Quantstamp

Levva Updates And Oracle Feature

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Margin Trading and Derivatives
Timeline	2024-03-28 through 2024-04-16
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Marginly Documentation Website
Source Code	• https://gitlab.com/eq-lab/marginly #2509922
Auditors	• Ruben Koch Senior Auditing Engineer • Julio Aguilar Auditing Engineer • Shih-Hung Wang Auditing Engineer • Mostafa Yassin Auditing Engineer

Documentation quality	Medium
Test quality	High
Total Findings	10 Fixed: 8 Acknowledged: 1 Mitigated: 1
High severity findings	1 Fixed: 1
Medium severity findings	3 Fixed: 3
Low severity findings	2 Fixed: 1 Mitigated: 1
Undetermined severity findings	0
Informational findings	4 Fixed: 3 Acknowledged: 1

Summary of Findings

Note: Since the audit, the client rebranded "Marginly" to "Levva", hence the report refers to the protocol as Marginly.

Marginly is a margin trading and derivatives platform that allows users to provide liquidity to the protocol while gaining interest, deposit collateral, open long/short positions, and liquidate unhealthy positions to ensure overall system health.

In this audit, we reviewed the changes to the core contracts that were added since our last audit report of the codebase and the code additions for the added oracles.

The changes to the core contract mainly revolve around the feature of adding the possibility to flip an existing position within one transaction, which is achieved with the added `sellBaseForQuote()` and `sellQuoteForBase()` functions, as well as some changes to the `receivePosition()` function. Furthermore, in addition to the previously supported UniswapV3 oracle, AlgebraFinance, Pyth and Chainlink are now supported as possible sources for on-chain prices.

The main finding of the audit revolves around an incorrect assumption in the Pyth and Chainlink integration that both base token and quote token share the same amount of decimals (MAR-2-1). Other, minor issues around the oracle integrations were identified in MAR-2-3, MAR-2-7 and MAR-2-10. Furthermore, the `receivePosition()` function is currently missing accounting for the deleveraging aspect of the protocol (MAR-2-2).

The audit team continues to find the codebase to be of high quality and was also pleased to see that the test suite benchmarks having improved since the last audit to nearly perfect metrics.

Update Fix-Review

All issues have been either fixed or acknowledged. Tests have been added to validate the fixes. The Marginly team has been very communicative and conscientious in the audit and fix-review process.

ID	DESCRIPTION	SEVERITY	STATUS
MAR-2-1	**Incorrect Chainlink and Pyth Oracle Implementation**	• High	Fixed
MAR-2-2	**Incorrect Accounting in `receivePosition()`**	• Medium	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
MAR-2-3	Missing Chainlink Price Freshness Checks	Medium	Fixed
MAR-2-4	Inaccurate Interest Calculation	Medium	Fixed
MAR-2-5	Minor Incorrect Roundings	Low	Mitigated
MAR-2-6	Potential DoS via Empty Market Exploit	Low	Fixed
MAR-2-7	Pyth Oracle Integration Improvements	Informational	Fixed
MAR-2-8	Bypassing the Position Minimum Amount Limit	Informational	Fixed
MAR-2-9	Missing Input Validation	Informational	Fixed
MAR-2-10	Notes on Oracle Price Update Delays	Informational	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

> **Disclaimer**
> Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

In scope of this audit were the changes of the core contracts of the protocols since the last audit (`packages/contracts/contracts/*`), as well as the changes and additions to the `packages/periphery/contracts/oracles/*` package. Notably, this audit did not include `packages/router` folder, which contains the contract responsible for routing the swaps to the specified DEX.

Files Included

Repo: https://gitlab.com/eq-lab/marginly(2509922690ff1b25175562f2517f4627181b8265)

Files:

packages/contracts/contracts/*

packages/periphery/contracts/oracles/*

Files Excluded

Everything not specified in "Files Included".

Findings

MAR-2-1 Incorrect Chainlink and Pyth Oracle Implementation

• High Fixed

Update

Marked as "Fixed" by the client.

Addressed in: `a76d4e3da33de7e040972c6c9bb3a7950330b73e` .

File(s) affected: `ChainlinkOracle.sol` , `PythOracle.sol` , `CompositeOracle.sol`

Description: The `ChainlinkOracle` and `PythOracle` contracts calculate the base prices based on the value returned by the price feeds. However, unlike Uniswap V3 TWAPs, Chainlink or Pyth Network price feeds do not consider the ERC-20 token decimals of the two tokens in the returned price. Therefore, the oracle contracts must scale the returned price based on the differences between the base and quote token's decimals. Otherwise, an incorrect price will be used if the token's decimals differ.

For example, assume a pool has ETH as the base token and USDC as the quote token, and Chainlink USDC/ETH price feed is configured as the oracle. If the oracle returns an `answer` of `3e14` with `decimals` as 18, since ETH has 18 decimals, and USDC has 6, the final base price should be `10**oracleDecimals / answer * 10**usdcDecimals / 10**ethDecimals = (1e18 / 3e14) * (1e6 / 1e18) = 3.3e-09` , i.e., 1 wei ETH is worth `3.3e-09` wei USDC.

It should be noted that the order of `quoteToken` and `baseToken` configured in the oracle may not necessarily be the same as that configured in `MarginlyPool` . Specifically, for the above example, the `baseToken` parameter to the `ChainlinkOracle.setPair()` call should be USDC, and the `quoteToken` parameter should be ETH. Admins should be cautious about this difference when configuring the oracle settings.

Recommendation: Consider modifying the `CompositeOracle` contract to consider the ERC-20 token decimals of the base and quote tokens. For example, in the `_getPrice()` function:

```
if (commonParams.pairMode == PairMode.Direct || commonParams.pairMode == PairMode.Reverse) {
    (uint256 priceNom, uint256 priceDenom) = getRationalPrice(quoteToken, baseToken);
    (priceNom, priceDenom) = applyDirection(priceNom, priceDenom, commonParams.pairMode);

    return Math.mulDiv(
        priceNom * 10 ** IERC20(quoteToken).decimals(),
        X96ONE,
        priceDenom * 10 ** IERC20(baseToken).decimals()
    );
}
```

A similar fix needs to be applied for the `PairMode.Composite` case as well.

Also, consider adding code comments to the `setPair()` function so developers or admins can be fully aware of such details regarding the order of base and quote tokens.

MAR-2-2 Incorrect Accounting in `receivePosition()`

• Medium Fixed

Update

Marked as "Fixed" by the client.

Addressed in: `2491e965ace6b7bc49f4da5e4514bfd0d5afac7a` .

File(s) affected: `MarginlyPool.sol`

Description: Whenever a position's debt is adjusted, the collateral part of the position should also be adjusted based on the deleveraging coefficients multiplied by the change in debt.

In the `receivePosition()` function, however, the position's collateral values are not correctly adjusted based on the (partially) repaid debt. This can cause overtaken positions to be accounted for more collateral than they should have since the reduction by the deleverage coefficient multiplied by the debt change is missing.

Recommendation: Add the subtraction of the deleveraging-accounted debt change to both the pool's and the position's collateral.

MAR-2-3 Missing Chainlink Price Freshness Checks

Medium Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `1c0d426a4c497ee7078b6ac1a3d2f966de388023`, `a76d4e3da33de7e040972c6c9bb3a7950330b73e` and `09c4e448bf536bd4446b9f5d773f7daeeee09603`.

File(s) affected: `ChainlinkOracle.sol`

Description: The `getRationalPrice()` function of `ChainlinkOracle` fetches the latest price from a Chainlink price feed. However, the code does not check price freshness. It is best practice to check whether the returned value, `updatedAt`, is within the acceptable range of `[block.timestamp - delta, block.timestamp]` to ensure the price's freshness and detect potential off-chain oracle failures.

If the oracle will be deployed on Arbitrum, it is recommended that the sequencer's uptime be checked before querying the price feed. An incorrect or stale price may be used if the sequencer is down. See Chainlink L2 Sequencer Uptime Feeds for more details.

Additionally, Chainlink oracles have minimum and maximum prices coded in the aggregator contracts. The price update transaction will fail if the new price exceeds the range. For example, the aggregator for the MATIC/USD price feed on Polygon has a hard-coded `minAnswer` of 1000000 (0.01 USD). If MATIC prices fall below 0.01 USD, the oracle will not be updated but still return a stale price.

Furthermore, the price returned by `params.dataFeed.latestRoundData()` should be validated to be greater than zero. The current check for `< 0` is not sufficient.

Recommendation:

1. Consider adding price freshness checks to the oracle contracts. A suitable `delta` value depends on the heartbeat of the price feed.
2. Consider adding sequencer uptime checks for oracles deployed on Arbitrum.
3. This is a general issue when using Chainlink price feeds. Possible mitigations can be implementing off-chain price monitoring and reacting when abnormalities occur, e.g., pausing the oracle.
4. Add a check to ensure that the price is greater than zero.

MAR-2-4 Inaccurate Interest Calculation

Medium Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `2ab4168f017a49e0b7055bbf9bc248bc06916ce6`.

File(s) affected: `MarginlyPool.sol`

Description: The interest rates for long and short positions are calculated based on the `longX96` and `shortX96` variables when the `accrueInterest()` function is called. Therefore, to maintain an accurate interest calculation, these variables must be updated through the `updateSystemLeverageLong()` and `updateSystemLeverageShort()` functions after the system's overall debt or collateral total value has been changed.

The `updateSystemLeverageLong()` and `updateSystemLeverageShort()` functions are called in the `execute()` function after a user action is performed. However, if the caller's position is underwater and liquidated, the system leverage will not be updated, causing inaccuracies when interests are accrued in the next block.

For example, when a large long position is liquidated (by the owner calling the `execute()` function), the interest rate owed by longs to shorts should decrease. Without updating the system's long leverage, longs will owe shorts more interest starting from the next block.

Recommendation: Consider updating the system leverage when the caller's position gets liquidated to ensure the interests are accrued accurately in the next block.

MAR-2-5 Minor Incorrect Roundings

Low Mitigated

Update

The client marked the issue as mitigated with the following explanation:

> All rounding changes must be tested on edge cases. It takes a lot of time to develop. We have decided not to follow this recommendation as we already have a "balanceSync" feature that solves all possible precision issues at the expense of "techPosition"

We agree that the listed mechanics mitigate the impact of potential rounding errors.

File(s) affected: `MarginlyPool.sol`

Description: It is important to always round in the protocol's favor to not accrue bad debt. Below we outline minor instances of rounding. Given the `X96` notation, these rounding errors are extremely minor, yet we would still like to point them out.

- In `withdrawBase()`, `discountedBaseCollateralDelta` should be rounded up explicitly. Conversely, in `withdrawQuote()`, `discountedQuoteCollateralDelta` should be rounded up explicitly.
- In `closePosition()`, `discountedCollateralDelta` should be explicitly rounded up, both in the long and short position case.
- In `short()`, `discountedBaseDebtChange` should be explicitly rounded up. Conversely, in `long()`, `discountedQuoteDebtChange` should be explicitly rounded up.
- In `depositBase()`, `realBaseDebt`, `discountedQuoteCollDelta`, and the calculation performed in the conditional statement in L476 should be rounded up explicitly. The result of the `mul()` operation for `discountedQuoteCollDelta` at L495 should be rounded up explicitly too.
- In `depositQuote()`, `realQuoteDebt` and `discountedBaseCollDelta` should be rounded up explicitly.
- In `liquidate()`, `positionBaseDebt`, `quoteCollToReduce`, `positionQuoteDebt`, and `baseCollToReduce` should be rounded up explicitly. When liquidating a short-position, `disQuoteDelta` should be explicitly rounded up. When liquidating long-positions, `disBaseDelta` should be explicitly rounded up.
- In `enactMarginCall()`, the multiplication performed for `realBaseDebt` and `realQuoteDebt` should be rounded up explicitly.
- In `sellBaseForQuote()`, the multiplication performed for `realQuoteDebt` should be rounded up. Conversely, in `sellQuoteForBase()`, `realBaseDebt` should be rounded up explicitly.
- In `accrueInterest()`, the multiplications performed for `realBaseDebtPrev` and `realQuoteDebtPrev` should be rounded up explicitly.
- In `calcRealBaseCollateral()`, the result of `baseDelevCoeff.mul(disQuoteDebt)` should be rounded up explicitly. Conversely, in `calcRealQuoteCollateral()`, the result of `quoteDelevCoeff.mul(disBaseDebt)` should be rounded up explicitly.
- In `positionHasBadLeverage()`, all multiplications performed between `FP96.FixedPoint` and `uint256` for `realTotalDebt` should be rounded up explicitly.
- In `updateHeap()` in the short case, `initialPrice.mul(position.discountedBaseAmount)` should be round up explicitly.
- In `receivePosition()`, the multiplications performed for `badPositionBaseDebt` and `badPositionQuoteDebt` should be rounded up explicitly.
- In `shutDown()`, the result of `basePrice.mul(baseDebt)` in L1271 should be rounded up explicitly.
- In `setEmergencyMode()`, the result of `shutDownPrice.mul(emergencyDebt)` should be rounded up explicitly.
- In `emergencyWithdraw()`, the multiplications performed for `positionBaseNet` and `positionQuoteNet` should be rounded up explicitly.
- In `updateSystemLeverageLong()`, `realQuoteDebt` should be rounded up. Conversely, in `updateSystemLeverageShort()` both multiplications performed for `realBaseDebt` should be rounded up explicitly. In both functions, `leverageX96` should be rounded up explicitly.

Recommendation: Consider adding the listed explicit roundings.

MAR-2-6 Potential DoS via Empty Market Exploit

Low · Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `9981adfa15fb8ed054d71f0378c2962fd123a12a` by implementing a reasonable lower bound value for the relevant coefficients.

File(s) affected: `MarginlyPool.sol`

Description: In two scenarios, the coefficients in the protocol get reduced:

1. Debt coefficients may decrease when deleveraging occurs.
2. Collateral and deleverage coefficients may decrease when the liquidated position has a negative net difference.

If a debt coefficient, for example, the `quoteDebtCoeff`, would ever turn zero, basically all aspects around the long positions in the pool would fully break, causing a DoS:

- Longs would no longer have bad leverage if `quoteDebtCoeff = 0`, given the `positionHasBadLeverage()` function.
- However, the longs could never be closed too if `quoteDebtCoeff = 0`, since the `SlippageLimit()` error should always hit, because `realQuoteDebt = 0`.
- Fee accrual that longs pay shorts would also stop working, because `factor` would always be of value zero.

For the `quoteDebtCoeff` to turn zero due to long-deleveraging, the following condition would need to hold:

`realQuoteDebtToDeleverage = quoteDebtCoeff * discountedQuoteDebt <=> realQuoteDebtToDeleverage = realPoolQuoteDebt`, because that would result in a `quoteDebtCoeff` to be assigned a zero value in L322 during the `deleverageLong()`

function. If a deleveraging requires the full pool debt to be reduced, i.e. when the only short position is fully deleveraged, the debt coefficients can turn irrevocably turn zero, causing a DoS on positions on one side of the pool.

This issue is mainly relevant for newly deployed pools, as in other scenarios a full debt deleverage is extremely unlikely. However, the consequences are severe.

Recommendation: Consider adding small positions to either side as part of the deployment of each Marginly pool. Consider enforcing `quoteDebtCoeff != 0` and `baseDebtCoeff != 0` after the update in `deleverageLong()` and `deleverageShort()`, respectively, even if it might cause bad debt in some scenarios.

MAR-2-7 Pyth Oracle Integration Improvements

• Informational Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `c63d22bb6f74e783f703381f520b8b557eebcfbc`. The second recommendation was acknowledged, which we find reasonable.

File(s) affected: `PythOracle.sol`

Description: This issue describes the possible improvements on the `PythOracle` contract that may enhance its robustness and security:

- `getPrice()` has a built-in check that ensures that the latest price has been updated within the last `getValidTimePeriod()` seconds. The default valid period is set to a reasonable value on each chain and is typically around 1 minute, which may not be suitable for every asset. The `getPriceNoOlderThan()` function allows the caller to adjust the time interval with the `age` parameter.
- Pyth reports prices including a confidence interval. For the base price, it might be worth enforcing that the interval does not exceed a certain percentual deviation from the reported value. Additionally, it might be worth using the lower bound value for collateral price calculations and using the upper bound value for debt and leverage calculations. For more information, refer to the Pyth Best listed on Pyth's website.

Recommendation: Consider implementing the recommendations.

MAR-2-8 Bypassing the Position Minimum Amount Limit

• Informational Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `4a6d83b7ce8e78720bbfd0452bbe0540dffd5a36`.

File(s) affected: `MarginlyPool.sol`

Description: The `positionMinAmount` parameter of a pool specifies the minimum base token amount to open a long or short position. Note that this minimum amount is a soft requirement, as users can always open a short position with a base token amount larger than the minimum amount, repay partial debt, and withdraw partial collateral from the position to reduce the position size. This can be done similarly with long positions.
Note that small underwater positions may de-incentivize liquidators to liquidate them as they need to pay a larger portion of gas fees, but the pool's built-in auto-liquidation mechanism can help liquidate them.

Recommendation: We mainly want to raise awareness for this possibility.

MAR-2-9 Missing Input Validation

• Informational Fixed

Update
Marked as "Fixed" by the client.
Addressed in: `7543ab090d46acacdf107beacf428c347110f0a7`. The second recommendation was acknowledged, which we find reasonable.

File(s) affected: `UniswapV3TickOracle.sol`, `UniswapV3TickOracleDouble.sol`, `AlgebraTickOracle.sol`, `AlgebraTickOracleDouble.sol`

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error:

1. In the `setOptions()` function of the `UniswapV3TickOracle`, `UniswapV3TickOracleDouble`, `AlgebraTickOracle`, and `AlgebraTickOracleDouble` contracts, consider adding a check to ensure `secondsAgoLiquidation <= secondsAgo` to avoid misconfiguration of the TWAP price intervals.

2. In the same function, consider enforcing a reasonable lower bound for the `secondsAgo` and `secondsAgoLiquidation` parameter.

Recommendation: Consider adding the checks as mentioned above.

MAR-2-10 Notes on Oracle Price Update Delays

Informational | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client.
> The client provided the following explanation:
>
> `We will update our documentation in oracle section`

Description: Chainlink and Pyth aggregate prices off-chain and update the price value periodically on-chain. By design this creates a minor price update delay that has different impacts on each protocol. It is worth noting that also using an on-chain TWAP can cause a similar delay if the `secondsAgo` is set to a sufficiently large value.

Below, we would like to outline how the slight price delays can impact Marginly:

1. A lender or a holder of an active position of the opposite side could avoid absorbing bad debt with their collateral by front-running a significant price update with a withdrawal or position close that would else cause net negative liquidations
2. Theoretically, users could benefit from slightly decreased APYs until the next oracle update, in case the last reported price causes leverage calculations to increase.
3. Theoretically, users could front run price updates by opening a highly leveraged position with the asset that is about to get the APY increased in its favor. As soon as the price update is recorded and the increased interest is accounted for in retrospect, the position could be closed.

It is worth highlighting that both of these scenarios are quite unlikely and unlikely to be profitable. The first one in a worst-case analysis of a maximally leveraged position that is almost underwater requires a price increase of more than 0.25% between two oracle updates for bad debt to accumulate, while the second and third one have a theoretical opportunity window of increased APY equal to the period of the price updates, which is maximally in the realm of minutes.

Recommendation: We mainly want to raise awareness of this possibility. Consider only using oracles that update the price with sufficient frequency.

Code Documentation

1. The code readability would benefit from an increased use of in-line comments as well as NatSpec comments.

Adherence to Best Practices

1. Fixed The casting of the `uint256(amount2)` parameter as part of the call to `receivePosition()` in the `execute()` function is unchecked to be of negative value. If that were to be the case, it could cast the value to an undesired amount, given that `int256` employs the two's complement. Consider adding a check for a negative value or employ OpenZeppelin's `SafeCast` library for it.
2. Fixed The `MarginlyFactory` imports `IPriceOracle` but is not used and can be removed.
3. Fixed `AlgebraOracleLib` and `OracleLib` import `TickMathLib` but do not use it, so the import can be removed.
4. Fixed The internal function `MarginlyPool._setParameters()` uses magic numbers; consider replacing them with named constant variables to improve code readability, maintainability, and reduce error likelihood by providing meaningful names and a centralized point of modification for values used throughout the code.
5. Fixed The linked source given in the `AlgebraOracleLib` contract does not seem to be fully correct, as the `getArithmeticMeanTick()` function seems to be based on the `DataStorageLibrary.consult()` function and not any function given in the `WeightedDataStorageLibrary.sol` file.
6. Fixed The `AlgebraTickOracleDouble.decode()` function is unused. Consider removing it.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.

- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Files

- `400...20d ./contracts/MarginlyKeeperUniswapV3.sol`
- `cc7...f76 ./contracts/FullMarginlyPool.sol`
- `32e...d2a ./contracts/MarginlyPool.sol`
- `12e...ce1 ./contracts/MarginlyKeeper.sol`
- `140...daa ./contracts/MarginlyFactory.sol`
- `aa4...ec1 ./contracts/FullMarginlyFactory.sol`
- `f75...3ad ./contracts/interfaces/IWETH9.sol`
- `630...eeb ./contracts/interfaces/IPriceOracle.sol`
- `6af...408 ./contracts/interfaces/IMarginlyPool.sol`
- `b48...cc6 ./contracts/interfaces/IMarginlyPoolOwnerActions.sol`
- `b6b...504 ./contracts/interfaces/IMarginlyFactory.sol`
- `ba5...aaa ./contracts/libraries/OracleLib.sol`
- `61e...d12 ./contracts/libraries/MaxBinaryHeapLib.sol`
- `aa5...665 ./contracts/libraries/Errors.sol`
- `1f5...987 ./contracts/libraries/FP48.sol`
- `ade...9df ./contracts/libraries/FP96.sol`
- `d9b...8e3 ./contracts/dataTypes/Mode.sol`
- `bf8...2c5 ./contracts/dataTypes/Position.sol`
- `c5b...b72 ./contracts/dataTypes/MarginlyParams.sol`
- `b07...61a ./contracts/dataTypes/Call.sol`

Tests

- `526...dce ./test/MarginlyPool.spec.ts`
- `927...255 ./test/MarginlyFactory.spec.ts`
- `084...974 ./test/MarginlyPool.Deleverage.spec.ts`
- `1e4...982 ./test/MarginlyPool.gas.spec.ts`
- `99c...015 ./test/MaxBinaryHeapTest.spec.ts`
- `165...d1a ./test/MarginlyKeeperAave.spec.ts`
- `d38...5c7 ./test/MarginlyKeeperUniswapV3.spec.ts`
- `fd1...5c0 ./test/SwapRouterSettings.spec.ts`
- `d62...f82 ./test/MarginlyPool.Shutdown.spec.ts`
- `7cc...a7c ./test/MarginlyPool.Liquidation.spec.ts`
- `bfb...b92 ./test/benchmarks/FixedPoint.bench.ts`
- `792...7a2 ./test/benchmarks/MaxBinaryHeapBenchmarks.bench.ts`
- `bf1...b5d ./test/shared/fixtures.ts`
- `eb3...f2d ./test/shared/utils.ts`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.10.0

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

349 results were found. Non-false positive findings have been included in the report.

Test Suite Results

Test output was obtained using `yarn test`. The tests for the TestSwapRouter are currently pending as part of this execution flow, which we recommend fixing.

Update Fix-Review

12 tests have been added that amongst other things validate the fixes of the identified issues.

```
·---------------------------|---------------------------------|---------------------------------·
|  Solc version: 0.8.19     ·  Optimizer enabled: true        ·  Runs: 100                      |
····························|·································|·································
|  Contract Name            ·  Deployed size (KiB) (change)   ·  Initcode size (KiB) (change)  |
····························|·································|·································
|  FullMarginlyFactory      ·                  26.263 ()      ·                  26.774 ()      |
····························|·································|·································
|  FullMarginlyPool         ·                  23.816 ()      ·                  24.290 ()      |
····························|·································|·································
|  MarginlyFactory          ·                   2.027 ()      ·                   2.543 ()      |
····························|·································|·································
|  MarginlyKeeper           ·                   4.429 ()      ·                   4.700 ()      |
····························|·································|·································
|  MarginlyKeeperUniswapV3  ·                   4.035 ()      ·                   4.066 ()      |
····························|·································|·································
|  MarginlyPool             ·                  23.635 ()      ·                  23.687 ()      |
·---------------------------|---------------------------------|---------------------------------·

Warning: 1 contracts exceed the size limit for mainnet deployment (24.000 KiB deployed, 48.000 KiB init).
✨  Done in 10.01s.
yarn run v1.22.19
$ UPDATE_SNAPSHOT=1 REPORT_GAS=true hardhat test ./test/*.spec.ts
·---------------------------|---------------------------------|---------------------------------·
|  Solc version: 0.8.19     ·  Optimizer enabled: true        ·  Runs: 100                      |
····························|·································|·································
|  Contract Name            ·  Deployed size (KiB) (change)   ·  Initcode size (KiB) (change)  |
····························|·································|·································
|  FullMarginlyFactory      ·                  26.263 (0.000) ·                  26.774 (0.000) |
····························|·································|·································
|  FullMarginlyPool         ·                  23.816 (0.000) ·                  24.290 (0.000) |
····························|·································|·································
|  MarginlyFactory          ·                   2.027 (0.000) ·                   2.543 (0.000) |
····························|·································|·································
|  MarginlyKeeper           ·                   4.429 (0.000) ·                   4.700 (0.000) |
····························|·································|·································
|  MarginlyKeeperUniswapV3  ·                   4.035 (0.000) ·                   4.066 (0.000) |
····························|·································|·································
|  MarginlyPool             ·                  23.635 (0.000) ·                  23.687 (0.000) |
·---------------------------|---------------------------------|---------------------------------·

Warning: 1 contracts exceed the size limit for mainnet deployment (24.000 KiB deployed, 48.000 KiB init).

  MarginlyFactory
    ✔ should create pool
    ✔ should change router address
```

```
    ✔ should create the same pools
    ✔ should raise error when trying to renounce ownership
    ✔ should raise error when trying to deploy factory with wrong arguments

  MarginlyKeeperAave
    ✔ Should liquidate short bad position
    ✔ Should liquidate long position
    ✔ Should fail when profit after liquidation less than minimum

  MarginlyKeeperUniswapV3
    ✔ Should liquidate short bad position
    ✔ Should liquidate long position
    ✔ Should fail when profit after liquidation less than minimum

  Deleverage
    ✔ Deleverage long position
    ✔ Deleverage short position
    ✔ short call after deleverage
    ✔ long call after deleverage
    ✔ depositQuote call after deleverage
    ✔ depositBase call after deleverage
    ✔ withdrawQuote call after deleverage
    ✔ withdrawBase call after deleverage
price is 198070406285660843983859875 84
    ✔ close long position after deleverage
    ✔ close short position after deleverage
    ✔ receive short position after deleverage, decreasing debt
    ✔ receive long position after deleverage, decreasing debt
    ✔ receive short position after deleverage, debt fully covered
    ✔ receive long position after deleverage, debt fully covered
    ✔ receive short position after deleverage, increasing collateral
    ✔ receive long position after deleverage, increasing collateral

MarginlyPool.Liquidation
    ✔ should revert when existing position trying to make liquidation
    ✔ should revert when position to liquidation not exists
    ✔ should revert when position to liquidation not liquidatable
    ✔ should revert when new position after liquidation of short will have bad margin
    ✔ should revert when new position after liquidation of long will have bad margin
    ✔ should create new position without debt after short liquidation
    ✔ should create new position without debt after long liquidation
    ✔ should create new short position after short liquidation
    ✔ should create new long position after long liquidation
    ✔ should create better short position after short liquidation
    ✔ should create better long position after short liquidation

mc heap tests
    ✔ remove long caller
    ✔ remove short caller

  MarginlyPool.Shutdown
    ✔ should revert when collateral enough
    ✔ unavailable calls reverted in emergency mode
    ✔ should switch system in ShortEmergency mode
    ✔ should switch system in LongEmergency mode
    ✔ should switch system in ShortEmergency mode: non-emergency pos with negative net
    ✔ should switch system in LongEmergency mode: non-emergency pos with negative net
pool state after withdraw: base=2 quote=1
    ✔ withdraw tokens for Long/Lend position in ShortEmergency mode
pool state after withdraw: base=1 quote=2
    ✔ withdraw tokens for Short/Lend position in LongEmergency mode
    ✔ should unwrap WETH to ETH when withdraw in Emergency mode
    ✔ should revert withdraw tokens from Short position in ShortEmergency mode
    ✔ should revert withdraw tokens from Long position in LongEmergency mode

  Open position:
    ✔ depositBase
    ✔ depositQuote

  Deposit into existing position:
    ✔ depositBase
    ✔ depositQuote
```

```
System initialized:
  ✔ long
  ✔ long with flip
  ✔ short
  ✔ short with flip
  ✔ depositBase
  ✔ depositBase and long
  ✔ depositQuote
  ✔ depositQuote and short
  ✔ closePosition
  ✔ withdrawBase
  ✔ withdrawQuote

mc happens:
  ✔ depositBase with one mc
  ✔ depositQuote with one mc
  ✔ short with one mc
  ✔ long with one mc
  ✔ long initialized heap with one mc
  ✔ long closePosition with one mc
  ✔ depositBase with two mc
  ✔ depositQuote with two mc
  ✔ short with two mc
    ✔ long with two mc
    ✔ closePosition with two mc
    ✔ MC long position with deleverage
    ✔ MC short position with deleverage
    ✔ MC long reinit
    ✔ MC short reinit

  Liquidation
    ✔ liquidate long position and create new position
    ✔ liquidate short position and create new position
    ✔ liquidate long position and create new long position
    ✔ liquidate short position and create new short position

  MarginlyPool.Base
    ✔ should revert when second try of initialization
    ✔ should revert when somebody trying to send value
    ✔ sweepETH should revert when sender is not admin
    ✔ sweepETH should be called by admin
    ✔ should set Marginly parameters by factory owner
    ✔ should raise error when not an owner trying to set parameters
    ✔ should raise error when trying to set invalid parameters
BigNumber { value: "19807040628566084398385987584" }
    ✔ should limit system leverage after long liquidation
    ✔ should limit system leverage after short liquidation
✔ systemLeverageShort update after caller MC: worst position
✔ systemLeverageShort update after caller MC: not worst position
✔ systemLeverageLong update after caller MC: worst position
✔ systemLeverageLong update after caller MC: not worst position
Deposit base
  ✔ zero amount
  ✔ exceeds limit
  ✔ first deposit should create position
  ✔ different signers deposits
  ✔ deposit into positive base position
  ✔ depositBase into short position
  ✔ depositBase into long position
  ✔ depositBase and open long position
  ✔ depositBase and open short position
  ✔ depositBase and long into short position
  ✔ depositBase should wrap ETH into WETH
Deposit quote
  ✔ zero amount
  ✔ exceeds limit
  ✔ first deposit should create position
  ✔ deposit into positive quote position
  ✔ deposit into short position
  ✔ deposit into long position
  ✔ depositQuote and open short position
```

```
    ✔ depositQuote and open long position
    ✔ depositQuote and short into long position
    ✔ depositQuote and short into short position
    ✔ depositQuote should wrap ETH to WETH
  Withdraw base
    ✔ should raise error when trying to withdraw zero amount
    ✔ should raise error when position not initialized
    ✔ should decrease base position
    ✔ withdraw with position removing
    ✔ withdrawBase should unwrap WETH to ETH
    ✔ should raise error when trying to withdraw from short position
    ✔ positionMinAmount violation
  Withdraw quote
    ✔ should raise error when trying to withdraw zero amount
    ✔ should raise error when position not initialized
    ✔ should decrease quote position
    ✔ reinit
    ✔ withdraw with position removing
    ✔ withdrawQuote should unwrap WETH to ETH
    ✔ should raise error when trying to withdraw from long position
  Close position
    ✔ should raise error when attempt to close Uninitialized or Lend position
    ✔ close short slippage fail
    ✔ close long slippage fail
    ✔ should close short position
    ✔ should close long position
    ✔ positionMinAmount violation
  Short
    ✔ short, Uninitialized
    ✔ short minAmount violation
    ✔ exceeds limit
    ✔ slippage fail
    ✔ should not exceed quoteLimit when deposit base cover debt
    ✔ could exceed quoteLimit when deposit base amount
    ✔ short should update leverageShort
    ✔ short, changed from lend to short
    ✔ short, update short position
  Long
    ✔ uninitialized
    ✔ long minAmount violation
    ✔ exceeds limit
    ✔ slippage fail
    ✔ should not exceed quoteLimit when deposit base cover debt
    ✔ could exceed quoteLimit when deposit quote amount
    ✔ long should update leverageLong
    ✔ changed from lend to long
    ✔ update long position
  Flip
    ✔ Short with flip
    ✔ Long with flip
  Position sort keys
    ✔ should properly calculate sort key for long position
    ✔ should properly calculate sort key for short position
    ✔ long position sortKey
    ✔ short position sortKey

MaxBinaryHeapTest
  ✔ should create empty heap
  ✔ should return false when peek root on empty heap
  ✔ should return isEmpty
  ✔ should add new item and rebuild tree
  ✔ should return success=false when trying to get index of not existed in the heap account
  ✔ should return success=false on empty heap
  ✔ should remove root item and rebuild tree
  ✔ should remove item by index
  ✔ should remove from last to top
  ✔ should remove arbitary element
  ✔ should remove element and update tree (heapifyUp)
  ✔ should remove element and update tree (heapifyDown)
  ✔ should create max binary heap and remove items in right order
  Should update heap by index
    ✔ without changing position
```

```
    ✔ from middle to top
    ✔ from middle to bottom
    ✔ from top to bottom
    ✔ from bottom to top
    ✔ should update node account by index

TestSwapRouter
  - swap base to quote exact input
  - swap quote to base exact input
  - swap base to quote exact output
  - swap quote to base exact output


183 passing (59s)
4 pending
```

Code Coverage

Coverage output was obtained using `yarn test:coverage`. The already good tests of the core contracts present in the last audit were improved to close to perfect coverage metrics. The fact that `FullMarginlyFactory.sol` and `FullMarginlyPool.sol` continue to be untested is acceptable, given their extremely limited added functionality. Due to the nature of the difficulty of testing oracles, it is hard to test. We however encourage the client to add some tests for the `MarginlyKeeper.sol` and `MarginlyKeeperUniswapV3.sol` contracts.

Update Fix-Review

The added tests increased coverage metrics slightly.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	96.46	81.68	91.25	94.67	
FullMarginlyFactory.sol	0	0	0	0	... 69,70,71,75
FullMarginlyPool.sol	0	0	0	0	... 44,45,46,47
MarginlyFactory.sol	100	72.73	100	100	
MarginlyKeeper.sol	96.97	66.67	100	97.22	107
MarginlyKeeperUniswapV3.sol	97.06	71.43	100	97.14	69
MarginlyPool.sol	98.13	89.68	100	97.64	... 9,1610,1611
contracts/dataTypes/	100	100	100	100	
Call.sol	100	100	100	100	
MarginlyParams.sol	100	100	100	100	
Mode.sol	100	100	100	100	
Position.sol	100	100	100	100	
contracts/interfaces/	100	100	100	100	
IMarginlyFactory.sol	100	100	100	100	
IMarginlyPool.sol	100	100	100	100	
IMarginlyPoolOwnerActions	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
.sol					
IPriceOracle.sol	100	100	100	100	
IWETH9.sol	100	100	100	100	
contracts/libraries/	53.75	31.25	62.07	62.79	
Errors.sol	100	100	100	100	
FP48.sol	100	100	100	100	
FP96.sol	50	50	52.63	60	... 127,131,159
MaxBinaryHeapLib.sol	100	91.67	100	100	
OracleLib.sol	0	0	0	0	... 64,65,67,72
contracts/test/	67.27	78.57	28.57	60.11	
FixedPointTest.sol	100	100	0	0	11,15,19
MaxBinaryHeapTest.sol	100	100	100	100	
MockAavePool.sol	36.36	75	2.27	36.36	... 197,201,205
MockAavePoolAddressesProvider.sol	11.11	100	10	20	... 46,52,58,64
MockMarginlyFactory.sol	0	100	16.67	33.33	24,31
MockMarginlyPool.sol	100	81.25	30	100	
MockPriceOracle.sol	100	100	77.78	85.71	19,23
MockSwapRouter.sol	100	100	60	100	
TestERC20Token.sol	100	50	100	100	
TestSwapRouter.sol	100	87.5	75	100	
TestUniswapFactory.sol	44.44	100	16.67	54.55	22,26,30,34,38
TestUniswapPool.sol	70.83	75	33.33	39.06	... 270,271,272
All files	87.68	73.41	52.61	85.18	

Changelog

- 2024-04-15 - Initial report
- 2024-04-25 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and and may not be represented as such. No third party is entitled to rely on the report in any any way, including for the purpose of making any decisions to buy or sell a product, product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or or any open source or third-party software, code, libraries, materials, or information to, to, called by, referenced by or accessible through the report, its content, or any related related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.