SOLIDIFIED

Audit Report for Ola-X - November 7, 2022

Summary

Audit Report prepared by Solidified covering the Ola-X smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on November 7, 2022, and the results are presented here.

Audited Files

The source code has been supplied in a private source code repository:

https://github.com/ola-finance/olaX-audit

Commit number: `940ddbf67c5fed4d157380ea705bbad88d3db84c`

Update: code fixes were received on November 22, 2022.
Updated commit number: `a2726551342baae6e69fdfb0851a247d4c1475d1`

Intended Behavior

Ola-X is an on-chain system for margin trading that acts as an extension over AMMs that offer on-chain spot liquidity.

Ola-X contains 3 main parts:

1. A liquidity pool consisting of both lenders and borrowers.
2. A broker that allows traders to leverage, deleverage, and liquidate undercollateralized positions.
3. A registry that has a suite of administrative responsibilities.

Findings

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note, that high complexity or lower test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than a security audit and vice versa.

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	High	-

Issues Found

Solidified found that the Ola-X contracts contain no critical issues, 10 major issues, 6 minor issues, and 12 informational notes.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	Handler.sol: Function deleteOrder() does not refund the sender's bribe	Major	Resolved
2	Handler.sol: Replacing a pending order will not refund the bribe	Major	Resolved
3	BPErc20V0_01.sol: Function initialize() can be called multiple times	Major	Resolved
4	BPDelegator.sol: A BPDelegator admin can collude with a RegistryV0 admin to drain the BPDelegator contract funds	Major	Acknowledged
5	RegistryV0.sol: Contract owner can liquidate borrowers by assigning a malicious oracle	Major	Acknowledged
6	OlaMarginBrokerDeployer.sol: Anyone can deploy a fake margin broker at a deterministic address to prevent deploying a legitimate broker	Major	Resolved
7	RegistryV0.sol: Anyone can register a new domain and prevent others from registering the domain name	Major	Resolved
8	ChainlinkPriceOracle.sol: Chainlink's latestRoundData() might return stale or incorrect results	Major	Acknowledged
9	ChainlinkPriceOracle.sol: The Chainlink price safety check does not prevent signed integer overflow	Major	Resolved

10	TradeInteractor.sol: Missing trade slippage protection	Major	Resolved
11	Fees.sol: Function constructorInitializeFees() fails to validate _actionFee and _liquidationFee	Minor	Resolved
12	AssetsManager.sol: Function withdrawAllAssets() fails to verify amounts length	Minor	Resolved
13	MarginBroker.sol: A leveraged position can not be reduced by less than the debt	Minor	Resolved
14	RegistryV0.sol: The contract version can be set to a version that is not the latest	Minor	Resolved
15	MarginBroker.sol: Increasing a position is susceptible to high slippage	Minor	Resolved
16	AssetsManager.sol: liquidationLimit can be initialized to a value < 1e18 in setLiquidationLimit()	Minor	Resolved
17	TwoStepsAdmin.sol: Function initializeTwoStepsAdmin() fails to validate firstAdmin	Note	Resolved
18	BrokerBase.sol: Function initializeBase() declares _borrowMMs/_allAssets as memory instead of calldata	Note	Resolved
19	AcceptableImplementationClaimableAdmin.sol: Redundant address(0) check in function _acceptImplementation()	Note	Acknowledged
20	RegistryV0.sol: The domain owner can set the contract version to any arbitrary value	Note	Resolved
21	Handler.sol: Unchecked low-level call() in transferBribe	Note	Resolved
22	BPErc20ImmutableV0_01.sol: Unnecessary check for msg.sender == admin	Note	Resolved
23	BPErc20V0_01.sol: Inherited variable	Note	Resolved

	shadowing		
24	BrokerPool.sol: Possible gas savings by not performing operation on type(uint).max	Note	Acknowledged
25	Handler.sol: Possible transfer of 0 ETH	Note	Resolved
26	BPErc20Delegator.sol: Unused return values	Note	Acknowledged
27	BrokerPool.sol: Incorrect description of parameter argument	Note	Resolved
28	AssetsManager.sol: Potential gas optimization in function borrowUpdateBalance()	Note	Resolved

Critical Issues

No critical issues have been found.

Major Issues

1. Handler.sol: Function deleteOrder() does not refund the sender's bribe

The function `deleteOrder()` does not refund back the `bribe` paid by the sender when they created the order using `placeOrder()`, thus leading to permanent loss of funds for the user who created the order.

Recommendation

Refund back the paid `bribe` to the sender using `transferBribe()`.

Status

Resolved

2. Handler.sol: Replacing a pending order will not refund the bribe

A pending order with an associated bribe can be replaced by a new order by calling the `Handler.placeOrder()` function. However, the bribe of the replaced order is not refunded and remains locked in the contract.

Recommendation

Call the `Handler.deleteOrder()` function prior to replacing a pending order.

Status

Resolved

3. BPErc20V0_01.sol: Function initialize() can be called multiple times

The contract's `admin` can call the function `initialize()` multiple times, which allows them to change both the `underlying` and the `interestRateModel` after the market is live, potentially leading to loss of funds for the market users.

Recommendation

Restrict `initialize()` to be called only once.

Status

Resolved

4. BPDelegator.sol: A BPDelegator admin can collude with a RegistryV0 admin to drain the BPDelegator contract funds

Using the functions `RegistryV0.publishContractVersion()` and `BPDelegator.updateImplementationFromRegistry()`, both respective contract admins can collude to provide a malicious/buggy implementation that drains the users' funds after the market has gone live and deposits made.

Recommendation

Create a time interval that allows protocol participants ample time to withdraw their deposits before the newly set implementation is applied.

Status

Acknowledged. Team's response: "*The intention is that the admin of any BPDelegator contract is a contract that employs additional restrictions before executing any updates. Specifically before updating an implementation a timelock will be enforced*".

5. `RegistryV0.sol`: Contract owner can liquidate borrowers by assigning a malicious oracle

The function `setOracleForAsset()` allows the contract owner to potentially assign a malicious (or a buggy) oracle that would allow them (or an attacker) to liquidate the protocol borrowers.

Recommendation

`setOracleForAsset()` should not be able to immediately reassign an oracle, but rather give market participants adequate time to close their positions (in case they wish to) before a new oracle is assigned.

Status

Acknowledged. Team's response: "*The intention is that the Registry's admin is also a contract. However, in the case of updating the oracle, it might be a good idea to allow fast changes as oracles sometimes fail. It's more likely for the oracle to fail than for the Ola team to turn malicious, although this will be a decision that we, together with our partners, will take in the future*".

6. `OlaMarginBrokerDeployer.sol`: Anyone can deploy a fake margin broker at a deterministic address to prevent deploying a legitimate broker

A new margin broker can be deployed by calling the `RegistryV0.createBroker()` function. This function will call the `OlaMarginBrokerDeployer.deploy()` function to deploy a new `MarginBroker` contract with the `CREATE2` EVM opcode and register it in the registry. The address of the newly deployed margin broker is deterministic and is calculated by the `OlaMarginBrokerDeployer.generateSalt()` function.

The `OlaMarginBrokerDeployer.deploy()` function does not have any access restriction. Hence anyone can deploy a new margin broker contract at a deterministic address. This can be used to front-run and prevent deploying a legitimate broker at the same address.

Recommendation

We recommend adding the `msg.sender` address to the generated salt in the `OlaMarginBrokerDeployer.generateSalt()` function.

Status

Resolved

7. RegistryV0.sol: Anyone can register a new domain and prevent others from registering the domain name

The `RegistryV0.registerDomain()` function is used to register a new and unique domain name. However, this function does not have any access restrictions. Hence anyone can register a new domain name and prevent others from registering the same domain name.

Recommendation

We recommend adding admin-only access control to the `RegistryV0.registerDomain()` function.

Status

Resolved

8. ChainlinkPriceOracle.sol: Chainlink's latestRoundData() might return stale or incorrect results

In `ChainlinkPriceOracle.sol`, `latestRoundData()` is used, but there is no check if the return value indicates stale data. This could lead to stale prices, according to the Chainlink documentation:

https://docs.chain.link/docs/historical-price-data/#historical-rounds

Recommendation

Consider adding checks to prevent stale price data. In a highly volatile market environment, this could cause major issues.

Status

Acknowledged. Team's response: "*This is true, but there is not much we can do even if the returned value is stale. In most cases it is better to return the last value than to revert. We currently don't have an alternative price source we can use in such cases, but it is certainly on our pipeline*".

9. `ChainlinkPriceOracle.sol`: The Chainlink price safety check does not prevent signed integer overflow

In the `ChainlinkPriceOracle.getPriceFromSourceInternal()` function, checking the value of `int feedPriceRaw` to equal `int(feedPrice)` after casting `int feedPriceRaw` to the unsigned integer variable `feedPrice` is insufficient to prevent a signed integer overflow. The value of `feedPriceRaw` can be negative, and casting it to `uint` would result in a significant positive value. This would pass the `require` check and would continue with an invalid and inflated asset price.

Recommendation

We recommend either asserting that `feedPriceRaw` is a non-negative integer value before casting to `uint` or changing the `require` condition to `feedPrice == int(feedPrice)`.

Status

Resolved

10. `TradeInteractor.sol`: Missing trade slippage protection

The `TradeInteractor.trade` function is used to execute a trade between assets. However, even though the function parameter `TradeInfo.wantedOut` suggests slippage protection, the function does not check the actual amount of tokens received after the trade. This can be used to execute a trade with a significant slippage and cause a loss of funds.

Recommendation

We recommend adding slippage protection to the `TradeInteractor.trade` function by asserting the invariant `actualAmountOut >= info.wantedOut`.

Status

Resolved

Minor Issues

11. Fees.sol: Function constructorInitializeFees() fails to validate _actionFee and _liquidationFee

The function `constructorInitializeFees()` does not validate `_actionFee` and `_liquidationFee`, unlike functions `setActionFee()/setLiquidationFee()`.

Recommendation

Enforce that `_actionFee` and `_liquidationFee` are less than or equal to `maxActionFee` and `maxLiquidationFee`, respectively.

Status

Resolved

12. AssetsManager.sol: Function withdrawAllAssets() fails to verify amounts length

The function `withdrawAllAssets()` fails to verify that the `amounts` array is the same length as the `assets` array.

Recommendation

Verify that `amounts.length == assets.length` in order to minimize unintended mistakes by the user.

Note

The same issue also exists in `borrowAssetsUpdateBalances()`, `depositEquityAssets()`, `deleverageByDepositWithPermit()`, and `multiplePermits()`.

Status

Resolved

13. MarginBroker.sol: A leveraged position can not be reduced by less than the debt

The `MarginBroker.reducePosition()` function allows reducing the current leveraged position partially by a given `percentage` value. However, the current user debt `wantedOut` is used as the minimum amount of `_borrowAsset` to receive in the trade, preventing a partial position reduction.

Recommendation

Use an appropriate slippage value.

Status

Resolved

14. RegistryV0.sol: The contract version can be set to a version that is not the latest

Registering a new domain with the `RegistryV0.registerDomain()` function initializes the versions of the domain contracts with the `RegistryV0.setDomainVersionInternal()` function. This function is supposed to assert that the version is the latest version of the contract. However, the `require` statement in line 472 is a tautology. This can be used to set the contract version to a version different from the latest.

Recommendation

Change the `require` statement to `require(version == latestVersionForContract, "can only upgrade to latest");`.

Status

Resolved

15. MarginBroker.sol: Increasing a position is susceptible to high slippage

The `MarginBroker.increasePosition()` function is used to increase the size of an existing position. Part of the function logic is to execute a trade to buy the `boughtAsset` asset with the `tradeWrapper()` function. However, the provided slippage value `TradeInfo.wantedOut` is set to `0`. Due to the lack of slippage protection in the current `TradeInteractor.trade` implementation, changing the slippage value to a non-zero value has no effect unless the `TradeInteractor.trade()` function is updated to include slippage protection.

Recommendation

Use a non-zero slippage value to minimize the effect of slippage while increasing the position.

Status

Resolved

16. `AssetsManager.sol`: `liquidationLimit` can be initialized to a `value < 1e18` in `setLiquidationLimit()`

When initializing the `liquidationLimit` variable, it is possible to pass a parameter for `limit` that is less than `1e18`.

Recommendation

Enforce that `limit >= 1e18`.

Status

Resolved

Informational Notes

17. TwoStepsAdmin.sol: Function initializeTwoStepsAdmin() fails to validate firstAdmin

The `TwoStepsAdmin` contract becomes totally unusable in case `firstAdmin` is passed as `address(0)` by mistake in `initializeTwoStepsAdmin()`.

Recommendation

Consider enforcing that `firstAdmin != address(0)`.

Status

Resolved

18. BrokerBase.sol: Function initializeBase() declares _borrowMMs/_allAssets as memory instead of calldata

Since parameters declared as `calldata` are directly accessed without being copied from `memory` first, the function `initializeBase()` can potentially consume a lot more gas than it should when it declares `_borrowMMs/_allAssets` as `memory`.

Recommendation

Consider declaring `_borrowMMs/_allAssets` as `calldata` instead of `memory` in order to save on gas fees.

Note

This issue also applies to all other internal functions that declare array parameters as `memory`.

Status

Resolved

19. `AcceptableImplementationClaimableAdmin.sol`: Redundant `address(0)` check in function `_acceptImplementation()`

The `pendingImplementation != address(0)` check is redundant in function `_acceptImplementation()` since the function already enforces that `msg.sender == pendingImplementation`.

Recommendation

Consider removing the redundant statement in order to save on gas fees.

Note

A similar redundancy also exists in functions `AcceptableImplementationClaimableAdmin._acceptAdmin()` and `BPErc20ImmutableV0_01.constructor()`.

Status

Acknowledged. Team's response: "*This redundant check doesn't hurt*".

20. `RegistryV0.sol`: The domain owner can set the contract version to any arbitrary value

The domain owner can upgrade the version of specific contract types with the `RegistryV0.upgradeDomainVersion()` function. However, the function does not check if the new version is valid (i.e. the version is greater than the current version and the new version is the latest version). This will prevent updating a delegator's implementation if the version lacks an implementation.

Recommendation

Consider adding appropriate checks to the `RegistryV0.upgradeDomainVersion()` function to assert the `version` is higher than the currently used version and that the version can only be set to the latest version of `latestContractVersions[contractNameHash]`.

Status

Resolved

21. Handler.sol: Unchecked low-level call() in transferBribe

The peripheral `Handler` contract uses the low-level Solidity `call()` to transfer the bribe without checking the success value.

If the return value of a low-level `call()` is not checked, the execution may resume even if the function call throws an error. This will lead to the execution of the order without the bribe being transferred to the caller.

Recommendation

Consider checking the `success` return value of the low-level `call()` and reverting if it is `false`.

Status

Resolved

22. BPErc20ImmutableV0_01.sol: Unnecessary check for msg.sender == admin

The constructor in line 38 checks that `msg.sender == admin` . However, this will always be true because of line 37 where `admin = (payable) msg.sender`.

Recommendation

Consider removing the unnecessary check.

Status

Resolved

23. BPErc20V0_01.sol: Inherited variable shadowing

The `initialize` function shadows the inherited state variable `decimals`, which deviates from best practices.

Recommendation

Consider giving the variable a different name.

Note

The same issue also exists with the implementation variable in the function `publishContractVersion()`.

Status

Resolved

24. `BrokerPool.sol`: Possible gas savings by not performing operation on `type(uint).max`

In line 59 the `transferTokens()` function performs `startingAllowance - tokens` to calculate a new token allowance. However, if the `spender` and `src` are the same then `startingAllownace = type(uint).max` which results in more gas consumption than needed from the above subtraction operation.

Recommendation

Consider checking `startingAllownace != type(uint).max` before performing the subtraction operation to ensure the operation is not performed if `spender == src`.

Status

Acknowledged. Team's response: "*We prefer not to touch this piece of code that was taken as is from Compound*".

25. `Handler.sol`: Possible transfer of 0 ETH

The `transferBribe()` function does not check to ensure `msg.value > 0` when sending the `bribe`.

Recommendation

Consider adding a check to ensure `completeOrder.bribe > 0` before sending eth in the `transferBribe` function in order to save on gas fees.

Status

Resolved

26. `BPErc20Delegator.sol`: Unused return values

The `_setFixedReservedInterest()`, `_setFixedInterest()` and `_setInterestRateModel()` functions do not return the values of their respective `delegateToImplementation()` calls.

Recommendation

Consider returning the correct respective values in the aforementioned functions.

Status

Acknowledged. Team's response: "*The values returned are a byproduct of code forked from Compound, and don't obtain any interesting value*".

27. `BrokerPool.sol`: Incorrect description of parameter argument

The `repayBorrowInternal()`, `repayBorrowBehalfInternal()` and `repayBorrowFresh()` functions all mention a parameter value of -1 for `repayAmount` to indicate the repayment of the full amount. However, this is not possible as the parameter type is `uint`. Also, within the `repayBorrowFresh()` function `type(uint).max` is actually used to indicate the repayment of the full amount.

Recommendation

Consider updating the comments of all three functions to indicate the correct parameter value for the full repayment amount.

Status

Resolved

28. AssetsManager.sol: Potential gas optimization in function borrowUpdateBalance()

The `AssetsManager.borrowUpdateBalance()` function is executed for all assets in `borrowAssets`, even though the given `amount` can be `0`.

Recommendation

Consider returning early from the function in case `amount` equals `0` to save on gas fees.

Status

Resolved

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Lilum LTD or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Oak Security GmbH