



# Orange Finance Audit Report

Apr 25, 2023



# Table of Contents

Summary	2
Overview	3
Issues	4
[WP-H1] Attacker can manipulate the price (to a lower value), mint and reverse the price and burn to steal funds from other share holders	4
[WP-H2] <code>rebalance()</code> should add collateral when needed	8
[WP-H3] <code>_swapAndAddLiquidity()</code> can cause loss to the other shareholders when the market is volatile	10
[WP-H4] A malicious early user/attacker can manipulate the <code>pricePerShare</code> to take an unfair share of future users' deposits	11
[WP-M5] Lack of slippage control for <code>rebalance()</code>	13
[WP-I6] <code>OrangeAlphaVault.constructor()</code> depends on the order of <code>(token0, token1)</code> being <code>(weth, usdc)</code> , which is not the case on Ethereum and Polygon	14
[WP-G7] Calling <code>OracleLibrary.getQuoteAtTick()</code> only when needed	16
Appendix	18
Disclaimer	19

## Summary

This report has been prepared for Orange Finance Audit Report smart contract, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.



# Overview

## Project Summary

Project Name	Orange Finance Audit Report
Codebase	<a href="https://github.com/orange-finance/alpha-contract">https://github.com/orange-finance/alpha-contract</a>
Commit	37659c1a5b2d091813b236a4a9dd4e75fb99bb97
Language	Solidity

## Audit Summary

Delivery Date	Apr 25, 2023
Audit Methodology	Static Analysis, Manual Review
Total Issues	7

## [WP-H1] Attacker can manipulate the price (to a lower value), mint and reverse the price and burn to steal funds from other share holders

High

### Issue Description

<https://github.com/orange-finance/alpha-contract/blob/d70731409d520555dab276594bd45eff7c43f82f/contracts/core/OrangeAlphaVault.sol#L745-L821>

```
745 function deposit(  
746     uint256 _assets,  
747     address _receiver,  
748     uint256 _minShares  
749 ) external returns (uint256 shares_) {  
750     //validation  
751     if (_receiver != msg.sender) {  
752         revert(Errors.DEPOSIT_RECEIVER);  
753     }  
754     if (_assets == 0) revert(Errors.DEPOSIT_ZERO);  
755     if (deposits[_receiver].assets + _assets > depositCap(_receiver)) {  
756         revert(Errors.DEPOSIT_CAP_OVER);  
757     }  
758     deposits[_receiver].assets += _assets;  
759     if (totalDeposits + _assets > totalDepositCap) {  
760         revert(Errors.TOTAL_DEPOSIT_CAP_OVER);  
761     }  
762     totalDeposits += _assets;  
763  
764     Ticks memory _ticks = _getTicksByStorage();  
765  
766     //mint  
767     shares_ = _convertToShares(_assets, _ticks);  
768     if (_minShares > shares_) {  
769         revert(Errors.LESS_THAN_MIN_SHARES);  
770     }  
771     _mint(_receiver, shares_);
```

```

    @@ 772,820 @@
821 }

```

<https://github.com/orange-finance/alpha-contract/blob/96be6d17b9185aa8c499332364aff8f6b9f07dee/contracts/core/OrangeAlphaVault.sol#L229-L250>

```

229 function _totalAssets(Ticks memory _ticks) internal view returns (uint256) {
230     UnderlyingAssets memory _underlyingAssets = _getUnderlyingBalances(
231         _ticks
232     );
233
234     // Aave positions
235     uint256 amount0Debt = debtToken0.balanceOf(address(this));
236     uint256 amount1Supply = aToken1.balanceOf(address(this));
237
238     return
239         _alignTotalAsset(
240             _ticks,
241             _underlyingAssets.amount0Current +
242             _underlyingAssets.accruedFees0 +
243             _underlyingAssets.amount0Balance,
244             _underlyingAssets.amount1Current +
245             _underlyingAssets.accruedFees1 +
246             _underlyingAssets.amount1Balance,
247             amount0Debt,
248             amount1Supply
249         );
250 }

```

<https://github.com/orange-finance/alpha-contract/blob/d70731409d520555dab276594bd45eff7c43f82f/contracts/core/OrangeAlphaVault.sol#L357-L392>

```

357 function _getUnderlyingBalances(Ticks memory _ticks)
358     internal
359     view
360     returns (UnderlyingAssets memory underlyingAssets)
361 {
362     (
363         uint128 liquidity,

```

```

364         uint256 feeGrowthInside0Last,
365         uint256 feeGrowthInside1Last,
366         uint128 tokensOwed0,
367         uint128 tokensOwed1
368     ) = pool.positions(_getPositionID(_ticks.lowerTick, _ticks.upperTick));
369
370     // compute current holdings from liquidity
371     if (liquidity > 0) {
372         (
373             underlyingAssets.amount0Current,
374             underlyingAssets.amount1Current
375         ) = LiquidityAmounts.getAmountsForLiquidity(
376             _ticks.sqrtRatioX96,
377             _ticks.lowerTick.getSqrtRatioAtTick(),
378             _ticks.upperTick.getSqrtRatioAtTick(),
379             liquidity
380         );
381     }
382
383     underlyingAssets.accruedFees0 =
384         _computeFeesEarned(true, feeGrowthInside0Last, liquidity, _ticks) +
385         uint256(tokensOwed0);
386     underlyingAssets.accruedFees1 =
387         _computeFeesEarned(false, feeGrowthInside1Last, liquidity, _ticks) +
388         uint256(tokensOwed1);
389
390     underlyingAssets.amount0Balance = token0.balanceOf(address(this));
391     underlyingAssets.amount1Balance = token1.balanceOf(address(this));
392 }

```

`_totalAssets()` will be used to calculate the price per share in `mint()` .

However, as the `ticks.sqrtRatioX96` can be manipulated, there is a chance for the attacker to manipulate the price of ETH to a lower value, mint and reverse the price and burn to steal funds from other share holders.

## PoC

Notice: This is NOT a traditional MEV sandwich attack vector, therefore it CAN NOT be prevented by introducing a proper slippage control parameter.

Given:

- market price of ETH: 2000 USDC
- total vault holdings: 1000 ETH + 1 USDC, worth \$2M
- totalShares: 100

The attacker can:

1. Manipulate the price of the pool by swapping a huge amount of ETH to USDC, crashing the price from **2000** to **1000** .
2. `deposit()` with 10k USDC. As the price is now manipulated, `_totalAssets()` is worth only \$1M, therefore, the attacker will get **2** shares.
3. Restore the price by swapping USDC back to ETH;
4. `redeem()` the **2** shares received from step 2, as the price has been restored, the `_totalAssets()` is about \$2M now, therefore the attacker will get about 2000 USDC.

## Recommendation

Consider changing the `deposit()` function to always add liquidity and hedge position proportionally to the current holdings. And pull the amount of USDC needed from the user's wallet directly.

## Status

✓ Fixed



## [WP-H2] `rebalance()` should add collateral when needed

High

### Issue Description

<https://github.com/orange-finance/alpha-contract/blob/d70731409d520555dab276594bd45eff7c43f82f/contracts/core/OrangeAlphaVault.sol#L947-L1036>

```

947  function rebalance(int24 _newLowerTick, int24 _newUpperTick)
948      external
949      onlyOwner
950  {
951      @@ 951,978 @@
952
953      //calculate repay or borrow amount
954      (, uint256 _newBorrow) = _computeSupplyAndBorrow(
955          _totalAssets(_ticks),
956          _ticks
957      );
958
959      // 4. Swap
960      // 5. Repay or borrow (if swapping from ETH to USDC, do borrow)
961      @@ 987,1007 @@
962
963      } else {
964          //borrow
965          aave.borrow(
966              address(token0),
967              _newBorrow - _debtBalance,
968              2,
969              0,
970              address(this)
971          );
972      }
973
974      // 6. Add Liquidity
975      uint256 reinvest0 = token0.balanceOf(address(this));
976      uint256 reinvest1 = token1.balanceOf(address(this));
977      _swapAndAddLiquidity(reinvest0, reinvest1, _ticks);
978
979  }

```

```
1024      (uint128 newLiquidity, , , ) = pool.positions(  
1025          _getPositionID(_ticks.lowerTick, _ticks.upperTick)  
1026      );  
1027      if (newLiquidity == 0) {  
1028          revert(Errors.NEW_LIQUIDITY_ZERO);  
1029      }  
1030  
1031      emit Rebalance(_newLowerTick, _newUpperTick, liquidity, newLiquidity);  
1032      _emitAction(3, _ticks);  
1033  
1034      //reset stoplosses  
1035      stoplosses = false;  
1036  }
```

When `_newBorrow` is greater than `_debtBalance` , `rebalance()` will borrow more to increase the hedge position.

It should add collateral at the same time, if needed.

Otherwise, the health factor on Aave can be low, putting the whole hedge position at risk of liquidation.

## Status

✓ Fixed

## [WP-H3] `_swapAndAddLiquidity()` can cause loss to the other shareholders when the market is volatile

High

### Issue Description

<https://github.com/orange-finance/alpha-contract/blob/d70731409d520555dab276594bd45eff7c43f82f/contracts/core/OrangeAlphaVault.sol#L796-L803>

```
796 // 4. Swap from USDC to ETH (if necessary)
797 // 5. Add Liquidity
798 uint256 _addingUsdc = _assets - _supply;
799 (
800     _liquidity,
801     _amountDeposited0,
802     _amountDeposited1
803 ) = _swapAndAddLiquidity(_borrow, _addingUsdc, _ticks);
```

`_swapAndAddLiquidity()` in `deposit()` is using the vault's public money to swap, so the slippage is at the cost of all shareholders.

Attacker can exploit this with a sandwich attack on `deposit()`.

### Recommendation

See the Recommendation on [WP-H1].

### Status

✓ Fixed

## [WP-H4] A malicious early user/attacker can manipulate the pricePerShare to take an unfair share of future users' deposits

High

### Issue Description

<https://github.com/orange-finance/alpha-contract/blob/96be6d17b9185aa8c499332364aff8f6b9f07dee/contracts/core/OrangeAlphaVault.sol#L689-L757>

```

689  function deposit(uint256 _assets, address _receiver)
690      external
691      returns (uint256 shares_)
692  {
693      //validation
694      if (_receiver != msg.sender) {
695          revert InvalidDepositReceiver();
696      }
697      if (_assets == 0) revert InvalidDepositZero();
698      if (deposits[_receiver].assets + _assets > depositCap(_receiver)) {
699          revert InvalidDepositCapOver();
700      }
701      deposits[_receiver].assets += _assets;
702      if (totalDeposits + _assets > totalDepositCap) {
703          revert InvalidTotalDepositCapOver();
704      }
705      totalDeposits += _assets;
706
707      Ticks memory _ticks = _getTicksByStorage();
708
709      //mint
710      shares_ = _convertToShares(_assets, _ticks);
711      _mint(_receiver, shares_);
712
713      @@ 712,756 @@
757  }
```

<https://github.com/orange-finance/alpha-contract/blob/96be6d17b9185aa8c499332364aff8f6b9f07dee/contracts/core/OrangeAlphaVault.sol#L253-L263>

```

253 function _convertToShares(uint256 _assets, Ticks memory _ticks)
254     public
255     view
256     returns (uint256 shares)
257 {
258     uint256 supply = totalSupply(); // Saves an extra SLOAD if totalSupply is
    non-zero.
259     return
260         supply == 0
261         ? _assets
262         : _assets.mulDiv(supply, _totalAssets(_ticks));
263 }

```

The first minter can `deposit()` , and then withdraw all but a small amount (eg, `199 wei`) of the deposit to inflate the pps of the vault.

- `deposit()` `1e18 wei` and get `1e18 wei` of shares;
- `redeem()` `1e18 - 199 wei` of shares

Then the attacker can send `100e18 - 199` of tokens and inflate the price per share to `100e18 * 1e18 / 199` .

As a result, the future user who deposits `1e18` will only receive `1e18 * 199 / 100e18 = 1 wei` of shares.

They will immediately lose `0.495e18` or half of their deposits if they `redeem()` right after the `deposit()` .

## Recommendation

Consider requiring a minimal amount of shares to be minted for the first minter, and send a portion of the initial mints as a reserve to the DAO so that the pricePerShare can be more resistant to manipulation.

## Status

✓ Fixed

## [WP-M5] Lack of slippage control for `rebalance()`

Medium

### Issue Description

<https://github.com/orange-finance/alpha-contract/blob/96be6d17b9185aa8c499332364aff8f6b9f07dee/contracts/core/OrangeAlphaVault.sol#L880-L883>

```
880     function rebalance(int24 _newLowerTick, int24 _newUpperTick)
881         external
882         onlyOwner
883     {
```

There is no slippage control in the `rebalance()` function.

This means that a sudden market movement or an intentional frontrun price manipulation may result in a different output for the caller (the manager).

Specifically, a different `newLiquidity` as the result of the `rebalance()` .

### Status

✓ Fixed

## [WP-I6] `OrangeAlphaVault.constructor()` depends on the order of `(token0, token1)` being `(weth, usdc)` , which is not the case on Ethereum and Polygon

### Informational

### Issue Description

[https://github.com/orange-finance/alpha-contract/blob/](https://github.com/orange-finance/alpha-contract/blob/96be6d17b9185aa8c499332364aff8f6b9f07dee/contracts/core/OrangeAlphaVault.sol#L48-L52)

[96be6d17b9185aa8c499332364aff8f6b9f07dee/contracts/core/OrangeAlphaVault.sol#L48-L52](https://github.com/orange-finance/alpha-contract/blob/96be6d17b9185aa8c499332364aff8f6b9f07dee/contracts/core/OrangeAlphaVault.sol#L48-L52)

```

48     IERC20 public token0; //weth
49     IERC20 public token1; //usdc
50     IAaveV3Pool public aave;
51     IERC20 public debtToken0; //weth
52     IERC20 public aToken1; //usdc

```

[https://github.com/orange-finance/alpha-contract/blob/](https://github.com/orange-finance/alpha-contract/blob/96be6d17b9185aa8c499332364aff8f6b9f07dee/contracts/core/OrangeAlphaVault.sol#L64-L104)

[96be6d17b9185aa8c499332364aff8f6b9f07dee/contracts/core/OrangeAlphaVault.sol#L64-L104](https://github.com/orange-finance/alpha-contract/blob/96be6d17b9185aa8c499332364aff8f6b9f07dee/contracts/core/OrangeAlphaVault.sol#L64-L104)

```

64     constructor(
65         string memory _name,
66         string memory _symbol,
67         address _pool,
68         address _aave,
69         int24 _lowerTick,
70         int24 _upperTick
71     ) ERC20(_name, _symbol) {
72         // setting addresses and approving
73         pool = IUniswapV3Pool(_pool);
74         token0 = IERC20(pool.token0());
75         token1 = IERC20(pool.token1());
76         token0.safeApprove(_pool, type(uint256).max);
77         token1.safeApprove(_pool, type(uint256).max);
78
79         aave = IAaveV3Pool(_aave);
80         token0.safeApprove(_aave, type(uint256).max);
81         token1.safeApprove(_aave, type(uint256).max);

```

```

82     DataTypes.ReserveData memory reserveDataToken0 = aave.getReserveData(
83         address(token0)
84     );
85     debtToken0 = IERC20(reserveDataToken0.variableDebtTokenAddress);
86     DataTypes.ReserveData memory reserveDataToken1 = aave.getReserveData(
87         address(token1)
88     );
89     aToken1 = IERC20(reserveDataToken1.aTokenAddress);
90     //this decimal is same as token1's decimal
91     _decimal = IERC20Decimals(address(token1)).decimals();
92
93     // these variables can be updated by the manager
94     _depositCap = 1_000_000 * 1e6;
95     totalDepositCap = 1_000_000 * 1e6;
96     slippageBPS = 500; // default: 5% slippage
97     slippageInterval = 5 minutes;
98     maxLtv = 8000; //80%
99
100    //setting ticks
101    _validateTicks(_lowerTick, _upperTick);
102    lowerTick = _lowerTick;
103    upperTick = _upperTick;
104 }

```

- USDC / ETH 0.05% on Ethereum: <https://etherscan.io/address/0x88e6a0c2ddd26feeb64f039a2c41296fcb3f5640#readContract>  
– token0  $\boxtimes$  USDC
- USDC / WETH 0.05% on Polygon: <https://polygonscan.com/address/0x45dda9cb7c25131df268515131f647d726f50608#readContract>  
– token0  $\boxtimes$  USDC

## Status

 Acknowledged



## [WP-G7] Calling `OracleLibrary.getQuoteAtTick()` only when needed

Gas

### Issue Description

<https://github.com/orange-finance/alpha-contract/blob/d70731409d520555dab276594bd45eff7c43f82f/contracts/core/OrangeAlphaVault.sol#L319-L348>

```

319     function _alignTotalAsset(
320         Ticks memory _ticks,
321         uint256 amount0Current,
322         uint256 amount1Current,
323         uint256 amount0Debt,
324         uint256 amount1Supply
325     ) internal view returns (uint256 totalAlignedAssets) {
326         if (amount0Current < amount0Debt) {
327             uint256 amount0deducted = amount0Debt - amount0Current;
328             amount0deducted = OracleLibrary.getQuoteAtTick(
329                 _ticks.currentTick,
330                 uint128(amount0deducted),
331                 address(token0),
332                 address(token1)
333             );
334             totalAlignedAssets =
335                 amount1Current +
336                 amount1Supply -
337                 amount0deducted;
338         } else {
339             uint256 amount0Added = amount0Current - amount0Debt;
340             amount0Added = OracleLibrary.getQuoteAtTick(
341                 _ticks.currentTick,
342                 uint128(amount0Added),
343                 address(token0),
344                 address(token1)
345             );
346             totalAlignedAssets = amount1Current + amount1Supply + amount0Added;
347         }
348     }

```

## Recommendation

```

319     function _alignTotalAsset(
320         Ticks memory _ticks,
321         uint256 amount0Current,
322         uint256 amount1Current,
323         uint256 amount0Debt,
324         uint256 amount1Supply
325     ) internal view returns (uint256 totalAlignedAssets) {
326         if (amount0Current < amount0Debt) {
327             uint256 amount0deducted = amount0Debt - amount0Current;
328             amount0deducted = OracleLibrary.getQuoteAtTick(
329                 _ticks.currentTick,
330                 uint128(amount0deducted),
331                 address(token0),
332                 address(token1)
333             );
334             totalAlignedAssets =
335                 amount1Current +
336                 amount1Supply -
337                 amount0deducted;
338         } else {
339             uint256 amount0Added = amount0Current - amount0Debt;
340             if (amount0Added > 0)
341                 amount0Added = OracleLibrary.getQuoteAtTick(
342                     _ticks.currentTick,
343                     uint128(amount0Added),
344                     address(token0),
345                     address(token1)
346                 );
347             totalAlignedAssets = amount1Current + amount1Supply + amount0Added;
348         }
349     }

```

## Status

✓ Fixed



# Appendix

## 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 WatchPug; however, WatchPug 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.

## Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Smart Contract technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim 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. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, 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, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. 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. 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.