Stonks.sol

The `Stonks` contract is used for creating and managing swap orders between the specified ERC-20 tokens pair. The contract inherits from `AssetRecoverer` contract. Therefore contains asset recovery functions and the `Ownable` logic as well. The order creation is realized in the `placeOrder` function using the following steps:

1. Validate the `minBuyAmount_` for non-zero value and the contract balance of `TOKEN_FROM` has to be greater than `MIN_POSSIBLE_BALANCE`.
2. Deploy a minimal proxy (using OpenZeppelin `Clones` library) for the provided `ORDER_SAMPLE` logic contract.
3. Transfer the whole contract's balance of `TOKEN_FROM` to the newly created clone.
4. Call the `initialize` function on the clone.

Finally, there are a few view functions:

- `estimateTradeOutput` - fetches the expected `TOKEN_TO` amount from the `AmountConverter` according to input `amount_` of `TOKEN_FROM` and cuts the

`MARGIN_IN_BASIS_POINTS` (CoW Protocol fees and maximum accepted losses),

- `estimateTradeOutputFromCurrentBalance` - calls the `estimateTradeOutput` with the contract's `TOKEN_FROM` balance,
- `getOrderParameters` - returns `TOKEN_FROM`, `TOKEN_TO` and `ORDER_DURATION_IN_SECONDS` immutables,
- `getPriceTolerance` - returns the `PRICE_TOLERANCE_IN_BASIS_POINTS` immutable.

Order.sol

The `Order` contract is supposed for one-time use and holds all necessary parameters for the swap execution (`sellAmount`, `buyAmount`, `validTo` and `orderHash`). The contract inherits from `AssetRecoverer` and implements the ERC-1271 standard for signature validation. The function `initialize` can be called only once and initializes the contract using the following steps:

1. Sets the `stonks` address to `msg.sender` and `manager` to `manager_` passed by parameter.
2. Gets `tokenFrom`, `tokenTo` and `orderDurationInSeconds` using the `stonks.getOrderParameters` function.
3. Creates the `GPv2Order` of `KIND_SELL` using the parameters above, `receiver` is set to `AGENT`. The contract's `tokenFrom` balance is used as the `sellAmount` and the `buyAmount` is used as `Math.max` from `minBuyAmount_` and `stonks.estimateTradeOutput`.

The ERC-1271 `isValidSignature` function contains the additional logic for time validation. And if the `currentCalculatedBuyAmount` is greater than `buyAmount`, there is a price tolerance logic for invalidating the order in case of excessive output amount deviation defined in the `Stonks.PRICE_TOLERANCE_IN_BASIS_POINTS`.

The overridden function `recoverERC20` allows the Agent and Manager to recover any ERC-20 token except the `TOKEN_FROM`.

The function `recoverTokenFrom` allows anyone to recover the `TOKEN_FROM` to the `Stonks` contract after order expiration.

The view function `getOrderDetails` returns the `tokenFrom` and `tokenTo` from the `Stonks` contract and adds `orderHash`, `sellAmount`, `buyAmount` and `validTo` parameters.

AmountConverter.sol

The `AmountConverter` contract is a Chainlink data provider for `Stonks` contract. It contains mappings for `allowedTokensToSell` and `allowedTokensToBuy`. Also, there is a `priceFeedsHeartbeatTimeouts` mapping for price feed timeouts.

The only external function `getExpectedOut` provides the expected amount output, based on input parameters `tokenFrom_`, `tokenTo_` and `amountFrom_`.

AssetRecoverer.sol

The `AssetRecoverer` contains asset recovery functions, namely `recoverEther`, `recoverERC20`, `recoverERC721` and `recoverERC1155`. These functions allow the Agent and Manager to withdraw the respective assets from the contract to the agent address. The `AssetRecoverer` contract inherits from the `Ownable` contract.

Ownable.sol

The Ownable contract introduces two roles - Agent and Manager. It contains two modifiers `onlyAgent` and `onlyAgentOrManager` which restrict functions execution only to these roles. The `AGENT` is immutable and cannot be set to zero-address. The Manager can be set by the Agent and can be zero-address.

Actors

This part describes actors of the system, their roles, and permissions.

Agent

The Agent role is introduced in the `Ownable` contract. The agent can set the Manager and has a privileged role in child contracts. In `AssetRecoverer`, the Agent can call `recoverEther`, `recoverERC20`, `recoverERC721` and `recoverERC1155` functions. Additionally, in `Stonks` contract, the agent can call the `placeOrder` function.

Manager

The Manager role is defined in the `Ownable` contract, can be set by the Agent and can be zero-address. The manager has the same permissions as the Agent, excluding the role management.

5.2. Trust Model

The system heavily relies on Agent and Manager roles which can place orders and recover assets from `Stonks` and `Order` contracts to the predefined `AGENT` address. After the order expiration, anyone can call `Order.recoverTokenFrom` and trigger the `TOKEN_FROM` transfer to the `Stonks` contract.

M1: `SafeERC20` not used for `approve`

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	Order	Type:	Non-standard token interactions

Description

The contract `Order` has to give approval to the CoW Swap relayer contract for the future trade to succeed.

Listing 1. Excerpt from Order

```
121         IERC20(tokenFrom).approve(RELAYER, type(uint256).max);
```

The standard IERC20 interface is used for the approval call. The interface expects a boolean value returned. However, some tokens (including USDT, for example) do not return any value, resulting in the transaction failing because of an ABI decoding error.

This issue was found by a fuzz test written in the Wake testing framework. See Appendix C for more details.

Exploit scenario

The project is deployed, and an `Order` contract instance is created to sell USDT tokens. Due to the unsafe variant of the `approve` call, the transaction fails.

Recommendation

Use OpenZeppelin's `forceApprove` function from the `SafeERC20` library to give approval to the relayer contract.

Fix 1.1

Fixed, `ERC20.approve` has been replaced by `SafeERC20.forceApprove`.

Listing 2. Excerpt from Order

```
121         IERC20(tokenFrom).forceApprove(RELAYER, type(uint256).max);
```

Go back to Findings Summary

W1: `MIN_POSSIBLE_BALANCE` inadequate logic

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

Description

The project contains two constants named `MIN_POSSIBLE_BALANCE` with the same values used similarly.

Listing 3. Excerpt from Order.recoverTokenFrom

```
206         if (balance <= MIN_POSSIBLE_BALANCE) revert
    InvalidAmountToRecover(balance);
```

Listing 4. Excerpt from Stonks.placeOrder

```
133         if (balance <= MIN_POSSIBLE_BALANCE) revert
    MinimumPossibleBalanceNotMet(MIN_POSSIBLE_BALANCE, balance);
```

The name implies the constants hold the minimum value that must be used as a balance for a transaction to succeed. This is further supported by the name of the error `MinimumPossibleBalanceNotMet` in the above example. However, the logic in both cases uses `MIN_POSSIBLE_BALANCE` as the highest value that causes the transaction to fail.

This issue was found by a fuzz test written in the Wake testing framework. See Appendix C for more details.

Recommendation

Change the equation sign from `<=` to `<` to make the behavior more intuitive based on the name of the constants. If necessary, also update the value of both constants.

Fix 1.1

Fixed, the operator has been changed to < according to our recommendation.

Listing 5. Excerpt from Order.recoverTokenFrom

```
206         if (balance < MIN_POSSIBLE_BALANCE) revert
    InvalidAmountToRecover(balance);
```

Listing 6. Excerpt from Stonks.placeOrder

```
131         if (balance < MIN_POSSIBLE_BALANCE) revert
    MinimumPossibleBalanceNotMet(MIN_POSSIBLE_BALANCE, balance);
```

Go back to Findings Summary

W2: Chainlink feed registry mainnet-only

Impact:	Warning	Likelihood:	N/A
Target:	`**/*`	Type:	Chain differences

Description

The project heavily depends on the Chainlink Feed Registry. According to the documentation, the registry contract is only deployed on the Ethereum mainnet. This is a limiting factor for deployment of this project to other chains.

Recommendation

Ensure the project is expected to be deployed only on the Ethereum mainnet, or it is acceptable to maintain a modified version of this project for different chains. Otherwise, re-implement the project using addresses of the necessary data feeds directly from constructor parameters or initializers.

Fix 1.1

Acknowledged.

> Risk acknowledged. Deployment on different chains isn't planned by design.
>
> — Lido

Go back to Findings Summary

W3: `InvalidExpectedOutAmount` reverts

Impact:	Warning	Likelihood:	N/A
Target:	Order, Stonks	Type:	Code quality, Logic error

Description

The error `InvalidExpectedOutAmount` is raised in the function `AmountConverter.getExpectedOut` when the expected amount of tokens to receive (from a trade, for example) reaches zero.

Listing 7. Excerpt from AmountConverter.getExpectedOut

```
103         (uint256 currentPrice, uint256 feedDecimals) =
    _fetchPriceAndDecimals(tokenFrom_, CONVERSION_TARGET);
104
105         uint256 decimalsOfSellToken = IERC20Metadata(tokenFrom_).decimals();
106         uint256 decimalsOfBuyToken = IERC20Metadata(tokenTo_).decimals();
107
108         int256 effectiveDecimalDifference = int256(decimalsOfSellToken +
    feedDecimals) - int256(decimalsOfBuyToken);
109
110         if (effectiveDecimalDifference >= 0) {
111             expectedOutputAmount = (amountFrom_ * currentPrice) / 10 **
    uint256(effectiveDecimalDifference);
112         } else {
113             expectedOutputAmount = (amountFrom_ * currentPrice) * 10 **
    uint256(-effectiveDecimalDifference);
114         }
115
116         if (expectedOutputAmount == 0) revert
    InvalidExpectedOutAmount(expectedOutputAmount);
```

The function `AmountConverter.getExpectedOut` is external and may be called by EOA (externally owned account). Furthermore, it is called from the function `Stonks.estimateTradeOutput`, where an additional margin is applied, lowering the final out amount.

Listing 8. Excerpt from Stonks.estimateTradeOutput

```
161     function estimateTradeOutput(uint256 amount_) public view returns
    (uint256) {
162         if (amount_ == 0) revert InvalidAmount(amount_);
163         uint256 expectedBuyAmount =
    IAmountConverter(AMOUNT_CONVERTER).getExpectedOut(TOKEN_FROM, TOKEN_TO,
    amount_);
164         return (expectedBuyAmount * (MAX_BASIS_POINTS -
    MARGIN_IN_BASIS_POINTS)) / MAX_BASIS_POINTS;
165     }
```

The margin application may cause the returned value to be zero, making the discussed zero amount check in the contract `AmountConverter` insufficient.

The function `Stonks.estimateTradeOutput` is then used in two project-critical functions:

- `Order.initialize` when computing the amount of tokens to receive (buy token amount) for a new order,
- `Order.isValidSignature` when checking if the trade would be accepted under conditions currently favorable to the contract owner.

Zero buy token amount in the first case cannot cause any harm, as the final amount of tokens is a result of a maximum from the computed value and `minBuyAmount_`, which cannot be zero.

Listing 9. Excerpt from Order.initialize

```
99          buyAmount = Math.max(IStonks(stonks).estimateTradeOutput(sellAmount),
   minBuyAmount_);
```

In the second case, it is preferable to return the zero value rather than reverting the transaction. With the zero value returned, the trade is accepted under conditions currently favorable to the contract owner, while the revert statement causes the trade to be rejected.

This issue was found by a fuzz test written in the Wake testing framework. See Appendix C for more details.

Recommendation

Consider removing the revert statement raising the `InvalidExpectedOutAmount` error as:

- the logic of the project does not allow creating an order with the zero out amount (buy token amount) anyway,
- the revert may cause a rejection of a trade (favorable to the contract owner) under edge case conditions.

Fix 1.1

Fixed, the revert statement and the error `InvalidExpectedOutAmount` itself have been removed from the `AmountConverter` contract.

Go back to Findings Summary

W4: Stablecoins assumed stable

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

Description

The project is generally designed to trade a given amount of sell tokens for another amount of buy tokens. A single aggregated Chainlink data feed is used to calculate the buy amount. However, prices from Chainlink data feeds are not denominated in each common stablecoin but in USD. The scenario of selling a token for a stablecoin while using a price denominated in USD is further supported by integration and unit tests present in the project.

Using the value of a stablecoin as 1 USD can lead to catastrophic consequences in the event of the stablecoin depeg.

Recommendation

Consider using one more price data feed for correct price calculation between a stablecoin and USD (for each stablecoin in a trade). Otherwise, ensure everyone using the project is familiar with the limitations of the solution, and a trade during a stablecoin depeg must never be opened. Document this limitation in the project's README.

Fix 1.1

Acknowledged.

> Risk of depeg acknowledged. The solution is only meant to be used a good market conditions, should be handled by ops.
>
> — Lido

Go back to Findings Summary

I1: Unused code

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

Description

The contract `AmountConverter` contains the `NoPriceFeedFound` error declaration, but the error is never used.

Listing 10. Excerpt from AmountConverter

```
35     error NoPriceFeedFound(address tokenFrom, address tokenTo);
```

The contract `Stonks` uses the `SafeCast` library through a using-for directive.

Listing 11. Excerpt from Stonks

```
28     using SafeCast for uint256;
```

However, the library `SafeCast` is never used in the contract.

Recommendation

Remove the unused error and using-for directive to keep up with the best practices.

Fix 1.1

Fixed, the unused code has been removed.

Go back to Findings Summary

I2: Missing event

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

Description

The contract `Stonks` emits an event for each parameter set, except for the price tolerance parameter.

Listing 12. Excerpt from Stonks

```
110         PRICE_TOLERANCE_IN_BASIS_POINTS = priceToleranceInBasisPoints_;
111
112         emit ManagerSet(manager_);
113         emit AmountConverterSet(amountConverter_);
114         emit OrderSampleSet(orderSample_);
115         emit TokenFromSet(tokenFrom_);
116         emit TokenToSet(tokenTo_);
117         emit OrderDurationInSecondsSet(orderDurationInSeconds_);
118         emit MarginInBasisPointsSet(marginInBasisPoints_);
```

Recommendation

Emit a new event with the `PRICE_TOLERANCE_IN_BASIS_POINTS` set in the contract constructor to keep up with the conventions established in the project.

Fix 1.1

Fixed. The event `PriceToleranceInBasisPointsSet` has been added.

Listing 13. Excerpt from Stonks

```
116         emit PriceToleranceInBasisPointsSet(priceToleranceInBasisPoints_);
```

Go back to Findings Summary

I3: Duplicated code

Impact:	Info	Likelihood:	N/A
Target:	`**/*`	Type:	Code quality

Description

In the `Stonks` contract constructor, there is a block of code that can be simplified.

Listing 14. Excerpt from Stonks

```
86          if (orderDurationInSeconds_ < MIN_POSSIBLE_ORDER_DURATION_IN_SECONDS)
    {
87              revert InvalidOrderDuration(
88                  MIN_POSSIBLE_ORDER_DURATION_IN_SECONDS,
    MAX_POSSIBLE_ORDER_DURATION_IN_SECONDS, orderDurationInSeconds_
89              );
90          }
91          if (orderDurationInSeconds_ > MAX_POSSIBLE_ORDER_DURATION_IN_SECONDS)
    {
92              revert InvalidOrderDuration(
93                  MIN_POSSIBLE_ORDER_DURATION_IN_SECONDS,
    MAX_POSSIBLE_ORDER_DURATION_IN_SECONDS, orderDurationInSeconds_
94              );
95          }
```

Recommendation

Join the condition using the `OR` operator and remove the code duplication.

```
if (orderDurationInSeconds_ < MIN_POSSIBLE_ORDER_DURATION_IN_SECONDS ||
orderDurationInSeconds_ > MAX_POSSIBLE_ORDER_DURATION_IN_SECONDS) {
    revert InvalidOrderDuration(
        MIN_POSSIBLE_ORDER_DURATION_IN_SECONDS,
MAX_POSSIBLE_ORDER_DURATION_IN_SECONDS, orderDurationInSeconds_
    );
}
```

Fix 1.1

Fixed. The conditions have been joined according to our recommendation.

Listing 15. Excerpt from Stonks

```
85         if (
86             orderDurationInSeconds_ > MAX_POSSIBLE_ORDER_DURATION_IN_SECONDS
87                 || orderDurationInSeconds_ <
   MIN_POSSIBLE_ORDER_DURATION_IN_SECONDS
88         ) {
```

Go back to Findings Summary

I4: Typos and documentation

Impact:	Info	Likelihood:	N/A
Target:	`**/*`	Type:	Code quality

Description

The codebase contains a few typos and inconsistencies in the documentation and code itself.

1) In the Stonks contract documentation, there is a typo ”tokerance”.

Listing 16. Excerpt from Stonks

```
20  *  - Stores key trading parameters: token pair, margin, price tokerance and
   order duration in immutable variables.
```

2) Error definitions `InvalidOrderDuration`, `MarginOverflowsAllowedLimit`, `PriceToleranceOverflowsAllowedLimit`, and `MinimumPossibleBalanceNotMet` in the `Stonks` contract contain typo ”recieved”.

Listing 17. Excerpt from Stonks

```
59      error InvalidOrderDuration(uint256 min, uint256 max, uint256 recieved);
60      error MarginOverflowsAllowedLimit(uint256 limit, uint256 recieved);
61      error PriceToleranceOverflowsAllowedLimit(uint256 limit, uint256
   recieved);
62      error MinimumPossibleBalanceNotMet(uint256 min, uint256 recieved);
```

3) The diagram in `Stonks.estimateTradeOutput` function documentation contains `expectedBuyAmount`, but the description refers to `expectedPurchaseAmount`.

Listing 18. Excerpt from Stonks

```
150     *
```

```
151      * |       estimatedTradeOutput        expectedBuyAmount
152      * |  --------------*--------------------------*----------------->
    amount
153      * |                 <-------- margin -------->
154      *
155      * where:
156      *      expectedPurchaseAmount - amount received from the
    amountConverter based on Chainlink price feed.
157      *      margin - % taken from the expectedPurchaseAmount includes CoW
    Protocol fees and maximum accepted losses
158      *               to handle market volatility.
159      *      estimatedTradeOutput - expectedPurchaseAmount subtracted by the
    margin that is expected to be result of the trade.
160      */
```

Recommendation

Fix typos and inconsistencies in the code.

Fix 1.1

Fixed. All reported typos and inconsistencies have been fixed.

Go back to Findings Summary

6. Report revision 1.1

6.1. System Overview

Aside from issue fixes, no additional changes in the system or trust model were made.

7. Report revision 1.2

7.1. System Overview

Three minor changes were made in this revision.

The `Order.DomainSeparatorSet` event parameter name and type was changed to reflect the event name and the corresponding value was passed to an emit statement in the contract constructor.

The name of the `AmountConverter.priceFeedsHeartbeatTimeouts` mapping key parameter was changed for improved readability.

The `Order.APP_DATA` constant used as a CoW Swap order parameter was changed from `keccak256("LIDO_DOES_STONKS")` to `keccak256("{}")`.

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Lido Finance: Stonks, 18.3.2024.

Appendix B: Glossary of terms

The following terms might be used throughout the document:

Superclass/Ancestor of C

A contract that C inherits/derives from.

Subclass/Child of C

A contract that inherits/derives from C.

Syntactic contract

A Solidity contract. May have an inheritance chain, and may be deployed.

Deployed contract

An EVM account with non-zero code. If its source was written in Solidity, it was created through at least one syntactic contract. If that contract had superclasses (parents), it would be composed of multiple syntactic contracts.

Init/initialization function

A non-constructor function that serves as an initializer. Often used in upgradeable contracts.

External entrypoint

A `public` or `external` function.

Public/Publicly-accessible function/entrypoint

An `external` or `public` function that can be successfully executed by any network account.

Mutating function

A non-`view` and non-`pure` function.

Appendix C: Wake outputs

This section contains the selected outputs of the Wake tool used during the audit.

C.1. Graphs

The following inheritance graphs generated by the Wake tool were used during the audit to streamline the review process.

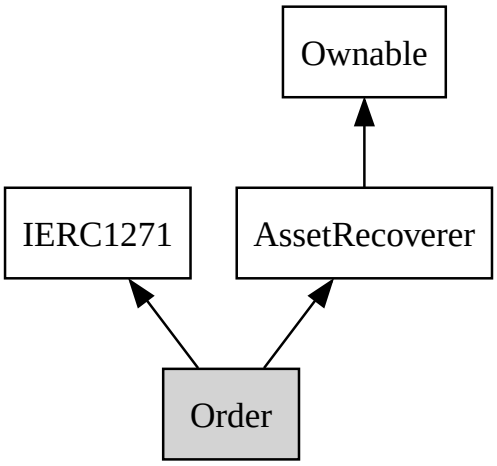

Figure 1. Order inheritance graph

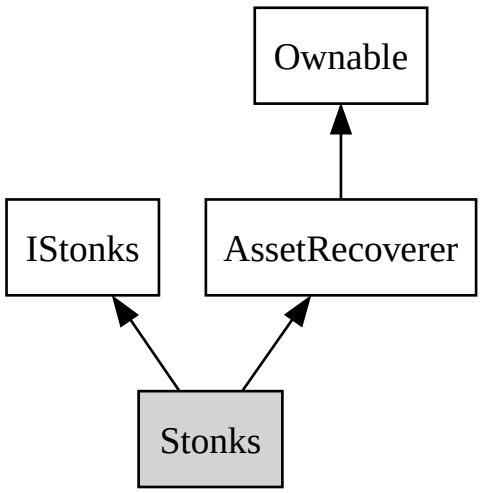

Figure 2. Stonks inheritance graph

C.2. Tests

The project was tested with a differential fuzz test written in the Wake testing framework. The test was designed to test different parameters of placed orders and, thus, different scenarios of the contract's behavior. The fuzzing run was scheduled using multiprocessing, with each process having a different seed for random number generation. Solidity code coverage was performed using the Wake tool to ensure that all desired code paths were sufficiently tested.

The issues M1, W1, and W3 were discovered during the fuzzing process. The complete source code of the fuzz test can be found at https://github.com/Ackee-Blockchain/tests-lido-stonks.

C.3. Deployment verification

The following script and compiler settings in the Wake testing framework were used to verify the deployment of contracts in the Revision 1.2. The verification concluded successfully with an exact bytecode match for all deployed contracts.

```
[compiler.solc]
exclude_paths = ["node_modules", "venv", ".venv", "lib", "script", "test"]
include_paths = ["node_modules"]
target_version = "0.8.23"
evm_version = "paris"

[compiler.solc.optimizer]
enabled = true
runs = 200
```

```
pre_deploy = Chain()
post_deploy = Chain()

@pre_deploy.connect(fork="http://localhost:8545@19434933")
@post_deploy.connect(fork="http://localhost:8545")
def test_deployment():
    stonks = Account("0x3e2D251275A92a8169A3B17A2C49016e2de492a7",
chain=post_deploy)
    stonks_factory = Account("0x1d64f763a9d9dC7d86b0EC17471FF1B13a5345c8",
chain=post_deploy)
    amount_converter = Account("0x12cc60eea45F705f069B43095FbF2Fb3c7f874c1",
chain=post_deploy)
    amount_converter_factory =
Account("0x589757ad5A90e2092d353Ccb05C91D5704696DD7", chain=post_deploy)
    order = Account("0x2cB20C0223a159DD861c76eCEB419C6030a4c9c2",
chain=post_deploy)

    pre_deploy.set_default_accounts(
```

```
"0x7Cd64b87251f793027590c34b206145c3aa362Ae")

    assert stonks_factory.code == StonksFactory.deploy(
        "0x3e40D73EB977Dc6a537aF587D48316feE66E9C8c",
        "0x9008D19f58AAbD9eD0D60971565AA8510560ab41",
        "0xC92E8bdf79f0507f65a392b0ab4667716BFE0110",
        chain=pre_deploy,
    ).code

    assert order.code == Order.deploy(
        "0x3e40D73EB977Dc6a537aF587D48316feE66E9C8c",
        "0xC92E8bdf79f0507f65a392b0ab4667716BFE0110",
        bytes
.fromhex("c078f884a2676e1345748b1feace7b0abee5d00ecadb6e574dcdd109a63e8943"),
        chain=pre_deploy,
    ).code

    assert amount_converter_factory.code == AmountConverterFactory.deploy(
        "0x47Fb2585D2C56Fe188D0E6ec628a38b74fCeeeDf", chain=pre_deploy
    ).code

    assert amount_converter.code == AmountConverter.deploy(
        "0x47Fb2585D2C56Fe188D0E6ec628a38b74fCeeeDf",
        "0x0000000000000000000000000000000000000348",
        [
            "0xae7ab96520DE3A18E5e111B5EaAb095312D7fE84",  # stETH
            "0x6B175474E89094C44Da98b954EedeAC495271d0F",  # DAI
            "0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48",  # USDC
            "0xdAC17F958D2ee523a2206206994597C13D831ec7",  # USDT
        ],
        [
            "0x6B175474E89094C44Da98b954EedeAC495271d0F",  # DAI
            "0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48",  # USDC
            "0xdAC17F958D2ee523a2206206994597C13D831ec7",  # USDT
        ],
        [4500, 4500, 88200, 88200],
        chain=pre_deploy,
    ).code

    assert stonks.code == Stonks.deploy(
        "0x3e40D73EB977Dc6a537aF587D48316feE66E9C8c",
        "0xa02FC823cCE0D016bD7e17ac684c9abAb2d6D647",
        "0xae7ab96520DE3A18E5e111B5EaAb095312D7fE84",
        "0x6B175474E89094C44Da98b954EedeAC495271d0F",
        "0x12cc60eea45F705f069B43095FbF2Fb3c7f874c1",
```

```
        "0x2cB20C0223a159DD861c76eCEB419C6030a4c9c2",
        1800, 110, 550, chain=pre_deploy
    ).code
```

ackee | blockchain security

Thank You

Ackee Blockchain a.s.


Prague, Czech Republic


hello@ackeeblockchain.com


https://twitter.com/AckeeBlockchain