Report

v. 3.0

Customer

ReservoirFi

Smart Contract Audit

AMM Core update

31st January 2025

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	14.01.25	A. Zveryanskaya	Initial Draft
0.2	14.01.25	A. Zveryanskaya	Minor revision
1.0	14.01.25	A. Zveryanskaya	Release
1.1	28.01.25	A. Zveryanskaya	The ″Code with Fixes″ link has been updated
2.0	28.01.25	A. Zveryanskaya	Release
2.1	31.01.25	A. Zveryanskaya	The ″Code with Fixes″ link has been updated
3.0	31.01.25	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

Reservoir Finance is an EVM compatible protocol providing capital efficiency to DeFi through transparency, responsible credit usage, and superfluid collateral.

3 Project scope

We've been asked to review the Automated Market Maker (AMM) update:

- Diff Code
- Code with Fixes

Files:

/		
	Constants.sol	ReservoirPair.sol
asset-management/		
	EulerV2Manager.sol	
curve/constant-product/		
	ConstantProductPair.sol	
curve/stable/		
	StablePair.sol	
libraries/		
	ConstantProduct OracleMath.sol	
libraries/		
	StableMath.sol	StableOracleMath.sol
structs/		
	Observation.sol	

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 1 critical, 4 major, and a few less important issues. All identified Critical issues have been fixed.

Fixed 3 out of 5 issues

6 Critical Issues

CVF-1. FIXED

- **Category** Flaw
- **Source** EulerV2Manager.sol

Description This call doesn't make sense and could potentially open an attack vector.

Recommendation Remove it.

```
205 +SafeTransferLib.safeApprove(address(aToken), address(aPair),
      ↪ aAmount);
```

7 Major Issues

CVF-2. FIXED

- **Category** Overflow/Underflow
- **Source** StablePair.sol

Description Overflow is possible here.

Recommendation Use safe conversion.

Client Comment *Related to #8 and #27, removed conversion and use SafeCast for casting into uint192*

```
105 +lReserve0 += uint104(lFee0);
106 +lReserve1 += uint104(lFee1);
```

```
129 +lastInvariant = uint192(lNewLiq);
```

CVF-3. INFO

- **Category** Procedural
- **Source** StablePair.sol

Description Updating storage after external calls is a dangerous practice.

Recommendation Perform token transfers after all the storage updates.

Client Comment *Acknowledged.*
Will not fix because: 1. UniV2 also has this pattern of updating storage after external call 2. This case is only highlighted for burn, but a similar pattern also exists for swap which can't be re-arranged as it will break the flash swap logic. 3. Will increase the bytecode size beyond the 24576 bytes limit

```
162 +_checkedTransfer(token0, aTo, rAmount0, lReserve0, lReserve1);
163 +_checkedTransfer(token1, aTo, rAmount1, lReserve0, lReserve1);
```

```
171 +_update(lBalance0, lBalance1, lReserve0, lReserve1,
      ↪ lBlockTimestampLast, lIndex);
```

CVF-4. FIXED

- **Category** Unclear behavior
- **Source** EulerV2Manager.sol

Description There is no check to ensure that the asset actually has a vault.

Recommendation Add such a check.

```
293 +IERC4626 lVault = assetVault[aAsset];
294 +SafeTransferLib.safeApprove(address(aAsset), address(lVault),
       ↪ aAmount);
295 +uint256 lNewShares = lVault.deposit(aAmount, address(this));
```

CVF-5. INFO

- **Category** Unclear behavior
- **Source** ReservoirPair.sol

Description This won't work for tokens with more than 18 decimals.

Client Comment *This is the intended scenario, to only support tokens with decimals <= 18. Added documentation for clarification*

```
72 +token0PrecisionMultiplier = uint128(10) ** (18 - aToken0.decimals()
       ↪ ) ;
73 +token1PrecisionMultiplier = uint128(10) ** (18 - aToken1.decimals()
       ↪ ) ;
```

8 Moderate Issues

CVF-6. FIXED

- **Category** Suboptimal
- **Source** StablePair.sol

Description This sounds like a good reason to improve the "_computeLiquidity" function, rather than a reason to not improve this function.

Client Comment *Fixed by removing comment.*
Actually we have improved the _computeLiquidityFromAdjustedBalances function by introducing fullMulDiv at appropriate places. Therefore this comment is no longer valid.

```
119 +// in which case the mint will fail anyway because it would have
        ↪ reverted at _computeLiquidity
```

CVF-7. FIXED

- **Category** Procedural
- **Source** StablePair.sol

Description Relying on business-level constraints when deciding not to perform low-level overflow checks is a good practice, as business logic could change, and also could contain errors.

Recommendation Do not rely on business-level constraint in low-level code.

Client Comment *Fixed with safe conversion.*

```
125 +// Casting is safe as the max invariant would be 2 * uint104 *
        ↪ uint60 (in the case of tokens
126 +// with 0 decimal places).
127 +// Which results in 112 + 60 + 1 = 173 bits.
128 +// Which fits into uint192.
```

CVF-8. INFO

- **Category** Procedural
- **Source** StablePair.sol

Description Performing several token transfers in one transaction is a bad practice, as it once transfer fails (for example the token is frozen or the recipient is in the black list), the other transfer will also be reverted.

Recommendation Store token amounts, users are eligible to withdraw, in a mapping, and allow withdrawing via separate calls.

Client Comment *Acknowledged, but will not implement as it necessitates large changes to the existing infrastructure.*

```
162 +_checkedTransfer(token0, aTo, rAmount0, lReserve0, lReserve1);
163 +_checkedTransfer(token1, aTo, rAmount1, lReserve0, lReserve1);
```

CVF-9. INFO

- **Category** Suboptimal
- **Source** EulerV2Manager.sol

Description In case the call failed, the error message returned from the call is propagated as is, which in some cases could make it impossible to distinguish such propagated error from an error happened before or after the call. For example, the "functionCallWithValue" function may revert with the "AddressEmptyCode" error, but the same error could be returned from a failed call, and these two situations cannot be distinguished.

Recommendation Wrap error from inner call into a named error.

Client Comment *Given that this is an admin function, and won't be called frequently, the admins can debug the transaction manually with tools such as 'cast run' if necessary.*

```
100 +return Address.functionCallWithValue(aTarget, aCalldata, aValue);
```

CVF-10. INFO

- **Category** Suboptimal
- **Source** EulerV2Manager.sol

Description In case there is no vault for the token, the function silently returns zero.

Recommendation Revert in such a case.

Client Comment *This should not revert, as the pair will call 'getBalance' in '_syncManaged'. If there is no vault, it would mean there are no assets managed, so returning 0 instead of reverting is the correct behavior.*

```
128 +if (address(lVault) != address(0)) {
```

CVF-11. FIXED

- **Category** Procedural
- **Source** EulerV2Manager.sol

Description Silently modifying argument values is a bad practice, as it makes function less predictable and could hidh problems.

Recommendation Require a change amount to be zero in case the corresponding token has no value. Require change amounts to be non-positive in wind down mode.

```
157 +aAmount0Change = 0;
```

```
160 +aAmount1Change = 0;
```

```
165 +    aAmount0Change = 0;
```

```
168 +    aAmount1Change = 0;
```

CVF-12. FIXED

- **Category** Suboptimal
- **Source** EulerV2Manager.sol

Description The function allows returning either token0 or token1 but not both.

Recommendation Instead of a boolean flag and an amount argument, accept token0 amount and token1 amount. The function will become more efficient and more flexible.

```
236 +function returnAsset(bool aToken0, uint256 aAmount) external {
```

CVF-13. FIXED

- **Category** Suboptimal
- **Source** EulerV2Manager.sol

Description Effectively this checks aManaged*1e18/aReserve < lowerThreshold, which could lead to significant rounding errors in case of low managed balance.

Recommendation Check like this: aManager*1e18 < aReserve*lowerThreshold

```
245 +uint256 lRatio = aManaged.divWad(aReserve);
246 +if (lRatio < lowerThreshold) {
```

CVF-14. FIXED

- **Category** Suboptimal
- **Source** EulerV2Manager.sol

Description Effectively this checks aManaged*1e18/aReserve > upperThreshold, which could lead to significant rounding errors in case of low managed balance.

Recommendation Check like this: aManager*1e18 > aReserve*upperThreshold

```
245 +uint256 lRatio = aManaged.divWad(aReserve);
```

```
249 +} else if (lRatio > upperThreshold) {
```

CVF-15. FIXED

- **Category** Suboptimal
- **Source** EulerV2Manager.sol

Description This trick is unfair, as the last on in the list may get up to lLength-1 extra shares.

Recommendation Instead of implementing this trick, just do the following at the end of the loop above (and make the loop to iterate through all the pairs including the last one): lNewShares-=lNewSharesEntitled; lOldTotalShares-=oOldShares;

```
311 +// the last in the list will take all the remaining shares, and
    ↪ sometimes will get 1 or 2 more than they're
312 +// entitled to due to the rounding down in previous calculations
    ↪ for other pairs
313 +// this is to prevent the sum of each pair+token's shares not
    ↪ summing up to totalShares
```

CVF-16. INFO

- **Category** Procedural
- **Source** ReservoirPair.sol

Description Here two separate calls in a raw a made to the same smart contract.

Recommendation Merge two calls into one.

Client Comment *Not fixing as these are two distinct variables with different meanings. Combining both into one call is possible but will add complexity in the constructor logic.*

```
79 +factory.read(MAX_CHANGE_RATE_NAME).toUint128(), factory.read(
   ↪ MAX_CHANGE_PER_TRADE_NAME).toUint128()
```

9 Minor Issues

CVF-17. INFO

- **Category** Suboptimal
- **Source** Observation.sol

Description Referring to a test in production code looks odd.

Client Comment *Acknowledged.*

```
7  +// See `LogCompressionTest` for more info
```

CVF-18. INFO

- **Category** Procedural
- **Source** ConstantProductOracleMath.sol

Description We didn't review these files.

Client Comment *Acknowledged.*

```
17 17  import { FixedPointMathLib } from "solady/utils/FixedPointMathLib.
          ↪ sol";

19 19  import { LogCompression } from "src/libraries/LogCompression.sol";
```

CVF-19. INFO

- **Category** Procedural
- **Source** StableMath.sol

Description We didn't review this file.

Client Comment *Acknowledged.*

```
6  +import { FixedPointMathLib } from "solady/utils/FixedPointMathLib.
          ↪ sol";
```

CVF-20. FIXED

- **Category** Suboptimal
- **Source** StableMath.sol

Recommendation It would be more efficient to use right shift instead of division by 4.

```
130 +uint256 dP = ((D * D) / xp0).fullMulDiv(D, xp1) / 4;
```

CVF-21. FIXED

- **Category** Procedural
- **Source** StableOracleMath.sol

Recommendation Outer brackets are redundant.

```
58 +uint256 ay2 = ((a * reserve1).mulWad(reserve1));
```

```
65 +uint256 ax2 = ((a * reserve0).mulWad(reserve0));
```

CVF-22. INFO

- **Category** Procedural
- **Source** StablePair.sol

Description We didn't review this file.

Client Comment *Acknowledged.*

```
4 +import { FixedPointMathLib } from "solady/utils/FixedPointMathLib.
    ↪ sol";
```

```
10 +import { AmplificationData } from "src/structs/AmplificationData.
    ↪ sol";
```

CVF-23. INFO

- **Category** Suboptimal
- **Source** StablePair.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

Client Comment *Not fixing as reason for revert are clear enough in the corresponding require statements.*

```
27 +error InvalidA();
28 +error InvalidDuration();
29 +error AmpRateTooHigh();
30 +error NotSelf();
31 +error InvalidMintAmounts();
32 +error NonOptimalFeeTooLarge();
```

CVF-24. INFO

- **Category** Suboptimal
- **Source** StablePair.sol

Description Here, a value just written into the storage is read back.

Recommendation Reuse the written value.

Client Comment *No longer relevant as it is fixed in #26.*

```
43 +ampData.futureA = ampData.initialA;
```

```
48 +    ampData.initialA >= StableMath.MIN_A * uint64(StableMath.
   ↪ A_PRECISION)
49 +        && ampData.initialA <= StableMath.MAX_A * uint64(StableMath
   ↪ .A_PRECISION),
```

CVF-25. FIXED

- **Category** Procedural
- **Source** StablePair.sol

Recommendation This check should be performed earlier.

```
47 +require(
48 +    ampData.initialA >= StableMath.MIN_A * uint64(StableMath.
   ↪ A_PRECISION)
49 +        && ampData.initialA <= StableMath.MAX_A * uint64(StableMath
   ↪ .A_PRECISION),
50 +    InvalidA()
51 +);
```

CVF-26. FIXED

- **Category** Suboptimal
- **Source** StablePair.sol

Description These type conversions seem redundant.

Recommendation Remove them.

```
105 +lReserve0 += uint104(lFee0);
106 +lReserve1 += uint104(lFee1);
```

CVF-27. FIXED

- **Category** Procedural
- **Source** StablePair.sol

Recommendation Brackets around left multiplication are redundant.

```
198 +rToken1Fee = (swapFee * (aAmount1 - amount1Optimal)) / (2 *
   ↪ FEE_ACCURACY);
```

CVF-28. FIXED

- **Category** Procedural
- **Source** StablePair.sol

Recommendation This check should be performed earlier.

```
243 +require(aAmount != 0, AmountZero());
```

CVF-29. FIXED

- **Category** Documentation
- **Source** StablePair.sol

Description The semantics of the returned values is unclear.

Recommendation Give descriptive names to the returned values and/or explain in a documentation comment.

```
393 +returns (uint256, int256)
```

CVF-30. FIXED

- **Category** Documentation
- **Source** ConstantProductPair.sol

Description These comments are confusing.

Recommendation Elaborate more.

Client Comment *Fixed, added more explanatory comments.*

```
71 +uint256 lScaledTargetOwnership = lScaledMultiplier * aPlatformFee /
       ↪  FEE_ACCURACY; // ASSERT: < UINT144
72 +    // during maths, ends < UINT128
```

```
74 +rSharesToIssue = lScaledTargetOwnership * aCirculatingShares / (
       ↪ ACCURACY - lScaledTargetOwnership); // ASSERT:
75 +    // lScaledTargetOwnership < ACCURACY
```

CVF-31. INFO

- **Category** Procedural
- **Source** EulerV2Manager.sol

Description We didn't review these files.

Client Comment *Acknowledged.*

```
4  +import { ReentrancyGuardTransient as RGT } from "solady/utils/
       ↪ ReentrancyGuardTransient.sol";
5  +import { Owned } from "solmate/auth/Owned.sol";
6  +import { FixedPointMathLib } from "solady/utils/FixedPointMathLib.
       ↪ sol";
7  +import { SafeTransferLib } from "solady/utils/SafeTransferLib.sol";
```

```
14 +import { IERC4626 } from "lib/forge-std/src/interfaces/IERC4626.sol
       ↪ ";
```

CVF-32. INFO

- **Category** Procedural
- **Source** EulerV2Manager.sol

Recommendation The “newGuardian” parameter should be indexed.

Client Comment *These events are emitted very infrequently, and thus would not be searched a lot. The gas cost increase does not justify indexing these fields.*

```
20 +event Guardian(address newGuardian);
```

CVF-33. FIXED

- **Category** Procedural
- **Source** EulerV2Manager.sol

Recommendation The “asset” and “vault” parameters should be indexed.

```
22 +event VaultForAsset(IERC20 asset, IERC4626 vault);
```

CVF-34. FIXED

- **Category** Procedural
- **Source** EulerV2Manager.sol

Recommendation The “pair” and “token” parameters should be indexed.

```
24 +event Investment(IAssetManagedPair pair, IERC20 token, uint256
   ↪ shares);
25 +event Divestment(IAssetManagedPair pair, IERC20 token, uint256
   ↪ shares);
```

CVF-35. FIXED

- **Category** Unclear behavior
- **Source** EulerV2Manager.sol

Description These events are emitted even if nothing actually changed.

Client Comment *Fixed by adding extra conditions to ensure that event is only emitted when values differ from existing.*

```
76 +emit VaultForAsset(aAsset, aVault);
```

```
81 +emit Guardian(aGuardian);
```

```
86 +emit WindDownMode(aWindDown);
```

```
92 +emit Thresholds(aLowerThreshold, aUpperThreshold);
```

CVF-36. FIXED

- **Category** Procedural
- **Source** EulerV2Manager.sol

Description The bodies of these conditional statement are very similar.

Recommendation Merge them into a single conditional statement: if (lRatio < lowerThreshold || lRation > upperThreshold) { .... }

```
246 +if (lRatio < lowerThreshold) {
```

```
249 +} else if (lRatio > upperThreshold) {
```

CVF-37. INFO

- **Category** Suboptimal
- **Source** EulerV2Manager.sol

Recommendation It would be more efficient to pass a single array of structs with four fields, rather than four parallel arrays.

Client Comment *Not fixing as this interface is just a wrapper for calling merkl's contracts.*

```
262 +address[] calldata aUsers,
263 +address[] calldata aTokens,
264 +uint256[] calldata aAmounts,
265 +bytes32[][] calldata aProofs
```

CVF-38. INFO

- **Category** Bad datatype
- **Source** EulerV2Manager.sol

Recommendation The type for this argument should be "IERC20[]".

Client Comment *Not fixing as this interface is just a wrapper for calling merkl's contracts.*

```
263 +address[] calldata aTokens,
```

CVF-39. INFO

- **Category** Procedural
- **Source** ReservoirPair.sol

Description We didn't review these files.

Client Comment *Acknowledged.*

```
5  +import { ReentrancyGuardTransient as RGT } from "solady/utils/
       ↪ ReentrancyGuardTransient.sol";
```

```
13 +import { Buffer } from "src/libraries/Buffer.sol";
```

CVF-40. INFO

- **Category** Suboptimal
- **Source** ReservoirPair.sol

Description In ERC20 the decimals property is used by UI to render token amount in human-friendly format. Using this property in smart contracts is discouraged.

Recommendation Treat all token amounts are integers.

Client Comment *Like highlighted in the previous audit, this is necessary as we need it to calculate the price for stableswap, for various stablecoins with differing decimals.*

```
38 +// Multipliers for each pooled token's precision to get to
       ↪ POOL_PRECISION_DECIMALS. For example,
39 +// TBTC has 18 decimals, so the multiplier should be 1. WBTC has 8,
       ↪  so the multiplier should be
40 +// 10 ** 18 / 10 ** 8 => 10 ** 10.
41 +uint128 public immutable token0PrecisionMultiplier;
42 +uint128 public immutable token1PrecisionMultiplier;
```

CVF-41. INFO

- **Category** Procedural
- **Source** ReservoirPair.sol

Recommendation These errors could be made more helpful by adding certain parameters into them.

Client Comment *Not fixing as reason for revert are clear enough in the corresponding require statements.*

```
48 +error InvalidSwapFee();
49 +error InvalidPlatformFee();
50 +error InvalidTokenToRecover();
51 +error TransferFailed();
52 +error AssetManagerStillActive();
53 +error NotManager();
54 +error InvalidSkimToken();
55 +error InvalidChangePerSecond();
56 +error InvalidChangePerTrade();
```

```
58 +error InsufficientLiqMinted();
59 +error InsufficientLiq();
60 +error InsufficientAmtIn();
```

CVF-42. INFO

- **Category** Suboptimal
- **Source** ReservoirPair.sol

Description Merging several checks into a single “require”statement is a bad practice, as it makes it harder investigating problems.

Recommendation Use separate “require” statement for each check.

Client Comment *Separated the balance checks into different require statements.*
For token0Managed / token1Managed, the two conditions are logically linked and can be grouped together.
For change rate, max change per trade, those two conditions simply check if the input is within the bounds, so we will keep it as is.

```
108 +require(aBalance0 <= type(uint104).max && aBalance1 <= type(uint104
    ↪ ).max, Overflow());
109 +require(aReserve0 <= type(uint104).max && aReserve1 <= type(uint104
    ↪ ).max, Overflow());
```

```
354 -require(token0Managed == 0 && token1Managed == 0, "RP:␣
    ↪ AM_STILL_ACTIVE");
348 +require(token0Managed == 0 && token1Managed == 0,
    ↪ AssetManagerStillActive());
```

```
498 +require(0 < aMaxChangeRate && aMaxChangeRate <= MAX_CHANGE_PER_SEC,
    ↪  InvalidChangePerSecond());
499 +require(0 < aMaxChangePerTrade && aMaxChangePerTrade <=
    ↪ MAX_CHANGE_PER_TRADE, InvalidChangePerTrade());
```

CVF-43. INFO

- **Category** Documentation
- **Source** ReservoirPair.sol

Description This assert comment is confusing, as it is unclear where this condition is guaranteed to be true.

Recommendation Use normal “require” or “assert” statement.

Client Comment *Acknowledged*

```
134 +aBlockTimestampLast // assert: aBlockTimestampLast == lPrevious.
    ↪ timestamp
```

CVF-44. FIXED

- **Category** Bad naming
- **Source** ReservoirPair.sol

Recommendation Events are usually named via nouns, such as “ClampParams”.

```
476 +event ClampParamsUpdated(uint128 newMaxChangeRatePerSecond, uint128
       ↪  newMaxChangePerTrade);
```

CVF-45. FIXED

- **Category** Procedural
- **Source** ReservoirPair.sol

Recommendation Brackets are redundant.

```
519 +(lRateOfChangeWithinThreshold && lPerTradeWithinThreshold) ||
       ↪ aPreviousTimestamp == 0 // first ever calculation of the
       ↪ clamped price, and so should be set to the raw price
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting