OBELISK

Part of Tibereum Group

AUDITING REPORT

Version Notes

Version	No. Pages	Date	Revised By	Notes
1.0	Total: 63	2023-01-24	Plemonade, Donut	Audit Final

Audit Notes

Audit Date	2022-11-08 - 2023-01-24
Auditor/Auditors	Plemonade, DoD4uFN
Auditor/Auditors Contact Information	contact@obeliskauditing.com
Notes	Specified code and contracts are audited for security flaws. UI/UX (website), logic, team, and tokenomics are not audited.
Audit Report Number	OB515474124

Disclaimer

This audit is not financial, investment, or any other kind of advice and is for informational purposes only. This report is not a substitute for doing your own research and due diligence. Obelisk is not responsible or liable for any loss, damage, or otherwise caused by reliance on this report for any purpose. Obelisk has based this audit report solely on the information provided by the audited party and on facts that existed before or during the audit being conducted. Obelisk is not responsible for any outcome, including changes done to the contract/contracts after the audit was published. This audit is fully objective and only discerns what the contract is saying without adding any opinion to it. The audit is paid by the project but neither the auditors nor Obelisk has any other connection to the project and has no obligations other than to publish an objective report. Obelisk will always publish its findings regardless of the outcome of the findings. The audit only covers the subject areas detailed in this report and unless specifically stated, nothing else has been audited. Obelisk assumes that the provided information and material were not altered, suppressed, or misleading. This report is published by Obelisk, and Obelisk has sole ownership of this report. Use of this report for any reason other than for informational purposes on the subjects reviewed in this report including the use of any part of this report is prohibited without the express written consent of Obelisk. In instances where an auditor or team member has a personal connection with the audited project, that auditor or team member will be excluded from viewing or impacting any internal communication regarding the specific audit.

Obelisk Auditing

Defi is a relatively new concept but has seen exponential growth to a point where there is a multitude of new projects created every day. In a fast-paced world like this, there will also be an enormous amount of scams. The scams have become so elaborate that it's hard for the common investor to trust a project, even though it could be legit. We saw a need for creating high-quality audits at a fast phase to keep up with the constantly expanding market. With the Obelisk stamp of approval, a legitimate project can easily grow its user base exponentially in a world where trust means everything. Obelisk Auditing consists of a group of security experts that specialize in security and structural operations, with previous work experience from among other things, PricewaterhouseCoopers. All our audits will always be conducted by at least two independent auditors for maximum security and professionalism.

As a comprehensive security firm, Obelisk provides all kinds of audits and project assistance.

Audit Information

The auditors always conducted a manual visual inspection of the code to find security flaws that automatic tests would not find. Comprehensive tests are also conducted in a specific test environment that utilizes exact copies of the published contract.

While conducting the audit, the Obelisk security team uses best practices to ensure that the reviewed contracts are thoroughly examined against all angles of attack. This is done by evaluating the codebase and whether it gives rise to significant risks. During the audit, Obelisk assesses the risks and assigns a risk level to each section together with an explanatory comment. Take note that the comments from the project team are their opinion and not the opinion of Obelisk.

Table of Contents

Project Information

Name	Level Finance
Description	A Decentralized Perpetual Exchange with Functional Risk Management and Innovative Liquidity Solutions.
Website	https://level.finance/
Contact	https://twitter.com/Level__Finance
Contact information	@sonic_level on TG
Token Name(s)	N/A
Token Short	N/A
Contract(s)	See Appendix A
Code Language	Solidity
Chain	BNB

Audit of Level Trading

Obelisk was commissioned by Level Finance on the 6th of November 2022 to conduct a comprehensive audit of Levels' contracts. The following audit was conducted between the 8th of November 2022 and the 24th of January 2023. The long audit was due to multiple iterations of the contracts, which needed development time from the project. Two of Obelisk's security experts went through the related contracts manually using industry standards to find if any vulnerabilities could be exploited by the project team or users.

The auditors found multiple issues with the initial contracts. The project team has worked on solving these issues and managed to close most of the issues found. There are still 2 high-risk issues open, issues #3 and #5 that could have an impact on the project depending on the situation. Please read the notes on these issues below. Issue #9 is related to currently not having a sufficient timelock in place but could be closed as soon as a 72h timelock is added. Looking at medium severity issues in the contract, there are still two of them open, #10 and #18 which both have attached notes.

NOTE: The deployed contracts differ from the audited contracts which means that we have not audited the changes made from these audited contracts and the ones that the team deployed.

The informational findings are good to know while interacting with the project but don't directly damage the project in its current state, hence it's up to the project team if they deem that it's worth solving these issues, however, please take note of them.

The team has not reviewed the UI/UX, logic, team, or tokenomics of the Level Finance project**.**

This document is a summary of the findings that the auditors found. Please read the full document for a complete understanding of the audit.

Summary Table

Code Analysis

Finding	ID	Severity	Status
Anyone Can Call burnFrom For An Address	#0001	High Risk	Closed
Liquidatable Positions	#0002	High Risk	Closed
No Max Capacity On Swaps	#0003	High Risk	On-Chain
Fee Based On Another Token Price	#0004	High Risk	Closed
Missing Contracts And Functions	#0005	High Risk	On-Chain
Possible Implementation Errors In _calcTrancheSharesAmount	#0006	High Risk	Closed
Token Functionality In Pool Can Be Broken	#0007	High Risk	Closed
No Limit For Protocol Values	#0008	High Risk	Closed
No Timelock When Changing Protocol Values	#0009	High Risk	On-Chain
Max Position Bypass	#0010	Medium Risk	Open
Unbounded Loop	#0011	Low Risk	Partially Closed
Redundant Code	#0012	Informational	Closed
Minor Issues	#0013	Informational	Closed
Token Cannot Be Relisted	#0014	Informational	Closed
Tranches Are Not Completely Isolated	#0015	Informational	Open
Denial Of Service(DOS) On Withdrawal	#0016	High Risk	Closed
Withdraw Cooldown Can Be Bypassed	#0017	Medium Risk	Closed
No Max Position Size	#0018	Medium Risk	Open
Rounding Error In	#0019	Low Risk	Open

_rebalanceTranches 1			
Rounding error in _rebalanceTranches 2	#0020	Informational	Open
Pool Amount Discrepancy	#0021	High Risk	Closed
Average Short Discrepancy	#0022	High Risk	Closed
Stablecoin Value Hardcoded	#0023	Informational	Open

On-Chain Analysis

Finding	ID	Severity	Status
Changes To Deployed Contracts	#0024	Unknown	Open
Unverified Contract	#0025	Unknown	Closed
Timelock Delay Is Short	#0026	Medium Risk	Open

Findings

Code Analysis

Anyone Can Call burnFrom For An Address

FINDING ID	#0001
SEVERITY	High Risk
STATUS	Closed
LOCATION	LPToken.sol -> 32-34

```
1     function burnFrom(address _account, uint256 _amount) public
  override {
2         require(lastMinted[_account] + redeemCooldown <=
  block.timestamp, "LPToken: redemption delayed");
3         _burn(_account, _amount);
4     }
```

DESCRIPTION	Anyone is able to call *burnFrom()* on an address. This means that a malicious actor can burn someone else's lp tokens as long as as it fulfills the *lastMinted* + *redeemCooldown* condition
RECOMMENDATION	Add a check to make sure only *pool.sol* is able to burn from an address.
RESOLUTION	The project team now makes use of OpenZeppelin's *ERC20Burnable.sol* and the *burnFrom()* function.

Liquidatable Positions

FINDING ID	#0002
SEVERITY	High Risk
STATUS	Closed
LOCATION	Pool.sol -> 1148-1190

```
    function _calcDecreasePayout(
        Position memory _position,
        address _indexToken,
        address _collateralToken,
        Side _side,
        uint256 _sizeChanged,
        uint256 _collateralChanged
    )
        internal
        view
        returns (DecreasePositionVars memory vars)
    {
        // ...

        vars.collateralReduced =
            _position.collateralValue < _collateralChanged ||
_position.size == _sizeChanged
            ? _position.collateralValue
            : _collateralChanged;

        // ...

        vars.remainingCollateral = remainingCollateral.isNeg() ? 0 :
remainingCollateral.abs;

        // ...
    }
```

LOCATION	Pool.sol -> 382-419

```
1     function liquidatePosition(address _account, address _indexToken,
  address _collateralToken, Side _side) external {
2         _requireValidTokenPair(_indexToken, _collateralToken, _side,
  false);
3         _accrueInterest(_collateralToken);
4         bytes32 key = _getPositionKey(_account, _indexToken,
  _collateralToken, _side);
5         Position memory position = positions[key];
6         if (address(positionHook) != address(0)) {
7             positionHook.preDecreasePosition(_account, _indexToken,
  _collateralToken, _side, position.size, bytes(""));
8         }
9         DecreasePositionVars memory vars =
10            _calcDecreasePayout(position, _indexToken,
  _collateralToken, _side, position.size, position.collateralValue);
11
12        if (vars.remainingCollateral > fee.liquidationFee) {
13            revert PoolErrors.PositionNotLiquidated(key);
14        }
15        uint256 liquidationFee = fee.liquidationFee /
  vars.collateralPrice;
16        _releasePoolAsset(vars, _indexToken, _collateralToken, _side,
  liquidationFee);
17
18        emit LiquidatePosition(
19            key,
20            _account,
21            _collateralToken,
22            _indexToken,
23            _side,
24            position.size,
25            position.collateralValue - vars.remainingCollateral,
26            position.reserveAmount,
27            vars.indexPrice,
28            vars.pnl,
29            vars.feeValue
30            );
31
32        delete positions[key];
33        _doTransferOut(_collateralToken, msg.sender, liquidationFee);
34
35        if (address(positionHook) != address(0)) {
36            positionHook.postDecreasePosition(_account, _indexToken,
  _collateralToken, _side, position.size, bytes(""));
37        }
38    }
```

DESCRIPTION	Anyone is able to liquidate any position. This is because the calculation for *remainingCollateral* takes the *collateralValue - (a value corresponding to collateralValue)*.

	The calculation for *remaining* variable in the case of a call from the function *liquidatePosition()* will then look like this: *remainingCollateral = _position.collateralValue - _position.collateralValue* This will then pass the revert statement shown below as *vars.remainingCollateral* is 0, which is less than *fee.liquidationFee*. *if (vars.remainingCollateral > fee.liquidationFee) {* *revert PoolErrors.PositionNotLiquidated(key);* *}*
RECOMMENDATION	Make sure positions are only liquidatable once they fall below a certain threshold and especially not when they have positive PNL.
RESOLUTION	The project team reworked the *liquidatePosition()* function and now there are liquidation checks in place. Note: if tranche asset max leverage capacity is added and exceeded partial liquidations should be done in order for the protocol to not take on too much risk.

No Max Capacity On Swaps

FINDING ID	#0003
SEVERITY	High Risk
STATUS	On-Chain
LOCATION	Pool.sol -> 210-234

```
1     function swap(address _tokenIn, address _tokenOut, uint256
  _minOut, address _to)
2         external
3         nonReentrant
4         onlyListedToken(_tokenIn)
5         onlyListedToken(_tokenOut)
6     {
7         if (_tokenIn == _tokenOut) {
8             revert PoolErrors.SameTokenSwap(_tokenIn);
9         }
10        _accrueInterest(_tokenIn);
11        _accrueInterest(_tokenOut);
12        uint256 amountIn = _getAmountIn(_tokenIn);
13        if (amountIn == 0) {
14            revert PoolErrors.ZeroAmount();
15        }
16        (uint256 amountOut, uint256 swapFee) =
  _calcSwapOutput(_tokenIn, _tokenOut, amountIn);
17        uint256 amountOutAfterFee = amountOut - swapFee;
18        if (amountOutAfterFee < _minOut) {
19            revert PoolErrors.SlippageExceeded();
20        }
21        poolTokens[_tokenOut].feeReserve += swapFee;
22        _rebalanceTranches(_tokenIn, amountIn, _tokenOut,
  amountOutAfterFee);
23        _doTransferOut(_tokenOut, _to, amountOutAfterFee);
24        emit Swap(msg.sender, _tokenIn, _tokenOut, amountIn,
  amountOutAfterFee, swapFee);
25    }
```

DESCRIPTION	There is no max capacity for a token to be swapped. This means that an exploited token can be used to drain every other token.
RECOMMENDATION	Add a max capacity for each token such that this cannot occur.
RESOLUTION	This is intended according to the project team: "The zero-slippage swap is our business decision, so the max pool swap is intended. We also encourage users to add

more tokens to the pool when its liquidity is low (via fee discount)."

This issue isn't about that zero-price impact but rather about the max amount you can swap. If a token oracle ever reports a manipulated price there is a risk of vault draining.

Fee Based On Another Token Price

FINDING ID	#0004
SEVERITY	High Risk
STATUS	Closed
LOCATION	Pool.sol -> 664-679

```
1     function _calcSwapOutput(address _tokenIn, address _tokenOut,
  uint256 _amountIn)
2         internal
3         view
4         returns (uint256 amountOut, uint256 feeAmount)
5     {
6         uint256 priceIn = _getPrice(_tokenIn);
7         uint256 priceOut = _getPrice(_tokenOut);
8         uint256 valueChange = _amountIn * priceIn;
9         uint256 poolValue = _getPoolValue();
10        uint256 feeIn = _calcAdjustedFee(poolValue, _tokenIn,
  priceIn, valueChange, true);
11        uint256 feeOut = _calcAdjustedFee(poolValue, _tokenOut,
  priceOut, valueChange, false);
12        uint256 _fee = feeIn > feeOut ? feeIn : feeOut;
13
14        amountOut = valueChange / priceOut;
15        feeAmount = (valueChange * _fee) / priceOut / FEE_PRECISION;
16    }
```

LOCATION	Pool.sol -> 791-815

```
    function _calcAdjustedFee(
        uint256 _poolValue,
        address _token,
        uint256 _tokenPrice,
        uint256 _valueChange,
        bool _isSwapIn
    )
        internal
        view
        returns (uint256)
    {
        if (_poolValue == 0) {
            return 0;
        }
        uint256 targetValue = (targetWeights[_token] * _poolValue) / totalWeight;
        uint256 currentValue = _tokenPrice * poolTokens[_token].poolBalance;
        if (currentValue == 0) {
            return 0;
        }
        uint256 nextValue = _isSwapIn ? currentValue + _valueChange : currentValue - _valueChange;
        (uint256 baseSwapFee, uint256 taxBasisPoint) =
            isStableCoin[_token]
            ? (fee.stableCoinBaseSwapFee, fee.stableCoinTaxBasisPoint)
            : (fee.baseSwapFee, fee.taxBasisPoint);
        return _calcAdjustedFee(targetValue, currentValue, nextValue, baseSwapFee, taxBasisPoint);
    }
```

DESCRIPTION	In *_calcSwapOutput*, the value of *feeOut* is calculated using *valueChange*, which is based on the amountIn pricing. When later used in the *_calcAdjustedFee()* it applies the *valueChange* for *tokenIn* (= *amountIn*priceIN*) to the *currentValue* of *tokenOut* (= *priceOut*poolbalance*) to calculate *nextValue*. This can lead to the incorrect calculation of fees.
RECOMMENDATION	Don't mix prices and fees for different tokens.
RESOLUTION	**Project team comment:** "Modify swap function, now swap fee charged in tokenIn, so the `fee_amount = amount_in * price_in * fee_rate` which doesn't rely on price_out. Please note that the fee_rate is calculated using

price_out, but it will never exceed swap_fee + tax_basis_point, so the risk of manipulation is low."

Obelisk comment: "The swap fee is charged in both *amountIn - swapFee* and also in *amountOutAfterFee = valueChange * (FEE_PRECISION - _fee) / priceOut / FEE_PRECISION;*. Is this intended?

This approach can be used, but instead of having the *_calcSwapOutput()* do calculate both *swapFee* and *amountAfterFee*, decouple by first calculating the *swapFee* so the fee is already applied to *amountIn* before calling *_calcSwapOutput()*."

Project team comment: "Yes, it's intended. We try to combine the computation to reduce the rounding errors when using a result from division. I think this is a problem of clarity, not a high risk."

Missing Contracts And Functions

FINDING ID	#0005
SEVERITY	High Risk
STATUS	On-Chain
LOCATION	Pool.sol -> 593-596

```
1    function setPositionHook(address _hook) external onlyOwner {
2        positionHook = IPositionHook(_hook);
3        emit PositionHookChanged(_hook);
4    }
```

LOCATION	OrderManager.sol -> 93-105

```
1    function placeSwapOrder(
2        IPool _pool,
3        address _tokenIn,
4        address _tokenOut,
5        uint256 _amountIn,
6        uint256 _minOut,
7        uint256 _price
8    ) external payable nonReentrant {
9        require(
10   IWhitelistedPool(address(_pool)).whitelistedTokens(_tokenIn) &&
11   IWhitelistedPool(address(_pool)).whitelistedTokens(_tokenOut),
12           "Invalid tokens"
13       );
14
15       // ...
```

LOCATION	OrderManager.sol -> 417-419

```
1    function setOrderHook(address _hook) external onlyOwner {
2        orderHook = IOrderHook(_hook);
3        emit OrderHookSet(_hook);
4    }
```

DESCRIPTION	No contract was provided for *positionhook*. Interactions

	with this contract can cause unexpected problems. There is no *whitelistedToken()* inside the version of *pool.sol* we have received. No contract given for *orderHook*. Interactions with this contract can cause unexpected problems.
RECOMMENDATION	Provide implementations for the missing contracts and functions.
RESOLUTION	• There are multiple blank implementations of functions inside *positionhook.sol* • No contract was given for *referralController*. Interactions with this contract can cause unexpected problems. • The *_rebalancePosition()* function in *Pool.sol* is unused **Project team comment:** "The referral controller contract is included in the core repository as it's not related to the trading function. Honestly, we don't have a clear plan for this feature yet so we leave it something like a placeholder rather than an actual implementation. We still looking for ideas from our community. And, we think it's out of the audit scope." The problem is that these functions can be used to freeze existing functionality and may cause other issues such as locking out user funds or the liquidation function. This is a potential security risk because they are directly called by the trading function. Depending on how they are implemented, there is also a risk for re-entrancy issues.

Possible Implementation Errors In _calcTrancheSharesAmount

FINDING ID	#0006
SEVERITY	High Risk
STATUS	Closed
LOCATION	Pool.sol -> 1067-1113

```
1     function _calcTrancheSharesAmount(address _token, uint256 _amount,
  uint256 _updateFlag)
2         internal
3         view
4         returns (uint256[] memory reserves)
5     {
6         // ...
7         for (uint256 k = 0; k < nTranches; k++) {
8             // ...
9
10            for (uint256 i = 0; i < nTranches; i++) {
11                // ...
12                uint256 riskFactor_ = riskFactor[_token][tranche];
13                uint256 shareAmount = MathUtils.frac(_amount,
  riskFactor_, totalRiskFactor_);
14
15                // ...
16            }
17        }
18
19        // ...
20    }
```

DESCRIPTION	Tranches are looped through n^2 times at max which could impact gas. Especially when there are a lot of tranches. The fraction calculation will have rounding errors under certain conditions. This means that with certain tokens some tranches will take on more/less risk because they round values down. In some conditions, going through all the tranches multiple times with rounding errors can cause the revert statement to run.
RECOMMENDATION	Ensure that the calculation does not revert unexpectedly, especially during liquidation.
RESOLUTION	The tranches are now limited by an upper bound of 3 tranches, so there will be no gas limit issues on most chains.

Obelisk Comment: There might be a way to do this with more global looping instead of local looping. For example, one could draw inspiration from the Masterchef contract and probably make it so a global loop has to run sometimes instead of a local one.

Token Functionality In Pool Can Be Broken

FINDING ID	#0007
SEVERITY	High Risk
STATUS	Closed
LOCATION	Pool.sol -> 765-785

```
1     function _accrueInterest(address _token) internal returns
  (uint256) {
2             // ...
3
4             tokenInfo.borrowIndex += (nInterval * interestRate *
  asset.reservedAmount) / asset.poolAmount;
5
6             // ...
7     }
```

DESCRIPTION	If a token *asset.poolAmount* becomes 0 then the calculation's division by 0 will throw an error, even if the unchecked keyword is used. This can occur as a swap has no maximum value and rebalance allows it to become 0 if the *reservedAmount* is 0. This would break the *_accrueInterest()* function, which in turn means that the token would be broken in the pool since so many functions call this function at the start. This would be hard to accomplish as every tranche would have to be drained so it's unlikely to occur for any existing assets. However, if a new token is added someone can backrun that transaction and add amount = 0 to initialize the token *poolAmount* to 0 and break the newly added token.
RECOMMENDATION	Guard invalid values for an asset's *reservedAmount* and *poolAmount*.
RESOLUTION	The project team has added a guard check for *poolAmount*.

No Limit For Protocol Values

FINDING ID	#0008
SEVERITY	High Risk
STATUS	Closed
LOCATION	OrderManager -> 65-73

```
1     function initialize(address _oracle, uint256 _minExecutionFee)
  external initializer {
2         __Ownable_init();
3         __ReentrancyGuard_init();
4         require(_oracle != address(0), "OrderManager:invalidOracle");
5         minExecutionFee = _minExecutionFee;
6         oracle = IOracle(_oracle);
7         nextOrderId = 1;
8         nextSwapOrderId = 1;
9     }
```

LOCATION	OrderManager.sol -> 410-414

```
1     function setMinExecutionFee(uint256 _fee) external onlyOwner {
2         require(_fee > 0, "OrderManager:invalidFeeValue");
3         minExecutionFee = _fee;
4         emit MinExecutionFeeSet(_fee);
5     }
```

LOCATION	LPToken.sol -> 16

```
1     uint256 public immutable redeemCooldown;
```

LOCATION	Pool.sol -> 94-118

```
    function initialize(
        uint256 _maxLeverage,
        uint256 _positionFee,
        uint256 _liquidationFee,
        uint256 _interestRate,
        uint256 _accrualInterval
    )
        external
        initializer
    {
        __Ownable_init();
        __ReentrancyGuard_init();
        if (_accrualInterval == 0) {
            revert PoolErrors.InvalidInterval();
        }
        if (_maxLeverage == 0) {
            revert PoolErrors.InvalidMaxLeverage();
        }
        maxLeverage = _maxLeverage;
        fee.positionFee = _positionFee;
        fee.liquidationFee = _liquidationFee;
        interestRate = _interestRate;
        accrualInterval = _accrualInterval;
        fee.daoFee = FEE_PRECISION;
    }
```

LOCATION	Pool.sol -> 525-532

```
    function setInterestRate(uint256 _interestRate, uint256
_accrualInterval) external onlyOwner {
        if (_accrualInterval == 0) {
            revert PoolErrors.InvalidInterval();
        }
        interestRate = _interestRate;
        accrualInterval = _accrualInterval;
        emit InterestRateSet(_interestRate, _accrualInterval);
    }
```

LOCATION	Pool.sol -> 588-591

```
1     function setMaxPositionSize(uint256 _maxSize) external onlyOwner
  {
2         maxPositionSize = _maxSize;
3         emit MaxPositionSizeSet(_maxSize);
4     }
```

DESCRIPTION	The following values can be set arbitrarily high or low, potentially breaking the functionality of the contracts: • *minExecutionFeè* • *redeemCooldown* • *positionFee* • *liquidationFee* • *interestRate* • *maxPositionSize*
RECOMMENDATION	Add a threshold to the values.
RESOLUTION	Thresholds were added to all values.

No Timelock When Changing Protocol Values

FINDING ID	#0009
SEVERITY	High Risk
STATUS	On-Chain
LOCATION	OrderManager -> 65-73

```
    function setOracle(address _oracle) external onlyOwner {
        require(_oracle != address(0),
  "OrderManager:invalidOracleAddress");
        oracle = IOracle(_oracle);
        emit OracleChanged(_oracle);
    }
```

LOCATION	Pool.sol -> 483-490

```
    function setOracle(address _oracle) external onlyOwner {
        if (_oracle == address(0)) {
            revert PoolErrors.ZeroAddress();
        }
        address oldOracle = address(oracle);
        oracle = IOracle(_oracle);
        emit OracleChanged(oldOracle, _oracle);
    }
```

LOCATION Pool.sol -> 483-490

```
    function setSwapFee(
        uint256 _baseSwapFee,
        uint256 _taxBasisPoint,
        uint256 _stableCoinBaseSwapFee,
        uint256 _stableCoinTaxBasisPoint
    )
        external
        onlyOwner
    {
        _validateMaxValue(_baseSwapFee, MAX_BASE_SWAP_FEE);
        _validateMaxValue(_stableCoinBaseSwapFee, MAX_BASE_SWAP_FEE);
        _validateMaxValue(_taxBasisPoint, MAX_TAX_BASIS_POINT);
        _validateMaxValue(_stableCoinTaxBasisPoint,
MAX_TAX_BASIS_POINT);
        fee.baseSwapFee = _baseSwapFee;
        fee.taxBasisPoint = _taxBasisPoint;
        fee.stableCoinBaseSwapFee = _stableCoinBaseSwapFee;
        fee.stableCoinTaxBasisPoint = _stableCoinTaxBasisPoint;
        emit SwapFeeSet(_baseSwapFee, _taxBasisPoint,
_stableCoinBaseSwapFee, _stableCoinTaxBasisPoint);
    }
```

LOCATION Pool.sol -> 519-523

```
    function setDaoFee(uint256 _daoFee) external onlyOwner {
        _validateMaxValue(_daoFee, FEE_PRECISION);
        fee.daoFee = _daoFee;
        emit DaoFeeSet(_daoFee);
    }
```

LOCATION Pool.sol -> 525-532

```
    function setInterestRate(uint256 _interestRate, uint256
_accrualInterval) external onlyOwner { //@audit no upper bound on
interestRate
        if (_accrualInterval == 0) {
            revert PoolErrors.InvalidInterval();
        }
        interestRate = _interestRate;
        accrualInterval = _accrualInterval;
        emit InterestRateSet(_interestRate, _accrualInterval);
    }
```

LOCATION	Pool.sol -> 534-540

```
    function setOrderManager(address _orderManager) external
onlyOwner {
        if (_orderManager == address(0)) {
            revert PoolErrors.ZeroAddress();
        }
        orderManager = _orderManager;
        emit SetOrderManager(_orderManager);
    }
```

LOCATION	Pool.sol -> 593-596

```
    function setPositionHook(address _hook) external onlyOwner {
        positionHook = IPositionHook(_hook);
        emit PositionHookChanged(_hook);
    }
```

DESCRIPTION	The owner can call *setOracle()*, *setOrderManager()*, *setSwapFee()*, *setDaoFee()*, *setInterestRate()*, *setPositionHook()* to change protocol values at any time.
RECOMMENDATION	Add a time restriction to the noted functions or transfer the contract ownership to a timelock. Obelisk recommends a time delay of at least 72 hours.
RESOLUTION	The project team will transfer the ownership to a timelock. Obelisk will validate it on-chain.

Max Position Bypass

FINDING ID	#0010
SEVERITY	Medium Risk
STATUS	Open
LOCATION	Pool.sol -> 689-710

```
    function _validatePosition(
        Position memory _position,
        address _collateralToken,
        Side _side,
        bool _isIncrease,
        uint256 _indexPrice
    )
        internal
        view
    {
        if ((_isIncrease && _position.size == 0) || (maxPositionSize > 0 && _position.size > maxPositionSize)) {
            revert PoolErrors.InvalidPositionSize();
        }

        uint256 borrowIndex = poolTokens[_collateralToken].borrowIndex;
        if (_position.size < _position.collateralValue || _position.size > _position.collateralValue * maxLeverage) {
            revert PoolErrors.InvalidLeverage(_position.size, _position.collateralValue, maxLeverage);
        }
        if (_liquidatePositionAllowed(_position, _side, _indexPrice, borrowIndex)) {
            revert PoolErrors.UpdateCauseLiquidation();
        }
    }
```

DESCRIPTION	*_position.size* > *maxPositionSize* can be bypassed by running multiple accounts. As there is no global max position for a tranche asset this could cause a manipulated asset to completely drain the vault of every token as the user shorts/longs that asset with multiple small positions.
RECOMMENDATION	A possible solution could be to add a *totalMaxpositionSize* so that every position combined does not exceed it.
RESOLUTION	The project team removed maxPositionSize.

Unbounded Loop

FINDING ID	#0011
SEVERITY	Low Risk
STATUS	Partially Closed
LOCATION	PriceFeed.sol -> 130-135

```
        for (uint256 i = 0; i < reporters.length; i++) {
            if (reporters[i] == reporter) {
                reporters[i] = reporters[reporters.length - 1];
                break;
            }
        }
```

LOCATION	OrderManager.sol -> 400-405

```
    for (uint256 i = 0; i < allPools.length; i++) {
        if (allPools[i] == _pool) {
            allPools[i] = allPools[allPools.length - 1];
            break;
        }
    }
```

LOCATION	Pool.sol -> 870-872

```
    for (uint256 i = 0; i < allTranches.length; i++) {
        sum += _getTrancheValue(allTranches[i]);
    }
```

LOCATION	Pool.sol -> 878-888

```
    for (uint256 i = 0; i < allAssets.length; i++) {
        address token = allAssets[i];
        assert(isAsset[token]); // double check
        AssetInfo memory asset = trancheAssets[_tranche][token];
        uint256 price = _getPrice(token);
        if (isStableCoin[token]) {
            aum = aum.add(price * asset.poolAmount);
        } else {
            aum = aum.add(_calcManagedValue(token, asset, price));
        }
    }
```

LOCATION	Pool.sol -> 921-927

```
    for (uint256 i = 0; i < allTranches.length; i++) {
        address tranche = allTranches[i];
        asset.poolAmount += trancheAssets[tranche]
[_token].poolAmount;
        asset.reservedAmount += trancheAssets[tranche]
[_token].reservedAmount;
        asset.totalShortSize += trancheAssets[tranche]
[_token].totalShortSize;
        asset.guaranteedValue += trancheAssets[tranche]
[_token].guaranteedValue;
    }
```

DESCRIPTION	Iterating over an unbounded array can cause transactions to revert due to the gas limit.
RECOMMENDATION	Provide a limit to the size of the array. Alternatively, pass a lower and upper index as parameters and iterate over a range.
RESOLUTION	The project team added an upper bound to the max number of tranches and assets, and removed the allPools loop from *OrderManager.sol*.

Redundant Code

FINDING ID	#0012
SEVERITY	Informational
STATUS	Closed
LOCATION	Pool.sol -> 1067-1113

```
1    function _calcTrancheSharesAmount(address _token, uint256
  _amount, uint256 _updateFlag)
2        internal
3        view
4        returns (uint256[] memory reserves)
5    {
6        // ...
7
8        for (uint256 k = 0; k < nTranches; k++) {
9            uint256 remaining = _amount; // amount distributed in
  this round
10
11           // ...
12
13           for (uint256 i = 0; i < nTranches; i++) {
14
15               // ...
16
17               if (remaining == 0) {
18                   return reserves;
19               }
20           }
21       }
22
23       if (_amount > 0) {
24           revert PoolErrors.CannotDistributeToTranches(_token,
  _amount, _updateFlag);
25       }
26   }
```

LOCATION	Pool.sol -> 1148-1190

```
    function _calcDecreasePayout(
        Position memory _position,
        address _indexToken,
        address _collateralToken,
        Side _side,
        uint256 _sizeChanged,
        uint256 _collateralChanged
    )
        internal
        view
        returns (DecreasePositionVars memory vars)
    {
        // ...

              SignedInt memory payoutValue = vars.pnl.add(vars.collateralReduced).sub(vars.feeValue);
        if (payoutValue.isNeg()) {
            // deduct uncovered lost from collateral
            remainingCollateral = remainingCollateral.add(payoutValue);
            payoutValue = SignedIntOps.wrap(uint256(0));
        }

        // ...

        vars.payout = payoutValue.isNeg() ? 0 : payoutValue.abs / vars.collateralPrice;

        // ...
    }
```

DESCRIPTION	In *_calcTrancheSharesAmount*: • Unnecessary statement if statement checking *(_amount > 0)*. Note: the code inside the if statement is still necessary. • The variable *remaining* is unnecessary as the *_amount* variable is the same In *_calcDecreasePayout*: • This calculation of *vars.payout* can never use the negative branch as *payoutValue* is set to 0 in the preceding code if it's negative.
RECOMMENDATION	Remove the redundant code.
RESOLUTION	The redundant code was removed.

Minor Issues

FINDING ID	#0013
SEVERITY	Informational
STATUS	Closed
LOCATION	Pool.sol OrderManager.sol

DESCRIPTION	1. In *Pool.sol*: *_transferIn* uses *UniERC20* to differentiate native ETH from wrapped ETH. The resulting logic is excessively complex. 2. In *Pool.sol*: two functions are named *_calcAdjustedFee()*. Both of them do different things and should then have different function names. 3. In *OrderManager.sol*: The receive function doesn't block people from forcing eth into the contract. This can be done by using a self-destruct on a contract with this contract as the receiver. This cannot be prevented.
RECOMMENDATION	1. Wrap all native ETH, and instead, only keep track of wrapped ETH. 2. Change the function names to reflect their different functionality 3. Be aware of this and never rely on the pool ETH balance matching a specific value
RESOLUTION	Project team implemented the recommendation

Token Cannot Be Relisted

FINDING ID	#0014
SEVERITY	Informational
STATUS	Closed
LOCATION	Pool.sol -> 453-462

```
1     function addToken(address _token, bool _isStableCoin) external
  onlyOwner {
2         if (isAsset[_token]) {
3             revert PoolErrors.DuplicateToken(_token);
4         }
5         isAsset[_token] = true;
6         isListed[_token] = true;
7         allAssets.push(_token);
8         isStableCoin[_token] = _isStableCoin;
9         emit TokenWhitelisted(_token);
10    }
```

LOCATION	Pool.sol -> 464-473

```
1     function delistToken(address _token) external onlyOwner {
2         if (!isListed[_token]) {
3             revert PoolErrors.TokenNotListed(_token);
4         }
5         isListed[_token] = false;
6         uint256 weight = targetWeights[_token];
7         totalWeight -= weight;
8         targetWeights[_token] = 0;
9         emit TokenWhitelisted(_token);
10    }
```

DESCRIPTION	If a token is delisted. Then using *addToken()* won't be possible as the token is still an asset. *IsAsset[token]* is never set to false.
RECOMMENDATION	Make sure this is intended as this could cause issues if a token is ever going to be relisted on the platform.
RESOLUTION	Project team implemented the recommendation and also added a check for the max amount of assets possible.

Tranches Are Not Completely Isolated

FINDING ID	#0015
SEVERITY	Informational
STATUS	Open
LOCATION	Pool.sol

DESCRIPTION	The behavior of tranches appears unusual as funds are taken from every unique tranche. If a high risk tranche runs out of money, additional money will be taken from the lower risk ones. Users will always be exposed to high-leverage long/shorters on all tranches which may not be expected.
RECOMMENDATION	Ensure that this is the intended behavior and ensure that adequate documentation is provided.
RESOLUTION	Project team acknowledges this issue. but says that this is intended. Users should be properly informed about the possible impact on their funds.

Denial Of Service(DOS) On Withdrawal

FINDING ID	#0016
SEVERITY	High Risk
STATUS	Closed
LOCATION	LPToken.sol -> 26-35

```
    function mint(address _to, uint256 _amount) external {
        require(msg.sender == minter, "LPToken: !minter");
        lastMinted[_to] = block.timestamp;
        _mint(_to, _amount);
    }

    function burnFrom(address _account, uint256 _amount) public override {
        require(lastMinted[_account] + redeemCooldown <= block.timestamp, "LPToken: redemption delayed");
        _burn(_account, _amount);
    }
```

LOCATION	Pool.sol -> 149-175

```
    function addLiquidity(address _tranche, address _token, uint256 _amountIn, uint256 _minLpAmount, address _to)
        external
        payable
        nonReentrant
        onlyListedToken(_token)
    {
        // ...

        ILPToken(_tranche).mint(_to, lpAmount);
        emit LiquidityAdded(_tranche, msg.sender, _token, _amountIn, lpAmount, feeAmount);
    }
```

LOCATION	Pool.sol -> 177-208

```
1     function removeLiquidity(address _tranche, address _tokenOut,
  uint256 _lpAmount, uint256 _minOut, address _to)
2         external
3         nonReentrant
4         onlyAsset(_tokenOut)
5     {
6         // ...
7
8         lpToken.burnFrom(msg.sender, _lpAmount);
9         _doTransferOut(_tokenOut, _to, outAmountAfterFee);
10        emit LiquidityRemoved(_tranche, msg.sender, _tokenOut,
  _lpAmount, outAmountAfterFee, feeAmount);
11    }
```

DESCRIPTION	If a malicous actor uses *addLiquidity()* and sets the receiver to someone else's address this will reset their *redeemCooldown*. The receiver now has to wait a whole cycle again to be able to withdraw. This blocks the user from withdrawing as long as someone is depositing into their account some small amount.
RECOMMENDATION	Possible solutions include: • Implementing this as fungible: Make an exit queue contract where users can deposit their tokens, then wait a certain time to withdraw their LPs. • Implementing this as non-fungible: Use a non-fungible asset to represent the assets instead (major overhaul) • Removing the delay feature altogether
RESOLUTION	The project team removed the aforementioned feature, thus there is no risk of DOS.

Withdraw Cooldown Can Be Bypassed

FINDING ID	#0017
SEVERITY	Medium Risk
STATUS	Closed
LOCATION	LPToken.sol -> 26-35

```
1     function mint(address _to, uint256 _amount) external {
2         require(msg.sender == minter, "LPToken: !minter");
3         lastMinted[_to] = block.timestamp;
4         _mint(_to, _amount);
5     }
6
7     function burnFrom(address _account, uint256 _amount) public
  override {
8         require(lastMinted[_account] + redeemCooldown <=
  block.timestamp, "LPToken: redemption delayed");
9         _burn(_account, _amount);
10    }
```

DESCRIPTION	If someone transfers their lp tokens to a new account the *require* statement in *burnFrom* can be bypassed. New accounts will have a value of 0 for *lastMinted* by default.
RECOMMENDATION	Possible solutions include: • Implementing this as fungible: Make an exit queue contract where users can deposit their tokens, then wait a certain time to withdraw their LPs. • Implementing this as non-fungible: Use a non-fungible asset to represent the assets instead (major overhaul) • Removing the delay feature altogether
RESOLUTION	The project team removed the aforementioned feature, thus there is no cooldown to be bypassed.

No Max Position Size

FINDING ID	#0018
SEVERITY	Medium Risk
STATUS	Open
LOCATION	Rev1 - Pool.sol -> 689-701

```
    function _validatePosition(
        Position memory _position,
        address _collateralToken,
        Side _side,
        bool _isIncrease,
        uint256 _indexPrice
    )
        internal
        view
    {
        if ((_isIncrease && _position.size == 0) || (maxPositionSize > 0 && _position.size > maxPositionSize)) {
            revert PoolErrors.InvalidPositionSize();
        }
```

LOCATION	Rev2 - Pool.sol -> 715-724

```
    function _validatePosition(
        Position memory _position,
        address _collateralToken,
        Side _side,
        bool _isIncrease,
        uint256 _indexPrice
    ) internal view {
        if ((_isIncrease && _position.size == 0)) {
            revert PoolErrors.InvalidPositionSize();
        }
```

DESCRIPTION	There is a risk that the removal of *maxPositionSize* could lead to manipulation of the pool assets on the decentralized exchanges (DEXes) or centralized exchanges (CEXes) from which the price is fetched. It is difficult to accurately assess the probability of this occurring without knowing the pool amounts and assets that will be used, but if it does happen, the consequences could be adverse. Additionally, there is a risk that an oracle hack or other malicious activity could manipulate

	the prices being reported, which could also have negative consequences.
RECOMMENDATION	Re-add the *maxPositionSize* variable
RESOLUTION	The project team acknowledges the risk and states that they will only allow highly liquid tokens. They plan to work on a longer timeline to address the issue with an alternative solution rather than using a maximum limit **Project Team Comment:** "We acknowledge this risk and select the market carefully with high-liquid tokens. And the price oracle relies on chainlink price feed which is hard to manipulate. We are working on a new version of price oracle with slippage based on token liquidity, but it will not be shipped any time soon."

Rounding Error In _rebalanceTranches 1

FINDING ID	#0019
SEVERITY	Informational
STATUS	Open
LOCATION	Rev-3 - Pool.sol -> 404

```
1    uint256 liquidationFee = fee.liquidationFee / vars.collateralPrice;
```

DESCRIPTION	The liquidation fee has no precision and because solidity doesn't work with decimal numbers it will round down every time collateralPrice is more than the liquidation fee. If there is no fee to be made then it is unclear whether the liquidate function can be called as potential callers get nothing in return for doing it.
RECOMMENDATION	Add precision for the liquidation fee.
RESOLUTION	The project team acknowledged the issue. **Project Team Comment:** “The rounding error is small enough to ignore (never larger than 2 in our fuzz test)”

Rounding error in _rebalanceTranches 2

FINDING ID	#0020
SEVERITY	Informational
STATUS	Open
LOCATION	Rev-5 - Pool.sol -> 1172-1192

```
1     function _rebalanceTranches(address _tokenIn, uint256 _amountIn,
  address _tokenOut, uint256 _amountOut) internal {
2         // amount devided to each tranche
3         uint256[] memory outAmounts;
4
5         if (!isStableCoin[_tokenIn] && isStableCoin[_tokenOut]) {
6             // use token in as index
7             outAmounts = _calcTrancheSharesAmount(_tokenIn, _tokenOut,
  _amountOut, false);
8         } else {
9             // use token out as index
10            outAmounts = _calcTrancheSharesAmount(_tokenOut, _tokenOut,
  _amountOut, false);
11        }
12
13        for (uint256 i = 0; i < allTranches.length;) {
14            address tranche = allTranches[i];
15            trancheAssets[tranche][_tokenOut].poolAmount -=
  outAmounts[i];
16            trancheAssets[tranche][_tokenIn].poolAmount +=
  MathUtils.frac(_amountIn, outAmounts[i], _amountOut);
17            unchecked {
18                ++i;
19            }
20        }
21    }
```

DESCRIPTION	In the *_rebalanceTranches()* function, when looping through all the tranches to redistribute them, a fraction is calculated. However, this calculation may sometimes result in rounding a number down, which can cause small losses in the pool amounts over time.
RECOMMENDATION	Make sure the losses are small enough to ignore.
RESOLUTION	The project team acknowledged the issue. **Project Team Comment:** "The rounding error is small enough to ignore (never larger than 2 in our fuzz test)"

Pool Amount Discrepancy

FINDING ID	#0021
SEVERITY	High risk
STATUS	Closed
LOCATION	Rev-5 - Pool.sol -> 1212-1253

```
function _calcDecreasePayout(
     Position memory _position,
     address _indexToken,
     address _collateralToken,
     Side _side,
     uint256 _sizeChanged,
     uint256 _collateralChanged,
     bool isLiquidate
 ) internal view returns (DecreasePositionVars memory vars) {
     // ...
 }
```

LOCATION	Rev-5 - Pool.sol -> 1063-1104

```
function _releaseTranchesAsset(
     bytes32 _key,
     DecreasePositionVars memory _vars,
     address _indexToken,
     address _collateralToken,
     Side _side
 ) internal {
     // ...
 }
```

DESCRIPTION	If someone opens a short and that short is being liquidated if we follow the liquidation call we can see that inside *_calcDecreasePayout()* In *_calcDecreasePayout* at line 1251: *SignedInt memory poolValueReduced = _side == Side.LONG ? payoutValue.add(vars.feeValue) : vars.pnl;* *vars.poolAmountReduced = poolValueReduced.div(vars.collateralPrice);* *vars.poolAmountReduced* will be a large negative number if the price changes fast.

	When calculating how much each tranche should receive. We take the negative profit and loss (PNL) that is stored in *vars.poolAmountReduced* and subtract it from the pool amount. But because we are dealing with signed integers it is adding the absolute value of the pnl to the pool amount. In *_reserveTrancheAsset* at line 1083. *collateral.poolAmount = SignedIntOps.wrap(collateral.poolAmount).sub(_vars.poolAmountReduced.frac(share, totalShare)).toUint();* This means that in the case of a very negative PNL for a synthetic position, we are adding the synthetic loss to the pool, making it appear as though we have more assets than we actually do.
RECOMMENDATION	Add the collateral amount the user lost and not their negative PNL when shorting.
RESOLUTION	The project team made changes to the code that limit the *poolValueReduced* when shorting, to *_position.collateralValue* minus *totalFee*. It then adds the modified *poolValueReduced* to the *poolAmount*. But before this, the code also divides *poolValueReduced* by the *collateralPrice*. This works as intended when the oracle reports the stablecoin's value as exactly 1 dollar. However, this will result in a discrepancy if the stablecoin's value deviates from 1 dollar as it will either add more or less than expected. The project team made additional changes such that the Stablecoin value is always reported to be 1 dollar.

Average Short Discrepancy

FINDING ID	#0022
SEVERITY	High risk
STATUS	Closed
LOCATION	Rev-5 - Pool.sol -> 1015-1061

```
    function _reserveTrancheAsset(
        bytes32 _key,
        IncreasePositionVars memory _vars,
        address _indexToken,
        address _collateralToken,
        Side _side
    ) internal {
        // ...
    }
```

LOCATION	Rev-5 - Pool.sol -> 935

```
    function _calcManagedValue(address _token, AssetInfo memory _asset,
uint256 _price)
        internal
        view
        returns (SignedInt memory aum)
    {
        uint256 averageShortPrice =
poolTokens[_token].averageShortPrice;
        SignedInt memory shortPnl = _asset.totalShortSize == 0
            ? SignedIntOps.wrap(uint256(0))
            :
SignedIntOps.wrap(averageShortPrice).sub(_price).mul(_asset.totalShortS
ize).div(averageShortPrice);

        aum =
SignedIntOps.wrap(_asset.poolAmount).sub(_asset.reservedAmount).mul(_pr
ice).add(_asset.guaranteedValue);
        aum = aum.sub(shortPnl);
    }
```

DESCRIPTION	When shorting, the global average price is saved over all the tranches. The short is then distributed to the available tranches, and the size of each tranche short is calculated based on its share, as shown in the code below: In *_reserveTrancheAsset* at line 1055:

	indexAsset.totalShortSize += MathUtils.frac(_vars.sizeChanged, share, totalShare); Where *indexAsset* is a storage slot for *trancheAssets[_tranche][token]* This works as long as all tranches receive the same percentage share each time. However, if a tranche becomes full, a new tranche is added or if there is a rounding error the global average may differ from the tranche average. In this case, the managed value for each tranche will be incorrect because it is based on the global average, as shown in the code below. In *_calcManagedValue* at line 943 *SignedIntOps.wrap(averageShortPrice).sub(_price).mul(_asset.totalShortSize).div(averageShortPrice);* *aum = SignedIntOps.wrap(_asset.poolAmount).sub(_asset.reservedAmount).mul(_price).add(_asset.guaranteedValue);* *aum = aum.sub(shortPnl);* This could allow liquidity providers to withdraw more or less than they are entitled to if there are open shorts. This issue is easily simulated when a tranche runs out of funds
RECOMMENDATION	The simplest solution would be to add an average short price per tranche rather than using a global average.
RESOLUTION	The project team added an averageShortPrice per tranche.

Stablecoin Value Hardcoded

FINDING ID	#0023
SEVERITY	Informational
STATUS	Open
LOCATION	Rev-7 - Pool.sol -> 1256-1261

```
1    function _getCollateralPrice(address _token, bool _isIncrease)
  internal view returns (uint256) {
2        return (isStableCoin[_token])
3            // force collateral price = 1 incase of using stablecoin as
  collateral
4            ? 10 ** (USD_VALUE_DECIMAL -
  IDecimalsErc20(_token).decimals())
5            : _getPrice(_token, !_isIncrease);
6    }
```

DESCRIPTION	As part of the resolution of finding #21, the user stablecoin collateral's value changed to a hardcoded value of 1 dollar. This has some risks associated with it. Users should make sure that only proper stablecoins are added and be aware of the relevant risks.
RECOMMENDATION	Create sufficient documentation regarding how the protocol handles stablecoins as collateral and which stablecoins are used.
RESOLUTION	

On-Chain Analysis

Changes To Deployed Contracts

FINDING ID	#0024
SEVERITY	Unknown
STATUS	Open
LOCATION	Rev-7 - Order Manager.sol Rev-7 - Pool.sol Rev-7 - PoolHook.sol Rev-7 - PriceFeed.sol

DESCRIPTION	Unaudited changes were made to the deployed contracts.
RECOMMENDATION	Ensure that changes are well documented and tested.
RESOLUTION	The project team has provided the following explanation for the modifications made to the smart contracts: **Project Team Comment:** “1. When cancelling order, users receive WBNB instead of BNB => so we need to fix by unwrap it 2. We introduced loyalty and referral program so in order for the PoolHook work properly, we need to pass some extra parameters to the hook call 3. Before everytime that users do a swap, we will dynamically calculate the weight of 3 tranches, which results in very high gas cost. Now we will use a virtual asset value which will be calibrate every hour, which reduce the gas cost for swapping substantially.”

Unverified Contract

FINDING ID	#0025
SEVERITY	Unknown
STATUS	Closed
LOCATION	OrderManager 0xf584A17dF21Afd9de84F47842ECEAF6042b1Bb5b Eth Unwrapper 0x3d54A363bC92a2da545f39a0F09dE62a2cC7E62d

DESCRIPTION	New functionality in the order manager links to an unverified contract, labeled as *OrderManager.ethUnwrapper*.
RECOMMENDATION	Verify this contract.
RESOLUTION	The contract was verified.

Timelock Delay Is Short

FINDING ID	#0026
SEVERITY	Medium Risk
STATUS	Partially Mitigated
LOCATION	OrderManager 0xf584A17dF21Afd9de84F47842ECEAF6042b1Bb5b Pool 0xA5aBFB56a78D2BD4689b25B8A77fd49Bb0675874 PoolHook 0x635aaC65f37a6bbE06a2dde77B0fD2F1748674d4 PriceFeed 0xe423BB0a8b925EABF625A8f36B468ab009a854e7 LevelOracle 0x04Db83667F5d59FF61fA6BbBD894824B233b3693 Timelock 0x360071D15cce5542E6B7209752eA479b84b28625

DESCRIPTION	The timelock contract has a 12-hour delay.
RECOMMENDATION	Obelisk recommends a delay of at least 72 hours.
RESOLUTION	**Project Team Comment:** “After launch, a lot of parameters need to be updated to optimize, so the timelock delay can't be long. We will schedule to increase timelock delay to 24hr, 48hr and 72hr later”

External Addresses

Externally Owned Accounts

Not Applicable

External Contracts

These contracts are not part of the audit scope.

Eth Unwrapper

ADDRESS	0x3d54A363bC92a2da545f39a0F09dE62a2cC7E62d
USAGE	0xf584A17dF21Afd9de84F47842ECEAF6042b1Bb5b *OrderManager.ethUnwrapper* - Variable
IMPACT	• receives allowance of tokens deposited by users

Executor

ADDRESS	0xe423BB0a8b925EABF625A8f36B468ab009a854e7
USAGE	0xf584A17dF21Afd9de84F47842ECEAF6042b1Bb5b *OrderManager.executor* - Variable
IMPACT	• has elevated permissions as owner, operator, or other

Extra Pool Hook

ADDRESS	0x8826C45DAB800697C48ea87322Ac0a6Ed70cE4A4
USAGE	0xA5aBFB56a78D2BD4689b25B8A77fd49Bb0675874 *Pool.poolHook* - Variable
IMPACT	• impacts ability to deposit or withdraw tokens • impacts other user actions

Fee Distributor

ADDRESS	0x8BFf27E9Fa1C28934554e6B5239Fb52776573619
USAGE	0xA5aBFB56a78D2BD4689b25B8A77fd49Bb0675874 *Pool.feeDistributor* - Variable
IMPACT	• has elevated permissions as owner, operator, or other

Proxy Admin

ADDRESS	0x8f886b4b10344289cEAd777953f95FA0317bcD33
USAGE	0xf584A17dF21Afd9de84F47842ECEAF6042b1Bb5b 0xA5aBFB56a78D2BD4689b25B8A77fd49Bb0675874 *eip1967.proxy.admin* - Variable
IMPACT	• has elevated permissions as owner, operator, or other

Timelock

ADDRESS	0x360071D15cce5542E6B7209752eA479b84b28625
USAGE	0xf584A17dF21Afd9de84F47842ECEAF6042b1Bb5b *OrderManager.owner* - Variable 0xA5aBFB56a78D2BD4689b25B8A77fd49Bb0675874 *Pool.owner* - Variable 0x635aaC65f37a6bbE06a2dde77B0fD2F1748674d4 *PoolHook.owner* - Variable 0xe423BB0a8b925EABF625A8f36B468ab009a854e7 *PriceFeed.owner* - Variable 0x04Db83667F5d59FF61fA6BbBD894824B233b3693 *LevelOracle.owner* - Variable
IMPACT	• has elevated permissions as owner, operator, or other

Timelock Admin

ADDRESS	0x6023C6afa26a68E05672F111FdbB1De93cBAc621
USAGE	0x360071D15cce5542E6B7209752eA479b84b28625 *Timelock.admin* - Variable
IMPACT	• has elevated permissions as owner, operator, or other

External Tokens

These contracts are not part of the audit scope.

LyLevel

ADDRESS	0x95883611685a20936EC935B0A33F82e11D478e3D
USAGE	0x635aaC65f37a6bbE06a2dde77B0fD2F1748674d4 *PoolHook.lyLevel* - Variable
IMPACT	• ERC20 Token

Wrapped BNB

ADDRESS	0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
USAGE	0xf584A17dF21Afd9de84F47842ECEAF6042b1Bb5b *OrderManager.weth* - Variable
IMPACT	• ERC20 Token

Appendix A - Reviewed Documents

Deployed Contracts

Document	Address
hooks/PoolHook.sol	0x635aaC65f37a6bbE06a2dde77B0fD2F1748674d4
oracle/LevelOracle.sol	0x04Db83667F5d59FF61fA6BbBD894824B233b3693
oracle/PriceFeed.sol	0xe423BB0a8b925EABF625A8f36B468ab009a854e7
orders/OrderManager.sol	Proxy 0xf584A17dF21Afd9de84F47842ECEAF6042b1Bb5b Implementation 0xEC602a92d744F502ad8ceb310928759e5f4FC363
pool/Pool.sol	Proxy 0xA5aBFB56a78D2BD4689b25B8A77fd49Bb0675874 Implementation 0x0564b4b30A97ae84b39a677a38E1a7CF4c0f8142
tokens/LPToken.sol	Senior LLP 0xB5C42F84Ab3f786bCA9761240546AA9cEC1f8821 Mezzanine LLP 0x4265af66537F7BE1Ca60Ca6070D97531EC571BDd Junior LLP 0xcC5368f152453D497061CB1fB578D2d3C54bD0A0

Libraries And Interfaces

interfaces/AggregatorV3Interface.sol interfaces/IETHUnwrapper.sol interfaces/ILevelOracle.sol/ interfaces/ILPToken.sol interfaces/IMintableErc20.sol interfaces/IOracle.sol interfaces/IOrderHook.sol interfaces/IOrderManager.sol interfaces/IPool.sol interfaces/IPoolHook.sol interfaces/IReferralController.sol interfaces/IWETH.sol

lib/MathUtils.sol lib/PositionUtils.sol lib/SignedInt.sol lib/UniERC20.sol pool/PoolErrors.sol pool/PoolStorage.sol

Revisions

Revision 1	f9df890a828d055fc86986f32f79426d5eec3178
Revision 2	8f560419a8b91c30040e97a0d2d3a6c44cc8863c
Revision 3	26e0b305ec53a3d5ecef935ec018e5f5cc71228b
Revision 4	f4a667eeb75db92a78032ec9484e75729c64586d
Revision 5	091c042987b67a487b51e20bb2e462553c92e7ef
Revision 6	c1063830e22f6a7a35db7fe07a33ca4fd2edd8bb
Revision 7	87f90307d1b1e38c66a2258e28012c01c6be07a4

Imported Contracts

OpenZeppelin	4.8.0
OpenZeppelin Upgradeable	4.8.0

Appendix B - Risk Ratings

Risk	Description
High Risk	Security risks that are ***almost certain*** to lead to ***impairment or loss of funds***. Projects are advised to fix as soon as possible.
Medium Risk	Security risks that are ***very likely*** to lead to ***impairment or loss of funds*** with ***limited impact***. Projects are advised to fix as soon as possible.
Low Risk	Security risks that can lead to ***damage to the protocol***. Projects are advised to fix. Issues with this rating might be

	used in an exploit with other issues to cause significant damage.
Informational	Noteworthy information. Issues may include code conventions, missing or conflicting information, gas optimizations, and other advisories.

Appendix C - Finding Statuses

Closed	Contracts were modified to permanently resolve the finding.
Mitigated	The finding was resolved on-chain. The issue may require monitoring, for example in the case of a time lock.
Partially Closed	Contracts were modified to partially fix the issue
Partially Mitigated	The finding was resolved by project specific methods which cannot be verified on chain. Examples include compounding at a given frequency, or the use of a multisig wallet.
Open	The finding was not addressed.

Appendix D - Glossary

Contract Structure

Contract: An address with which provides functionality to users and other contracts. They are implemented in code and deployed to the blockchain.
Protocol: A system of contracts which work together.
Stakeholders: The users, operators, owners, and other participants of a contract.

Security Concepts

Bug**:** A defect in the contract code.
Exploit**:** A chain of events involving bugs, vulnerabilities, or other security risks which damages a protocol.
Funds: Tokens deposited by users or other stakeholders into a protocol.
Impairment: The loss of functionality in a contract or protocol.
Security risk: A circumstance that may result in harm to the stakeholders of a protocol. Examples include vulnerabilities in the code, bugs, excessive permissions, missing timelock, etc.
Vulnerability: A vulnerability is a flaw that allows an attacker to potentially cause harm to the stakeholders of a contract. They may occur in a contract's code, design, or deployed state on the blockchain.

Appendix E - Audit Procedure

A typical Obelisk audit uses a combination of the three following methods:

Manual analysis consists of a direct inspection of the contracts to identify any security issues. Obelisk auditors use their experience in software development to spot vulnerabilities. Their familiarity with common contracts allows them to identify a wide range of issues in both forked contracts as well as original code.

Static analysis is software analysis of the contracts. Such analysis is called “static” as it examines the code outside of a runtime environment. Static analysis is a powerful tool used by auditors to identify subtle issues and to verify the results of manual analysis.

On-chain analysis is the audit of the contracts as they are deployed on the block-chain. This procedure verifies that:

- deployed contracts match those which were audited in manual/static analysis;
- contract values are set to reasonable values;
- contracts are connected so that interdependent contract function correctly;
- and the ability to modify contract values is restricted via a timelock or DAO mechanism. (We recommend a timelock value of at least 72 hours)

Each obelisk audit is performed by at least two independent auditors who perform their analysis separately.

After the analysis is complete, the auditors will make recommendations for each issue based on best practice and industry standards. The project team can then resolve the issues, and the auditors will verify that the issues have been resolved with no new issues introduced.

Our auditing method lays a particular focus on the following important concepts:

- Quality code and the use of best practices, industry standards, and thoroughly tested libraries.
- Testing the contract from different angles to ensure that it works under a multitude of circumstances.
- Referencing the contracts through databases of common security flaws.

Follow Obelisk Auditing for the Latest Information

 ObeliskOrg

 ObeliskOrg

OBELISK

Part of Tibereum Group