Quantstamp Security Assessment Certificate

QUANTSTAMP VERIFIED SECURITY CERTIFICATE

February 16th 2023 — Quantstamp Verified

UniBot (Diamond Protocol)

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

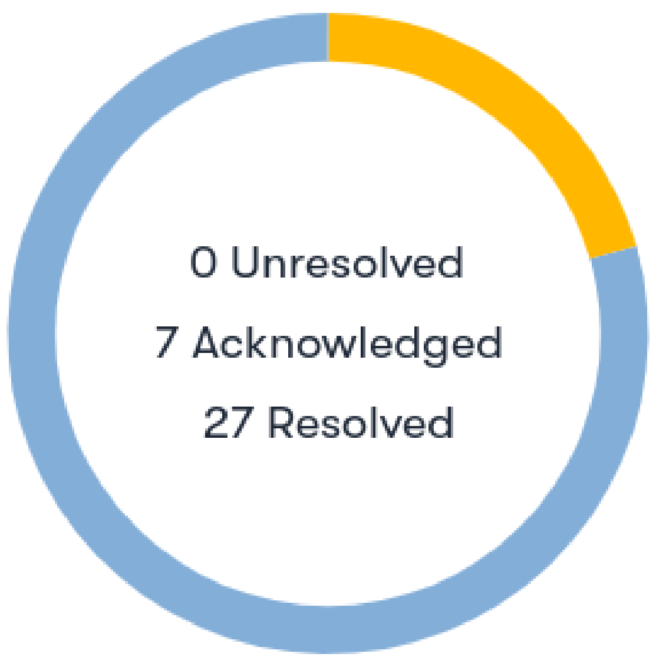
Executive Summary

ALL ISSUES ADDRESSED

Type	Leveraged Liquidity Provision Platform for Uniswap V3
Auditors	Faycal Lalidji, Senior Security Engineer Mostafa Yassin, Security Engineer Valerian Callens, Senior Research Engineer
Timeline	2022-10-31 through 2022-11-13
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Unibot Beta - Technical Document Whitepapter Public Documentation
Documentation Quality	Medium
Test Quality	Medium
Source Code	

Repository	Commit
ZooWallet/Diamond-Farm	f88f37f initial audit

Total Issues	34	(27 Resolved)
High Risk Issues	2	(2 Resolved)
Medium Risk Issues	9	(9 Resolved)
Low Risk Issues	6	(5 Resolved)
Informational Risk Issues	11	(8 Resolved)
Undetermined Risk Issues	6	(3 Resolved)



High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	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.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
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).
Fixed	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Initial audit:

We have raised 34 issues, ranging from high to undetermined severity, that need to be fixed before deployment.

Fix review:

All highlighted issues have been either fixed, mitigated, or acknowledged.

ID	Description	Severity	Status
QSP-1	Inappropriate Validation of Liquidation Fees Can Possibly Lead the Liquidity Pool to Be Drained	︽ High	Fixed
QSP-2	The Estimated `amountInWithSlippage` Can Lead `DiamondFactoryV3._trySwapExactOutput()` to Revert when Closing a Position	︽ High	Fixed
QSP-3	Possible Sandwich Attack when Opening or Closing a Position	^ Medium	Fixed
QSP-4	No Use of Re-Entrancy Guard or Respect of CEI Pattern	^ Medium	Fixed
QSP-5	Protocol Solvency Depends on the Availability and Reactivity of the Keepers	^ Medium	Fixed
QSP-6	Updates of Important Variables Are Not Always Checked and/or Logged	^ Medium	Mitigated
QSP-7	Automatic Max Approval when `lendingPool` Is Set and Updated	^ Medium	Fixed
QSP-8	Functions `init()` Vulnerable to Front-Running	^ Medium	Fixed
QSP-9	Infinite Loop if a User Has Too Many Positions	^ Medium	Fixed
QSP-10	Keeper Requirement Can Be Bypassed when Liquidating a Position or Executing a Stop Loss	^ Medium	Fixed
QSP-11	Factory Can Be Prevented From Repaying Borrowed Loans	^ Medium	Fixed
QSP-12	Incorrect Use of the Values Returned by Uniswap V3 Functions	⌄ Low	Acknowledged
QSP-13	Missing Integrity Checks when Updating Min and Max Limits	⌄ Low	Mitigated
QSP-14	Fee Recipient Cannot Be Updated in `LendingPool`	⌄ Low	Fixed
QSP-15	Use of Unchecked Arithmitic Operations	⌄ Low	Fixed
QSP-16	Allowance Removal	⌄ Low	Fixed
QSP-17	Allowing Changing Pool Address Represent a Risk	⌄ Low	Fixed
QSP-18	Optional Whitelisting Mechanism Has Drawbacks	○ Informational	Acknowledged
QSP-19	`Oracle.consult()` Can Revert for some Values of `oracleTimeWeightedSec`	○ Informational	Fixed
QSP-20	Functions `find()` and `get()` with Complex Logic	○ Informational	Fixed
QSP-21	Using Non-Standard ERC20 Can Disrupt Internal Accounting	○ Informational	Fixed
QSP-22	Collection of Performance Fees by the Protocol Not Logged	○ Informational	Fixed
QSP-23	Documentation Mismatch Between Interface and Implementation Contracts	○ Informational	Fixed
QSP-24	Confusing Names of Local Variables in `openPosition()`	○ Informational	Fixed
QSP-25	Tokens Can Get Frozen in Contracts	○ Informational	Fixed
QSP-26	Outdated Solidity Version	○ Informational	Acknowledged
QSP-27	Fixed Stop Loss Fee	○ Informational	Acknowledged
QSP-28	Missleading Comments in `DiamondFactoryV3._removeUniLiquidity()`	○ Informational	Fixed
QSP-29	Mismatch Between Documentation and Code for `lendingRate` Formula	? Undetermined	Acknowledged
QSP-30	Mismatch Between Documentation and Code for Whitelisted Factories	? Undetermined	Fixed
QSP-31	Time Management Specificities when Deploying on a Layer 2	? Undetermined	Acknowledged
QSP-32	Risks Related to Using a Proxy Pattern	? Undetermined	Acknowledged
QSP-33	`openPositionMinimumAmount` Cannot Be Updated Once Set	? Undetermined	Fixed
QSP-34	Variable `tokenDecimalDiff` Not Used in `DiamondFactoryV3`	? Undetermined	Fixed

Quantstamp Audit Breakdown

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

DISCLAIMER:
If the final commit hash provided by the client contains features that are not within the scope of the audit or an associated fix review, those features 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

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
    i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    iii. 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:
    i. 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.
    ii. 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, clarify, 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.

Findings

QSP-1 Inappropriate Validation of Liquidation Fees Can Possibly Lead the Liquidity Pool to Be Drained

Severity: *High Risk*

Status: Fixed

File(s) affected: `DiamondFactoryV3.sol`

Description: `DiamondFactoryV3._closePosition()` Liquidation fees are applied to the amounts returned by `DiamondFactoryV3._removeUniLiquidity()` which are equivalent to the initial want token deposit minus the reserve and the amount borrowed with leverage.
For example, assuming a liquidation fee of 5% and a 4x leverage with zero reserve ratio, the final fees applied on the original deposited amount turn out to be close to 20%, which is a high value.
The issue described can lead users if liquidation is allowed for non-keeper addresses (owners of the position e.g) or keepers to get back a value even higher than the initial deposit (worst case scenario).

Recommendation: We recommend strictly reviewing all important parameters to avoid such scenarios.

QSP-2 The Estimated `amountInWithSlippage` Can Lead `DiamondFactoryV3._trySwapExactOutput()` to Revert when Closing a Position

Severity: *High Risk*

Status: Fixed

File(s) affected: `DiamondFactoryV3.sol`

Description: `DiamondFactoryV3._trySwapExactOutput()` uses the average price tick (`oracleTimeWeightedSec = 900`) to estimate the maximum `quoteAmount` and can be the reason why the swap will possibly throw leading `DiamondFactoryV3._trySwapExactOutput()` to revert due to the requirement stated below:

```
require(_amountIn < amountInWithSlippage, "Slippage protect");
```

This will cause healthy positions to revert when closing them and possibly turn to unhealthy or even to bad positions. Please note that this issue depends on a random factor (the price volatility during the last `oracleTimeWeightedSec`), and it is hard to predict when a user will be able to claim his position again.

Recommendation: Using time-weighted average tick when calculating the maximum amount in for a swap can be problematic since if the token price changes rapidly during the last `timeWeightedAverageTick` it can cause the swap transaction to throw even if the position does not represent a bad debt. A good alternative will be to use chainlink since it has a more reliable instantaneous average price.

QSP-3 Possible Sandwich Attack when Opening or Closing a Position

Severity: Medium Risk

Status: Fixed

File(s) affected: `DiamondFactoryV3.sol`

Description: A common attack in DeFi is the sandwich attack. Upon observing a position opening or closing, an attacker can front-run the victim transaction by buying one of the assets, letting the victim execute the transaction, and then executing another trade after the victim by trading back the amount gained in the first trade.
In both `DiamondFactoryV3._provideUniLiquidity()` and `DiamondFactoryV3._removeUniLiquidity()` the following parameters are not set:

- When minting liquidity:
  - `MintParams.amount0Min`
  - `MintParams.amount1Min`
- When decreasing liquidity (burning):
  - `DecreaseLiquidityParams.amount0Min`
  - `DecreaseLiquidityParams.amount1Min`

As a slippage prevention mechanism, the official documentation of Uniswap V3 recommends that non-zero values should be used for the parameters `amount0Min` and `amount1Min` when using in production the functions:

- `NonfungiblePositionManager.mint()` https://docs.uniswap.org/protocol/guides/providing-liquidity/mint-a-position#calling-mint ;
- `NonfungiblePositionManager.decreaseLiquiditymint()` https://docs.uniswap.org/protocol/guides/providing-liquidity/decrease-liquidity#decrease-liquidity ;

This best practice is not respected when the functions are called by the functions `_provideUniLiquidity()` and `_removeUniLiquidity()`.

Recommendation: We recommend allowing the user to validate the minimum required amounts from the UI, both when opening or closing a position, and reflect these user inputs in the smart contract.

QSP-4 No Use of Re-Entrancy Guard or Respect of CEI Pattern

Severity: Medium Risk

Status: Fixed

File(s) affected: `LendingPoolV3.sol`, `BalanceVaultV3.sol`, `DiamondFactoryV3.sol`

Description: A reentrancy vulnerability is a scenario where an attacker can repeatedly call a function from itself, unexpectedly leading to potentially disastrous results. This is done through a function that does not respect Checks-Effects-Interactions.
The Checks-Effects-Interactions coding pattern is meant to mitigate any chance of other contracts manipulating the state of the blockchain in unexpected and possibly malicious ways before control is returned to the original contract. As the name implied, only after checking whether appropriate conditions are met and acting internally on those conditions should any external calls to, or interactions with, other contracts be done.
All the functions listed below need to have a re-entrancy guard set:

- `DiamondFactoryV3.liquidate()`
- `DiamondFactoryV3.stopLoss()`
- `DiamondFactoryV3.openPosition()`
- `DiamondFactoryV3.closePosition()`
- `DiamondFactoryV3.addCollateral()`
- `DiamondFactoryV3.decreaseCollateral()`
- `DiamondFactoryV3.collectFee()`
- `DiamondFactoryV3.updateStopLossPrice()`
- `BalanceVaultV3.deposit()`
- `BalanceVaultV3.withdraw()`
- `BalanceVaultV3.transferBalanceFromVault()`
- `LendingPoolV3.deposit()`
- `LendingPoolV3.withdraw()`

Even if a function seems re-entrancy safe due to its interaction with a trusted third-party contract (known token contract, e.g.), it should not be trusted.

Recommendation: We recommend using reentrancy guards and following the pattern checks-effects-interactions to prevent reentrancies in all user-accessible external and public functions.

QSP-5 Protocol Solvency Depends on the Availability and Reactivity of the Keepers

Severity: Medium Risk

Status: Fixed

File(s) affected: `DiamondFactoryController.sol`, `DiamondFactoryV3.sol`

Description: Keepers play an essential role in the protocol when enabled. They liquidate insolvent positions and activate stop losses when a price tick goes below or above stop loss limits defined by the owners of positions.
It implies that the protocol solvency highly depends on a limited set of addresses which could easily become single points of failure. On top of that, the function `removeKeeper()` is used to

remove a keeper. However, the fact that the last allowed keeper can be removed is not checked. As a result, the protocol can remain without any keeper.

Recommendation: Consider assessing if allowing a limited set of addresses to handle the liquidations and stop losses is a risk for the protocol. If it is, consider removing the restrictions for these actions. Also, assess the impact of remaining without any keeper. If this situation should be avoided, consider using a state variable storing the number of keepers currently allowed in the system, and make sure that it is not possible to remove the last active keeper without adding another one first.

QSP-6 Updates of Important Variables Are Not Always Checked and/or Logged

Severity: *Medium Risk*

Status: Mitigated

Description: At several locations, it is possible to set or update important state variables with dedicated functions. However, a check is not always made to make sure that the new value is correct, meaning between a minimum and a maximum value to prevent any disruption of the core functions of the protocol. In addition, an event is not always emitted. For example:

- `LendingPoolV3.setPerformanceFee()` should validate `_performanceFee` to be lower than BPS.
- `DiamondFactoryController.constructor()` does not validate any input.
- `DiamondFactoryController.setLiquidationBonus()` does not limit the liquidation bonus.
- `DiamondFactoryController.setMaxPositionNumber()` does not limit the `_maxPositionNumber`.
- `DiamondFactoryController.setPerformanceFee()` does not limit the `_performanceFee`.
- `DiamondFactoryController.setBalanceVaultAddress()` does not validate the `_balanceVault` address.
- `DiamondFactoryController.setHelperAddress()` does not validate the `_helper` address.
- `DiamondFactoryController.setFeeRecipient()` does not validate the `_feeRecipient`.
- `DiamondFactoryV3.init()` does not validate any of the inputs.
- `DiamondFactoryV3.setLiquidationThreshold()` does not set boundaries to the new liquidation threshold value.
- `DiamondFactoryV3.setBorrowRatioMax()` does not limit the max borrow rate in case of invalid input which represent a high risk.
- `DiamondFactoryV3.setReserveRatioMax()` does not validate if the reserve ratio is less than `BPS`.
- `DiamondFactoryV3.setStopLossFee()` does not limit the maximum fee amount.
- `DiamondFactoryV3.setController()` does not validate the controller address.
- `DiamondFactoryV3.setSlippage()` does not limit the slippage value.
- `DiamondFactoryHelper.getPositionTokenAmount()` does not validate if the pool address and the pool want token match.
- `DiamondFactoryController.addContractWhiteList()` has to validate that the input address is a contract address.
- `InterestModel.setNewInterest()` does not validate `_newThreshold`, `_newSlope1` and `_newSlope2`.

The severity is assessed as [Medium]:

- the impact is High. A wrong value could impact the core functions of the protocol, but it could generally be updated. However, logging such updates would allow the detection of any issue through off-chain monitoring.
- the likelihood is Low as these operations can only be initiated by the contract owner, and we can expect that double-checks are performed.

Recommendation: Consider assessing for each state variable if an update should be logged and if a range check must be done. Update the code accordingly.

Update: The issue is marked as mitigated since the recommended checks in the following functions were not implemented:

- `DiamondFactoryV3.setStopLossFee()`.
- `DiamondFactoryHelper.getPositionTokenAmount()`.

QSP-7 Automatic Max Approval when `lendingPool` Is Set and Updated

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `DiamondFactoryV3.sol`

Description: The lending pool used by the contract can be set and updated with the function `setLendingPoolAddress()`. Only the owner of the contract has the right to execute that function. However, if the lending pool is malicious or if it gets exploited, it gets the right to drain all the `borrowToken` owned by the contract.

Recommendation: Consider approving a limited amount when needed instead of approving an unlimited amount using `uint256(-1)`.

QSP-8 Functions `init()` Vulnerable to Front-Running

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `DiamondFactoryV3.sol`, `BalanceVaultV3.sol`

Description: The contracts `DiamondFactoryV3` and `BalanceVaultV3` have no constructor but have a function `init()`. As a result, once the contracts are deployed, everyone can call the function `init()` and become the owner of these contracts.

Recommendation: Since it is unclear if any upgradable proxy pattern is intended to be used, consider calling the function `init()` directly from the constructor or making sure that the contract deployment and the function `init()` are executed within the same block by the same address.

QSP-9 Infinite Loop if a User Has Too Many Positions

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `DiamondFactoryV3.sol`, `DiamondFactoryController.sol`

Description: The function `_findPositionIndex()` checks with a for loop if a user owns a given position based on its `_positionId`. However, the loop does not use the library `SafeMath` and uses an `uint8` variable which can store a maximum of 256 values. As a result, an overflow can happen on the variable `i`, which would lead to an infinite loop.
Such a situation can be mitigated with the state variable `maxPositionNumber` of the contract `DiamondFactoryController`, but there is no upper limit for its value. If `maxPositionNumber` is not set properly, closing a position that has an id higher than 255 will cause the transaction to throw.

Recommendation: Consider adding an upper limit for the variable `maxPositionNumber`, and using uint256 type for the variable `i`.

QSP-10 Keeper Requirement Can Be Bypassed when Liquidating a Position or Executing a Stop Loss

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `DiamondFactoryV3.sol`, `DiamondFactoryController.sol`

Description: If a particular position reaches a `healthFactor < 1` it can be liquidated through a call to `DiamondFactoryV3.liquidate()`. This function checks that a caller is registered as a keeper in `DiamondFactoryController` when `requireKeeper` flag is set.
However, a user can call `DiamondFactoryV3.closePosition()` on an unhealthy position he owns and bypass the keeper requirement imposed in `liquidate()`.

Recommendation: We recommend reverting the transaction if a position is unhealthy when executing `DiamondFactoryV3.closePosition()`, as the actual state will liquidate the position by calling `_closePosition(msg.sender, _positionId, true)`.

QSP-11 Factory Can Be Prevented From Repaying Borrowed Loans

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `LendingPoolV3.sol`

Description: `LendingPoolV3.repayBorrow()` checks if the caller (which should be factory) is in the whitelist. However, if a factory borrows from a lending pool, and then is removed from the whitelist, it will not be able to call `LendinPool.repayBorrow()`.

Recommendation: We recommend allowing any address that has a debt to execute `LendingPoolV3.repayBorrow()` regardless of the whitelisting mechanism.

QSP-12 Incorrect Use of the Values Returned by Uniswap V3 Functions

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `DiamondFactoryV3.sol`

Description: In the function `_removeUniLiquidity()`, the function `NonfungiblePositionManager.decreaseLiquidity()` is used. According to the documentation, it returns two parameters `amount0` and `amount1`, being respectively the amounts of `token0` and `token1` accounted for the position's tokens owed. However, these returned values are not checked.
https://docs.uniswap.org/protocol/reference/periphery/NonfungiblePositionManager#decreaseliquidity
Also, the function `NonfungiblePositionManager.collect()` is used. According to the documentation, the function `collect()` returns two parameters `amount0` and `amount1`, being respectively the amounts of fees collected in `token0` and `token1`. However, the code considers these returned values as the amounts of `token0` and `token1` removed from the pool.
https://docs.uniswap.org/protocol/reference/periphery/NonfungiblePositionManager#collect

Recommendation: Consider updating the code to correctly use the values returned by Uniswap V3 functions.

Update: The team acknowledged the issue "https://coinomo.notion.site/QSP-12-e95e61cf6e5e4a37b2f7896d7a9767e3".

QSP-13 Missing Integrity Checks when Updating Min and Max Limits

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `DiamondFactoryV3.sol`, `Position.sol`

Description: For three couples of variables, the fact that the max limit is greater than the min limit (and vice versa) is not checked in the functions `init()`, `_checkOpenPositionParams()`, and in the underlying setter functions. It is also the case in the function `update()` of the library `Positions`.
The couples are: - `stopLossLowerPriceTick` and `stopLossUpperPriceTick`; - `openPositionMinimumAmount` and `openPositionMaximumAmount`; - `borrowRatioMin` and `borrowRatioMax`;
Such inconsistencies could block the core mechanisms of the protocol.

Recommendation: Consider adding an integrity check for these variables.

Update: This issue is marked as mitigated since one more check is missing in `DiamondFactoryV3.init()` for `openPositionMaximumAmount` and `openPositionMinimumAmount.`

QSP-14 Fee Recipient Cannot Be Updated in `LendingPool`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `LendingPoolV3.sol`

Description: The state variable `feeReceipt` stores the address that will get the fees generated by the contract. Its value is set in the constructor. However, there is no function to update its value. As a result, if the address `feeReceipt` is compromised or blacklisted for the token want, the only way to react would be to set the variable `performanceFee` to 0.

Recommendation: Consider adding a method to update the state variable `feeReceipt` and emit an event when the transaction is executed successfully.

QSP-15 Use of Unchecked Arithmitic Operations

Severity: Low Risk

Status: Fixed

File(s) affected: `InterestModel.sol`

Description: `InterestModal` execute multiple arithmetic operations without the use of any safe math library or a solidity version greater than `0.8.0`. Unchecked arithmetic operations can lead to overflows and possible exploits.

Recommendation: Even if the executed operations seem safe, we highly recommend the use of Openzepplin `SafeMath` library or Solidity version greater than `0.8.0`.

QSP-16 Allowance Removal

Severity: Low Risk

Status: Fixed

File(s) affected: `DiamondFactoryV3.sol`

Description: `DiamondFactoryV3.setLendingPoolAddress()` approves the borrow token to be spent by the new pool but does not remove the allowance for the previous address. This represents a risk since a compromised pool will still be able to access possible funds on `DiamondFactoryV3` contract address even after removing them.

Recommendation: Reset the allowance of the previous pool to zero when setting a new pool address.

QSP-17 Allowing Changing Pool Address Represent a Risk

Severity: Low Risk

Status: Fixed

File(s) affected: `DiamondFactoryV3.sol`

Description: `DiamondFactoryV3.setLendingPoolAddress()` allows changing the lending pool address while the contract is deployed. This seems to introduce unnecessary risk since the new address state might not be the same as the previous one. As a consequence, users will likely not be able to withdraw their assets.

Recommendation: Check if this behavior is intended and remove the function if necessary.

QSP-18 Optional Whitelisting Mechanism Has Drawbacks

Severity: Informational

Status: Acknowledged

File(s) affected: `DiamondFactoryController.sol`

Description: The contract `DiamondFactoryController` uses an optional whitelisting mechanism to limit the addresses that can interact with some whitelisted functions. It is based on two data structures:

- the state `mapping(address=>bool) contractWhiteList`;
- a merkleTree with its root hash stored in the state variable `merkleRoot`;

1. One issue is that a MerkleTree data structure is used for access controls. Doing so has drawbacks:

    - for each update of the tree (address creation, update, removal), the variable `merkleRoot` must be updated in the contract;
    - the current tree state must be stored somewhere and maintained;
    - the list of authorized addresses is not visible on-chain;
    - an authorized address must know the current tree to build the parameter `_proof` needed to call a whitelisted function;

1. A second issue is that we can imagine an address being added to both data structures. Then, if the address needs to be removed, it should be removed from both data structures. If not, it would still be whitelisted in the second data structure.

Recommendation: One option to mitigate them is to only use the mapping `contractWhiteList` and rename it also to cover non-contract addresses. Another option is to implement mitigating measures for the different items described above.

Update: The team acknowledged the issue "https://coinomo.notion.site/QSP-18-0b0a6629de3347a5a0f29e222ead3248".

QSP-19 `Oracle.consult()` Can Revert for some Values of `oracleTimeWeightedSec`

Severity: Informational

Status: Fixed

File(s) affected: `DiamondFactoryV3.sol`

Description: The function `Oracle.consult()` is used to fetch time-weighted average ticks using Uniswap V3 oracle. It uses the state variable `oracleTimeWeightedSec` as a parameter. However, the call will revert if `oracleTimeWeightedSec` is equal to `0` or if it goes beyond the oldest observation recorded in the pool.

Recommendation: Consider keeping in mind these constraints and eventually adding integrity checks in the functions `init()` and `setOracleTimeWeightedSec()`.

QSP-20 Functions find() and get() with Complex Logic

Severity: Informational

Status: Fixed

File(s) affected: Positions.sol

Description: 1. The function find() of the library Positions: - returns true if a position exists; - reverts if a position doesn't exist;
As a result, it never returns false and reverts instead.

1. The function get() of the library Positions includes an optional check of ownership. However, it does not check if the position does exist. As a result, if the position does not exist, it is possible to:
    - return an empty position, if checkOwner == false;
    - revert with the incorrect reason "Invalid position owner", if checkOwner == true;

In both cases, a developer could be misled by the name of these functions if the library is reused in future versions.

Recommendation:

1. For the function find(), consider renaming the function as findOrRevert().
2. For the function get(), consider checking that the position exists, or extract the check of ownership in another function, or clearly describe the behavior of the function in a comment.

QSP-21 Using Non-Standard ERC20 Can Disrupt Internal Accounting

Severity: Informational

Status: Fixed

File(s) affected: BalanceVaultV3.sol

Description: In the contract BalanceVaultV3, four functions are used by the contract to receive or transfer funds: deposit(), withdraw(), transferBalanceFromVault(), transferBalanceToVault(). Tokens received are owned by the contract and the internal accounting of which address owns which amount of token is stored in the state variable balances. However, that state variable is updated based on the amount that is expected to be transferred, instead of based on what has been received. As a result, if the contract used an ERC20 where the amount to transfer is different than the amount received (greater or lower), the internal accounting of the contract would be broken and users could claim more or less than the actual amount of tokens owned by the contract.

Recommendation: Consider keeping in mind this issue when allowing a new ERC20 to be supported by the contract.

QSP-22 Collection of Performance Fees by the Protocol Not Logged

Severity: Informational

Status: Fixed

File(s) affected: DiamondFactoryV3.sol

Description: Performance fees are collected in the function collectFee(). No event is emitted with the number of performance fees that have been collected, which could negatively impact the off-chain monitoring of the protocol.

Recommendation: Consider assessing if the collection of performance fees should be logged. If yes, update the code accordingly.

QSP-23 Documentation Mismatch Between Interface and Implementation Contracts

Severity: Informational

Status: Fixed

File(s) affected: BalanceVaultV3.sol, IDiamondFactoryV3.sol, InterestModel.sol, IInterestModel.sol, LendingPoolV3.sol ILendingPoolV2.sol, LendingPoolV3.sol, ILendingPoolV2.sol, IBalanceVaultV3.sol, DiamondFactoryController.sol, IDiamondFactoryController.sol, DiamondFactoryV3.sol

Description: For several functions, there are differences between the documentation in the interface contract and the implementation contract. The description is sometimes more detailed in the interface and sometimes it is the inverse. The impact is that some integrity constraints could be added or ignored, which could ultimately break the interoperability expected from different contracts implementing the same interface. Also, there is no documentation in the interface IInterestModel.

Recommendation: Consider aligning the documentation of each contract pair, ensuring no constraint is added or removed.

QSP-24 Confusing Names of Local Variables in openPosition()

Severity: Informational

Status: Fixed

File(s) affected: DiamondFactoryV3.sol

Description: The names used for the local variables in the function openPosition() are close and also mix expected and actual amounts of both tokens. For example, the variables _strategyParams.wantTokenAmount and wantTokenAmount are very similar. Also, the variables usdcLiquidityAmount and wethLiquidityAmount have token names in their names, which is not recommended if the contract is meant to be used for other tokens than USDC and WETH. The impact is that it is hard to check that the function behaves as expected.

Recommendation: Consider renaming the variables.

QSP-25 Tokens Can Get Frozen in Contracts

Severity: Informational

Status: Fixed

File(s) affected: `All files`

Description: The contracts do not have specific functions to pull tokens or ETH sent on purpose or by mistake.

Recommendation: Add pulling functions accordingly while disallowing withdrawals for tokens used by the contract. For example, in `DiamondFactoryV3`when adding the pulling functions add checks to disallow `borrowToken` and `wantToken` from being withdrawn.

QSP-26 Outdated Solidity Version

Severity: Informational

Status: Acknowledged

Description: As security standards develop, so does the Solidity language. To stay up to date with current practices, it's important to use a recent version of Solidity and conventions.

Recommendation: Check the slither documentation for recommended versions of solidity.

Update: The team acknowledged the issue "https://www.notion.so/coinomo/QSP-26-bb08fb3471aa48c5b3a0a9402fc4ff50".

QSP-27 Fixed Stop Loss Fee

Severity: Informational

Status: Acknowledged

File(s) affected: `DiamondFactoryV3.sol`

Description: In `DiamondFactoryV3.stopLoss()` the fee to execute the operation for any position is fixed and is not a percentage of the position to be closed, meaning that a user with an amount lower or equal to the fee will not get any return value from his position being closed.

Recommendation: If this is intended behavior, it should be clearly communicated through user-facing documentation or the dapp UI.

Update: The team acknowledged the issue "https://coinomo.notion.site/QSP-27-7fa803869ae54477b131d97cd1ff1510".

QSP-28 Missleading Comments in `DiamondFactoryV3._removeUniLiquidity()`

Severity: Informational

Status: Fixed

File(s) affected: `DiamondFactoryV3.sol`, `DiamondFactoryHelper.sol`

Description: The following comments in `DiamondFactoryV3._removeUniLiquidity()` seems incorrect:

```
        // We charge performance fee based on uniswap fee, but when you removed liquidity
        // uniswap will not tell you how much fee do you earned. So we have to calculate
        // how much uniswap fee should we have first.
```

The whole `IDiamondFactoryHelper(_helper()).getPositionTokenAmount()` is executed to get the fees when only a single call to the line stated below will return the accumulated fees values:

```
        (uint256 fee0, uint256 fee1) = PositionValue.fees(
            INonfungiblePositionManager(UNI_POSITION_MANAGER),
            _positionId
        );
```

The comments are misleading since no real calculations are done to get the fee values.

Recommendation: We recommend removing the comments and executing `PositionValue.fees(INonfungiblePositionManager(UNI_POSITION_MANAGER), _positionId)` to get the fees instead of `IDiamondFactoryHelper(_helper()).getPositionTokenAmount()`.

QSP-29 Mismatch Between Documentation and Code for `lendingRate` Formula

Severity: Undetermined

Status: Acknowledged

File(s) affected: `InterestModel.sol`

Description: In the documentation (https://dmo-protocol.gitbook.io/diamond/unibot/pool), the formula to compute the Lending Rate includes a variable `Reserve Factor`. However, it does not appear in the function `getLendingRate()`.

Recommendation: Consider aligning the documentation and the code for the formula to calculate the lending rate.

Update: The team acknowledged the issue "https://www.notion.so/coinomo/QSP-29-4be93706124c4709a7218d895a423b8f".

QSP-30 Mismatch Between Documentation and Code for Whitelisted Factories

Severity: Undetermined

Status: Fixed

File(s) affected: `BalanceVaultV3.sol`

Description: In the contract `BalanceVaultV3`, it is possible to manage whitelisted addresses with the functions `addWhiteFactory()` and `removeWhiteFactory()`. However, the documentation of these functions states that an address is whitelisted for a given token and the function expects a parameter `_token`, which is not used in the code. Also, the state variable `whiteFactories` does not record which token should be whitelisted for a given factory.

Recommendation: Consider assessing if a factory should be whitelisted for a given token or all tokens. Then, align the documentation and the code accordingly.

QSP-31 Time Management Specificities when Deploying on a Layer 2

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `DiamondFactoryV3.sol`, `LendingPoolV3.sol`

Description: Different Layer 2 networks can have different ways to handle the values of block.number and block.timestamp. As a result, vulnerabilities can appear:

- For example, according to Uniswap V3 docs, the TWAP feature of a pool should be avoided on Optimism. https://docs.uniswap.org/protocol/concepts/V3-overview/oracle#oracles-integrations-on-layer-2-rollups
- Also, according to Arbitrum docs, the rule of thumb is the following: "As a general rule, any timing assumptions a contract makes about block numbers and timestamps should be considered generally reliable in the longer term (i.e., on the order of at least several hours) but unreliable in the shorter term (minutes). (It so happens these are generally the same assumptions one should operate under when using block numbers directly on Ethereum!)" https://developer.arbitrum.io/time.

The exact impact will depend on the chosen network, and it cannot be determined.

Recommendation: For each candidate Layer 2 network or chain, consider assessing how time is handled and what would impact the protocol, with a special focus on:

- the deadline parameters used when interacting with Uniswap V3;
- the calculation of interests in `LendingPoolV3`;
- the usage of TWAP oracles;

Update: The team acknowledged the issue "https://www.notion.so/coinomo/QSP-31-f23b298cf85745d5a0de069142463699".

QSP-32 Risks Related to Using a Proxy Pattern

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `DiamondFactoryV3.sol`, `LendingPoolV3.sol`

Description: During the audit, the team shared their intent to use a proxy pattern for the contracts `DiamondFactoryV3` and `LendingPoolV3`. Adding a proxy contract that is not audited could add challenges and vulnerabilities to the project:

- memory slot clashing;
- selector clashing;
- functions `init()` vulnerable to front-running;
- any upgrade of the implementation contract should deal with the fact that the current business logic of `DiamondFactoryV3` is already close to 24576 bytes;
- data should be stored in the proxy contract, not in the implementation contract;

Recommendation: Consider exploring all the measures to take before implementing a proxy pattern.

Update: The team acknowledged the issue "https://www.notion.so/coinomo/QSP-32-ad1f9a9ff1b14b408ab342886b7cb825".

QSP-33 `openPositionMinimumAmount` Cannot Be Updated Once Set

Severity: *Undetermined*

Status: Fixed

File(s) affected: `DiamondFactoryV3.sol`

Description: The state variable `openPositionMinimumAmount` acts as a lower limit for the number of tokens for a position. It is set in the function `init()`. However, there is no function to update it.

Recommendation: Consider assessing if it should be possible or not to update the state variable `openPositionMinimumAmount` and update the code accordingly.

QSP-34 Variable `tokenDecimalDiff` Not Used in `DiamondFactoryV3`

Severity: *Undetermined*

Status: Fixed

File(s) affected: `DiamondFactoryV3.sol`

Description: The state variable `tokenDecimalDiff` is set in the function `init()`. However, it is not used in the contract.

Recommendation: Consider double-checking the reason why that variable has been added to the contract.

Code Documentation

For the following items, the documentation could be improved:

- DiamondFactoryStorageV3: the measuring unit is not described for the state variables borrowRatioMax, borrowRatioMin, reserveRatioMax, stopLossFee.
- Positions: the measuring unit is not described for the state variable borrowRatio.
- DiamondFactoryV3: it is not stated in the documentation of the functions getHealthFactor(), canStopLoss(), canLiquidate() that they revert if no position identified by the parameter _positionId exists.

Adherence to Best Practices

1. Positions: the function update() plays two distinct roles, which could be split into two functions: updateAmount() and updateStopLossLimits().
2. InterestModel: typo in the name of the contract: InterestModal.
3. DiamondFactoryController: the instruction return passVerify; is redundant in both functions verifyWhitelist() and keeperOnlyCheck() because of the require statements above and the return statements below.
4. BalanceVaultV3: MerkleProof.sol is imported but not used.
5. LendingPoolV3: the fact that LendingPoolV3 inherits from ILendingPoolV2 and LendingPoolV2Storage is confusing.
6. LendingPoolV3: the field account of both events Borrow and RepayBorrow could be indexed to simplify off-chain monitoring.
7. LendingPoolV3: an integer overflow is technically possible in the function borrow(). However, it is very unlikely that it will happen due to the nature of the state variable _nextBorrowId.
8. DiamondFactoryV3: the function _notEmergency() can be removed and its code can be directly used in the modifier notEmergency().
9. DiamondFactoryV3: in the function _findPositionIndex(), the error descriptions of both require statements could be improved. The first could become "No position owned" and the second could become "Position not found".

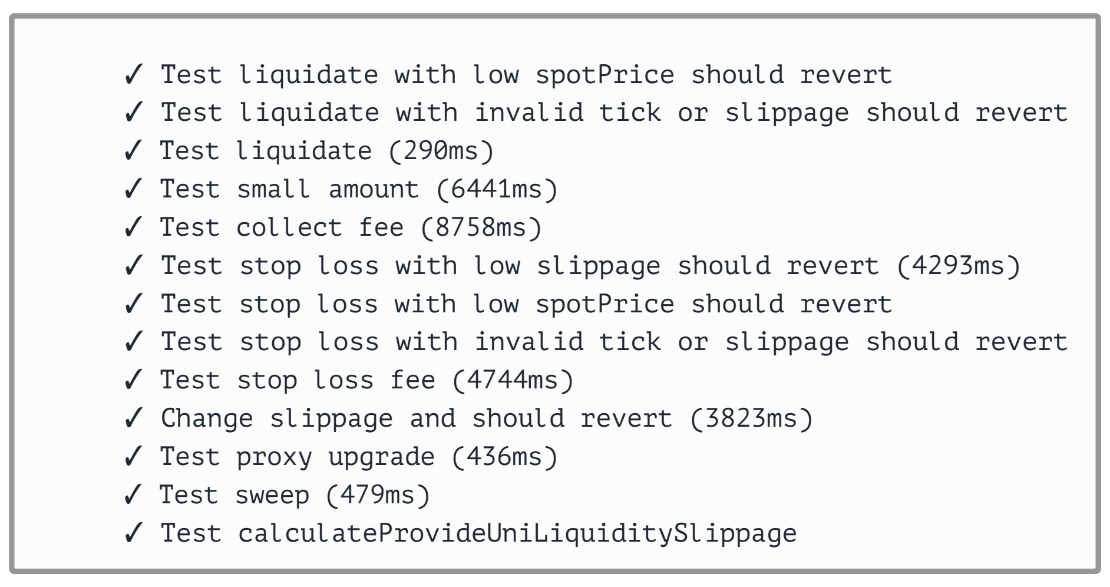
Test Results

Test Suite Results

```
BalanceVaultV3
   ✓ Test addSupportToken (49ms)
   ✓ Test removeSupportToken (240ms)
   ✓ Test removeSupportToken with not exist token (298ms)
   ✓ Test addWhiteFactory
   ✓ Test removeWhiteFactory (41ms)
   ✓ Test emergency (501ms)
   ✓ Test getAccountBalance
   ✓ Test deposit (86ms)
   ✓ Test deposit other token (1303ms)
   ✓ Test deposit small amount (64ms)
   ✓ Test user2 deposit (507ms)
   ✓ Test withdraw (52ms)
   ✓ Test withdraw other token (47ms)
   ✓ Test withdraw small amount (51ms)
   ✓ Test user2 withdraw small amount (45ms)
   ✓ Test approve factory (55ms)
   ✓ Test transferBalanceFromVault (150ms)
   ✓ Test transferBalanceFromVault with other token (91ms)
   ✓ Test transferBalanceToVault (192ms)
   ✓ Test transferBalanceToVault with other token (50ms)
   ✓ Test sweep function (41ms)
   ✓ Test deposit / withdraw with blackToken (109ms)
   ✓ Test transferBalanceFromVault / transferBalanceToVault with blackToken (137ms)
   ✓ Test proxy upgrade (537ms)

LendingPoolV3
  Check configuration
     ✓ Check deployed contract
  Setter
     ✓ setPerformanceFee (75ms)
     ✓ setFeeRecipient (46ms)
  Whitelist
     ✓ Add depositors to whitelist (68ms)
     ✓ Remove depositor 2 from whitelist
  Deposit and Withdraw
     ✓ Depositor 1 deposit and withdraw (3947ms)
     ✓ Depositor 1 & 2 deposit (886ms)
     ✓ Withdraw amount > balance (55ms)
     ✓ Get exchangeRate and check
     ✓ Get estimated exchange rate
     ✓ Get cash and check
  Borrow and Repay
     ✓ Borrow want token without being in whitelist
     ✓ Borrow want token (333ms)
     ✓ Check interest rate when utility rate < interestChangeThreshold
     ✓ Check interest rate when utility rate > interestChangeThreshold (1111ms)
     ✓ Borrow more tokens (40ms)
     ✓ Check total borrowed amount
     ✓ Repay some token for first borrow (355ms)
     ✓ Repay all left debt for first borrow (44ms)
     ✓ Mine blocks and increase interest (59ms)
     ✓ Repay small amount for second borrow (53ms)
     ✓ Accept other people repay borrow (686ms)
     ✓ Repay all debt (with repay amount > borrow amount) for second borrow (75ms)
     ✓ Borrower should able to repay borrow even when borrower is not in whitelist (140ms)
     ✓ Withdraw all token back (104ms)
     ✓ Test sweep (52ms)

FactoryV3
   ✓ Test DiamondBase setter (129ms)
   ✓ Test factory & controller setter (568ms)
   ✓ Get price tick
   ✓ Test emergency (296ms)
   ✓ Test user deposit (1558ms)
   ✓ Test open position with more than balance (555ms)
   ✓ Test open position with invalid ratio (52ms)
   ✓ Test open position with wrong stop tick
   ✓ Test contractWhitelist (52ms)
   ✓ Test open position with more than max allowed amount
   ✓ Test open position with amount lower than min amount
   ✓ Test open position with more than max allowed position number
   ✓ Test open position with invalid tick range (tickSpacing)
   ✓ Test open position with invalid price or slippage (84ms)
   ✓ SetOpenPositionMaximumAmount
   ✓ Test setOpenPositionMinimumAmount (40ms)
   ✓ Test user approve (60ms)
   ✓ User approve
   ✓ Test open position with invalid amount min (9523ms)
   ✓ Test open position with low slippage (2106ms)
   ✓ Test open position with low spotPriceTick (108ms)
   ✓ Open position  (4577ms)
   ✓ Test open second position with higher borrow ratio (3604ms)
   ✓ Test position info (4138ms)
   ✓ Test add collateral (253ms)
   ✓ Test decrease collateral (796ms)
   ✓ Test closePosition with low slippage should revert (2239ms)
   ✓ Test closePosition with low spotPrice should revert (217ms)
   ✓ Test closePosition with invalid tick or slippage should revert
   ✓ Test closePosition and should not have fee (649ms)
   ✓ Swap several time to gain fee (6487ms)
   ✓ Test closePosition and should have fee (1371ms)
   ✓ Test stopLoss (13388ms)
   ✓ Open position for later use (4012ms)
   ✓ Test updateStopLossPrice (72ms)
   ✓ Change time to accrue interest
   ✓ Should not closePosition and stopLoss when canLiquidate (924ms)
   ✓ Test liquidate with low slippage should revert (1425ms)
```



```
    ✓ Test liquidate with low spotPrice should revert
    ✓ Test liquidate with invalid tick or slippage should revert
    ✓ Test liquidate (290ms)
    ✓ Test small amount (6441ms)
    ✓ Test collect fee (8758ms)
    ✓ Test stop loss with low slippage should revert (4293ms)
    ✓ Test stop loss with low spotPrice should revert
    ✓ Test stop loss with invalid tick or slippage should revert
    ✓ Test stop loss fee (4744ms)
    ✓ Change slippage and should revert (3823ms)
    ✓ Test proxy upgrade (436ms)
    ✓ Test sweep (479ms)
    ✓ Test calculateProvideUniLiquiditySlippage
```

Code Coverage

Initial audit

Quantstamp usually recommends developers increase the branch coverage to 90% and above before a project goes live to avoid hidden functional bugs that might not be easy to spot during the development phase. For branch code coverage, the current targeted files by the audit achieve a good score but is still lower than the recommended value. Please note that the scores have been calculated using the highest returned values by solidity coverage for each test file.

Fix review

Some targeted files still show branch coverage lower than 90%.

LendingPoolV3 test:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	20.85	16.25	18.52	20.68	
BalanceVaultV3.sol	0	0	0	0	… 233,243,244
DiamondFactoryController.sol	0	0	0	0	… 253,254,256
DiamondFactoryV3.sol	0	0	0	0	… 2,1579,1580
InterestModel.sol	66.67	62.5	66.67	66.67	37,38,39,40,80
LendingPoolV3.sol	100	95.45	100	100	
helpers/	0	0	0	0	
DiamondFactoryHelper.sol	0	0	0	0	… 130,132,136
libraries/	0	0	0	0	
Positions.sol	0	0	0	0	… 114,115,117
storage/	100	100	100	100	
DiamondFactoryStorageV3.sol	100	100	100	100	
LendingPoolV3Storage.sol	100	100	100	100	
All files	**19.4**	**14.94**	**17.09**	**19.22**	

DiamondFactoryV3 test:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	85.63	65	85.19	85.74	
BalanceVaultV3.sol	75.93	47.62	75	76.36	… 192,243,244
DiamondFactoryController.sol	92.31	83.33	93.75	92.31	81,88,141,142
DiamondFactoryV3.sol	93.93	82.69	96.36	93.99	… 3,1579,1580
InterestModel.sol	33.33	37.5	33.33	33.33	… 80,82,83,84
LendingPoolV3.sol	70.97	36.36	61.11	70.97	… 231,332,333
helpers/	95	50	100	95.24	
DiamondFactoryHelper.sol	95	50	100	95.24	56
libraries/	94.12	91.67	100	94.12	
Positions.sol	94.12	91.67	100	94.12	83
storage/	100	100	100	100	
DiamondFactoryStorageV3.sol	100	100	100	100	
LendingPoolV3Storage.sol	100	100	100	100	
All files	**86.25**	**66.67**	**86.32**	**86.38**	

BalanceVault test:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	10.93	22.5	14.81	11.04	
BalanceVaultV3.sol	100	85.71	100	100	
DiamondFactoryController.sol	0	0	0	0	… 253,254,256
DiamondFactoryV3.sol	0	0	0	0	… 2,1579,1580
InterestModel.sol	0	0	0	0	… 80,82,83,84
LendingPoolV3.sol	0	0	0	0	… 323,332,333
helpers/	0	0	0	0	
DiamondFactoryHelper.sol	0	0	0	0	… 130,132,136
libraries/	0	0	0	0	
Positions.sol	0	0	0	0	… 114,115,117
storage/	100	100	100	100	
DiamondFactoryStorageV3.sol	100	100	100	100	
LendingPoolV3Storage.sol	100	100	100	100	
All files	**10.17**	**20.69**	**13.68**	**10.26**	

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.

Contracts

```
9da14c510edb61bdf624c04bfb90bdfdd3684254191f5424b747521d5831d8b0 ./contracts/v3/storage/DiamondFactoryStorageV3.sol
715707fe3c02059b37ac6ca13fb880c4b8ed79b104bcf51e1d69cd2b8d76a54c ./contracts/v3/storage/LendingPoolV2Storage.sol
dffc1ce13d82e88424a3f075737fbb0ec13cb71c928d3b4d3082b5940c5e0dcc ./contracts/v3/helpers/DiamondFactoryHelper.sol
da555193da618c7c733fbdf53212a921439eb46b131fef6773891763cda2c33f ./contracts/v3/interfaces/IDiamondFactoryHelper.sol
4f201004679437143746e08aac0653e68bb1f8178c3352cac2c6fcf439c89c16 ./contracts/v3/interfaces/IInterestModel.sol
bf90178030bd70b28481448d2147c4a01a103c22c7b2875fadb9b4749031c2a0 ./contracts/v3/interfaces/IDiamondFactoryController.sol
5ffe0be3c93098096f3df5d5eb319e80bf84e65b394a11c5c00c853cf9920b9b ./contracts/v3/interfaces/IDiamondFactoryV3.sol
9c9fe082036b4d0035c4b7bb9651c7541a259f76a9feae252c9cb80b0269ee32 ./contracts/v3/interfaces/ILendingPoolV2.sol
447643596e0455db46f25371a4147fa5e8f19573851f9173740e09c538a4d135 ./contracts/v3/interfaces/IBalanceVaultV3.sol
5f776407bf5f592afd9f82c97f5c86898775e12fde58f01162985234f307f6ab ./contracts/v3/contracts/DiamondFactoryV3.sol
3924d448ccaa1e6cac116a1b297457acd6dfdcd7b1589685f97d327c343c702b ./contracts/v3/contracts/DiamondFactoryController.sol
2fdb8757e2a32bbf49baa748623114faac2a3ad78830ea13fe7d3d4e4d6ba9bc ./contracts/v3/contracts/LendingPoolV3.sol
263f9529975678f2d54475c451a8a3d2d6af507b3aab9c96db82d4d38e449aaa ./contracts/v3/contracts/BalanceVaultV3.sol
3a7371dd5d17bf4d4541bc2dddd77bcb3fe1982a792241fcd66009ba1f44e89c ./contracts/v3/contracts/InterestModel.sol
0b58d061a601e228c36d3d816d30166050b5c94dc5ba13af8c6ef09105e643e2 ./contracts/v3/libraries/Positions.sol
```

Tests

```
0cc1f3f09d5de3fac08e0bb92d204e5b1a8a10382a5377ddf228b55f6def3238 ./test/unit-tests/LendingPoolV3.test.js
f80ef5b621b9200e581e6da660c766bcf888cc1233613fd9752d4e60c191c9c8 ./test/unit-tests/DiamondFarmAction.js
9be096b2d7e5e3c59a4fd3456d6fb914eb42bcafd35aa45c39894b278bea88b2 ./test/unit-tests/DiamondFactoryV3.test.js
aa23bc35a08bec9c8841b531a0e23f5e1f04371e3614633e8fd6a661746fd472 ./test/unit-tests/BalanceVault.test.js
```

Changelog

- 2022-11-13 - Initial report
- 2022-12-16 - Fix review

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, Aave, 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 or 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 your 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 other 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 may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a 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 any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any 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 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.