Report

v. 2.0

Customer

ReservoirFi

Smart Contract Audit

AMM Core

22th June 2023

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	09.06.23	A. Zveryanskaya	Initial Draft
0.2	09.06.23	A. Zveryanskaya	Minor revision
1.0	12.06.23	A. Zveryanskaya	Release
1.1	22.06.23	A. Zveryanskaya	”Code with fixes” link updated
2.0	22.06.23	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 builds simple and efficient DeFi primitives to push the efficiency of DeFi forward.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

src/			
	ReservoirPair.sol	ReservoirERC20.sol	GenericFactory.sol
	ReservoirTimelock.sol		
src/curve/stable/			
	StablePair.sol	StableMintBurn.sol	
src/curve/constant-product/			
	ConstantProductPair.sol		
src/asset-management/			
	AaveManager.sol		
src/interfaces/			
	IAssetManagedPair.sol	IGenericFactory.sol	IAssetManager.sol
	IReservoirCallee.sol		
src/libraries/			
	StableMath.sol	ConstantProduct Math.sol	stdMath.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 17 major, and a few less important issues.

	Active	Fixed
Major	1	16
Moderate	2	9
Minor	20	46

Fixed 71 out of 94 issues

6 Major Issues

CVF-1. FIXED

- **Category** Flaw
- **Source** ConstantProductMath.sol

Description There is no range check for the ”aSwapFee” argument.

Recommendation Consider adding appropriate check.

Client Comment *Kept implementation but added documentation clarifying assumed usage. All our usage is compliant.*

```
7  function getAmountOut(uint256 aAmountIn, uint256 aReserveIn, uint256
       ↪  aReserveOut, uint256 aSwapFee)
```

```
21 function getAmountIn(uint256 aAmountOut, uint256 aReserveIn, uint256
       ↪  aReserveOut, uint256 aSwapFee)
```

CVF-2. FIXED

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

Description Phantom overflow is possible here, i.e. situation when the final calculation result would fit into the destination type, while some intermediary calculation overflows.

Recommendation Consider using the ”muldiv” function.

Client Comment *Kept implementation but added documentation clarifying assumed usage. All our usage is compliant.*

```
15 uint256 lAmountInWithFee = aAmountIn * (FEE_ACCURACY - aSwapFee);
   uint256 lNumerator = lAmountInWithFee * aReserveOut;
   uint256 lDenominator = aReserveIn * FEE_ACCURACY + lAmountInWithFee;
   rAmountOut = lNumerator / lDenominator;
```

```
29 uint256 lNumerator = aReserveIn * aAmountOut * FEE_ACCURACY;
30 uint256 lDenominator = (aReserveOut - aAmountOut) * (FEE_ACCURACY -
       ↪ aSwapFee);
   rAmountIn = lNumerator / lDenominator + 1;
```

CVF-3. FIXED

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

Recommendation Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type, while some intermediary calculation overflows.

Client Comment *Kept implementation but added documentation clarifying assumed usage. All our usage is compliant.*

```
18 return absDelta * 1e18 / b;
```

CVF-4. FIXED

- **Category** Procedural
- **Source** IAssetManagedPair.sol

Recommendation These functions should be declared as ”view”.

```
9  function token0Managed() external returns (uint104);
10 function token1Managed() external returns (uint104);
```

```
12 function token0() external returns (ERC20);
   function token1() external returns (ERC20);
```

```
15 function getReserves()
```

```
19 function assetManager() external returns (IAssetManager);
```

CVF-5. FIXED

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

Description Phantom overflow is possible here.

Recommendation Consider using the ”muldiv” function.

Client Comment *Added documentation showing that phantom flow does not occur for valid mints.*

```
45 uint256 amount1Optimal = (aAmount0 * aReserve1) / aReserve0;
```

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

```
50     uint256 amount0Optimal = (aAmount1 * aReserve0) / aReserve1;
       rToken0Fee = (swapFee * (aAmount0 - amount0Optimal)) / (2 *
          ↪ FEE_ACCURACY);
```

CVF-6. FIXED

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

Description Phantom overflow is possible here.

Recommendation Consider using the ”muldiv” function.

Client Comment *Document the circumstances under which phantom overflow may occur. While extremely unlikely we do support recovery via burn's try/catch.*

```
79 rLiquidity = ((lNewLiq - lOldLiq) * lTotalSupply) / lOldLiq;
```

CVF-7. FIXED

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

Description Phantom overflow is possible here.

Recommendation Consider using the ”muldiv” function.

```
106 rAmount0 = (liquidity * lReserve0) / lTotalSupply;
    rAmount1 = (liquidity * lReserve1) / lTotalSupply;
```

CVF-8. FIXED

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

Description Phantom overflow is possible here.

Recommendation Consider using the ”muldiv” function.

Client Comment *Believe this will not phantom overflow in all reasonable cases, added documentation for that. Furthermore we have the try/catch in burn as a safety mechanism.*

```
140 uint256 lNumerator = rTotalSupply * (rD - lDLast) * lPlatformFee;
    uint256 lDenominator = (FEE_ACCURACY - lPlatformFee) * rD +
       ↪ lPlatformFee * lDLast;
    uint256 lPlatformShares = lNumerator / lDenominator;
```

CVF-9. FIXED

- **Category** Suboptimal
- **Source** ConstantProductPair.sol

Description This line should be executed only when ”lSharesToIssue” is not zero.

```
83 address platformFeeTo = factory.read(PLATFORM_FEE_TO_NAME).toAddress
      ↪ ();
```

CVF-10. FIXED

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

Description There should be a function to set both thresholds atomically.

```
93 function setUpperThreshold(uint128 aUpperThreshold) external
      ↪ onlyOwner {
```

```
98 function setLowerThreshold(uint128 aLowerThreshold) external
      ↪ onlyOwner {
```

CVF-11. FIXED

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

Description In order for the result to be rounded down, the exchange rate ought to be rounded up.

```
124 rShares = aAmount.divWad(_getExchangeRate(aAaveToken));
```

CVF-12. INFO

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

Description In ERC20 the ”decimals” property is used by UI to render token amounts in a human-friendly way. Using this property in contracts is discouraged.

Recommendation Consider treating all token amounts as integers.

Client Comment *Token0/1 precision are used by both the oracle math and stable swap math components. The oracle uses this to reduce the possible range of values and achieve greater compression. The stable swap math uses this to normalize decimal differences between stable assets.*

```
64 token0PrecisionMultiplier = aNotStableMintBurn ? uint128(10) ** (18
   ↪ - aToken0.decimals()) : 0;
   token1PrecisionMultiplier = aNotStableMintBurn ? uint128(10) ** (18
   ↪ - aToken1.decimals()) : 0;
```

CVF-13. FIXED

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

Description The getter functions generated by the compiler for the public variables may become inconsistent with the corresponding virtual internal getters, when the internal getters will be overridden.

Recommendation Consider making the variables private and making the virtual getters public.

85
```
ERC20 public immutable token0;
ERC20 public immutable token1;
```

91
```
uint128 public immutable token0PrecisionMultiplier;
uint128 public immutable token1PrecisionMultiplier;
```

94
```
function _token0() internal view virtual returns (ERC20) {
```

98
```
function _token1() internal view virtual returns (ERC20) {
```

102
```
function _token0PrecisionMultiplier() internal view virtual returns
   ↪ (uint128) {
```

106
```
function _token1PrecisionMultiplier() internal view virtual returns
   ↪ (uint128) {
```

CVF-14. FIXED

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

Description The "_slot0" value is read from the storage twice. Once here and another time implicitly inside the "_writeSlot0Timestamp" function.

Recommendation Consider reusing already read value.

Client Comment *We pass Slot0 around as a storage pointer, this produced gas savings, thanks.*

136
```
Slot0 memory lSlot0 = _slot0;
```

147
```
_writeSlot0Timestamp(rBlockTimestampLast, true);
```

CVF-15. FIXED

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

Description These lines look confusing as balances are assigned to reserves and reserves are not used.

Recommendation Consider explaining the logic in a comment.

```
174 _slot0.reserve0 = uint104(aBalance0);
    _slot0.reserve1 = uint104(aBalance1);
```

CVF-16. FIXED

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

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type, while some intermediary calculation overflows.

Recommendation Consider using the ”muldiv” function.

```
536 rClampedPrice = aPrevClampedPrice * (1e18 + (maxChangeRate *
       ↪ aTimeElapsed)) / 1e18;
```

```
539 rClampedPrice = aPrevClampedPrice * (1e18 - (maxChangeRate *
       ↪ aTimeElapsed)) / 1e18;
```

CVF-17. FIXED

- **Category** Suboptimal
- **Source** GenericFactory.sol

Description Updating the free memory pointer on every loop iteration is waste of gas.

Recommendation Consider updating once after the loop.

Client Comment *We elected to perform the entire operation in Yul to avoid any memory ambiguity.*

```
81 mstore(0x40, add(lFreeMem, lSize))
```

7 Moderate Issues

CVF-18. FIXED

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

Description Over-/underflow is possible here.

Recommendation Consider using checked math.

Client Comment *We documented the call assumptions on the library.*

```
38 uint256 adjustedReserve0 = reserve0 * token0PrecisionMultiplier;
   uint256 adjustedReserve1 = reserve1 * token1PrecisionMultiplier;
40 uint256 feeDeductedAmountIn = amountIn - (amountIn * swapFee) /
      ↪ ONE_HUNDRED_PERCENT;
```

```
44     uint256 x = adjustedReserve0 + (feeDeductedAmountIn *
          ↪ token0PrecisionMultiplier);
```

```
46     dy = adjustedReserve1 - y - 1;
```

```
49     uint256 x = adjustedReserve1 + (feeDeductedAmountIn *
          ↪ token1PrecisionMultiplier);
```

```
51     dy = adjustedReserve0 - y - 1;
```

CVF-19. FIXED

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

Description Over-/underflow is possible here.

Recommendation Consider using checked math.

Client Comment *We documented the call assumptions on the library.*

```
68 uint256 adjustedReserve0 = reserve0 * token0PrecisionMultiplier;
   uint256 adjustedReserve1 = reserve1 * token1PrecisionMultiplier;

73     uint256 y = adjustedReserve0 - amountOut *
           ↪ token0PrecisionMultiplier;

75     dx = x - adjustedReserve1 + 1;

78     uint256 y = adjustedReserve1 - amountOut *
           ↪ token1PrecisionMultiplier;

80     dx = x - adjustedReserve0 + 1;

85 dx = dx * (ONE_HUNDRED_PERCENT + swapFee) / ONE_HUNDRED_PERCENT;
```

CVF-20. FIXED

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

Description Over-/underflow is possible here.

Recommendation Consider using checked math or clearly explaining why overflow isn't possible or is desired.

Client Comment *Added comment explaining why overflow is desired/necessary.*

```
306 int112 logAccRawPrice = previous.logAccRawPrice + currLogRawPrice *
        ↪ int112(int256(uint256(aTimeElapsed)));

308     previous.logAccClampedPrice + int56(currLogClampedPrice) * int56
            ↪ (int256(uint256(aTimeElapsed)));
    int56 logAccLiq = previous.logAccLiquidity + int56(currLogLiq) *
        ↪ int56(int256(uint256(aTimeElapsed)));
310 _slot0.index += 1;
```

CVF-21. FIXED

- **Category** Suboptimal
- **Source** ConstantProductPair.sol

Description This formula assumes that aSqrtNewK >= aSqrtOldK, however this fact wasn't listed in the asserts above.

Recommendation Consider either listing it as an assert or checking it explicitly.

Client Comment *Added the additional assumption and improved the wording of the invariants.*

```
64 uint256 lScaledMultiplier = ACCURACY - (SQUARED_ACCURACY /
   ↪ lScaledGrowth); // ASSERT: < UINT128
```

CVF-22. FIXED

- **Category** Flaw
- **Source** ConstantProductPair.sol

Description There are no checks to guarantee that "lSqrtNewK" and "lSqrtOldK" do fit into 104 bits, while the "_calcFee" function relies on this.

Recommendation Consider adding appropriate checks or clearly explaining why such checks are not necessary.

Client Comment *Add short note at the callsite about the invariants being upheld.*

```
81 uint256 lSharesToIssue = _calcFee(lSqrtNewK, lSqrtOldK, platformFee,
   ↪  totalSupply);
```

CVF-23. FIXED

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

Description Phantom overflow is possible here, i.e. a situation when the final calculation result would fit into the destination type, while some intermediary calculation overflows.

Recommendation Consider using the "muldiv" function.

Client Comment *Added documentation of the bounds highlighting that phantom overflow is not possible for all valid mints.*

```
107 rLiquidity = Math.min(lAmount0 * lTotalSupply / lReserve0, lAmount1
    ↪ * lTotalSupply / lReserve1);
```

CVF-24. FIXED

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

Description Overflow is possible here.

Recommendation Consider using safe conversion.

```
270 int256 lAmount0Change = -int256(aToken0 ? aAmount : 0);
    int256 lAmount1Change = -int256(aToken0 ? 0 : aAmount);
```

CVF-25. FIXED

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

Description Overflow is possible here.

Recommendation Consider using safe conversion.

```
278 rAmountChange = int256(aReserve.mulWad(uint256(lowerThreshold).avg(
      ↪ upperThreshold)) - aManaged);
```

```
281 rAmountChange = int256(aReserve.mulWad(uint256(lowerThreshold).avg(
      ↪ upperThreshold))) - int256(aManaged);
```

CVF-26. FIXED

- **Category** Flaw
- **Source** ReservoirPair.sol

Description The upper bit is not cleared in ”aTimestamp” nor it is checked to be zero.

Client Comment *Our use of the _currentTime() wrapper function prevents the MSB from being set/used. However, we have added a dev comment clarifying this assumption.*

```
129 _slot0.packedTimestamp = aTimestamp | lLocked;
```

CVF-27. INFO

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

Description Transferring two tokens in a single transaction is a bad idea, as in case one transfer fails, the other transfer won't happen as well.

Recommendation Consider implementing some way to skim tokens separately.

Client Comment *If either token is frozen the AMM is inoperable. Additionally, there is no correlation between token0's balance in the pool and the frozen status of token1. We will keep the behaviour in line with Uniswap V2.*

206
```
_checkedTransfer(_token0(), aTo, _totalToken0() - lReserve0,
    ↪ lReserve0, lReserve1);
_checkedTransfer(_token1(), aTo, _totalToken1() - lReserve1,
    ↪ lReserve0, lReserve1);
```

CVF-28. INFO

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

Description Transferring two tokens in a single transaction is a bad idea, as in case one transfer fails, the other transfer won't happen as well.

Recommendation Consider implementing some way to skim tokens separately.

Client Comment *If either token is frozen the AMM is inoperable. Additionally, there is no correlation between token0's balance in the pool and the frozen status of token1. We will keep the behaviour in line with Uniswap V2.*

437
```
address(_token0()).safeTransfer(msg.sender, lDelta);
```

444
```
address(_token0()).safeTransferFrom(msg.sender, address(this),
    ↪ lDelta);
```

453
```
address(_token1()).safeTransfer(msg.sender, lDelta);
```

460
```
address(_token1()).safeTransferFrom(msg.sender, address(this),
    ↪ lDelta);
```

8 Minor Issues

CVF-29. FIXED

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

Recommendation The value ”1e18” should be a named constant.

```
18 return absDelta * 1e18 / b;
```

CVF-30. INFO

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

Description We didn’t review this file.

Client Comment *Acknowledged.*

```
4 import { MathUtils } from "src/libraries/MathUtils.sol";
```

CVF-31. INFO

- **Category** Readability
- **Source** StableMath.sol

Recommendation It would be more readable to specify these values as ”0.01e2” and ”100e2”.

Client Comment *These values are not ratios, but rather absolute values.*

```
16 uint256 public constant MIN_A = 1;
```

```
18 uint256 public constant MAX_A = 10_000;
```

CVF-32. INFO

- **Category** Readability
- **Source** StableMath.sol

Recommendation This formula could be reduced by D.

Client Comment *Due to integer precision, removing this D term would not result in equivalent formulas.*

```
106 D = (((N_A * s) / A_PRECISION + 2 * dP) * D) / ((N_A - A_PRECISION)
      ↪ * D / A_PRECISION + 3 * dP);
```

CVF-33. FIXED

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

Recommendation This value should be a named constant.

Client Comment *This has been removed as we now try catch the burn operations to guarantee asset recovery in the face of phantom overflow/non-convergence.*

```
114 if (percentDelta <= 0.0000000000004e18) {
```

CVF-34. INFO

- **Category** Procedural
- **Source** IAssetManager.sol

Description We didn't review this file.

Client Comment *Acknowledged.*

```
4 import { ERC20 } from "solmate/tokens/ERC20.sol";
```

CVF-35. FIXED

- **Category** Documentation
- **Source** IAssetManager.sol

Description The semantics of these functions is unclear.

Recommendation Consider documenting.

Client Comment *Natspec added.*

```
10 function afterLiquidityEvent() external;
   function returnAsset(bool aToken0, uint256 aAmount) external;
```

CVF-36. INFO

- **Category** Procedural
- **Source** IAssetManagedPair.sol

Description We didn't review this file.

Client Comment *Acknowledged.*

```
4 import { ERC20 } from "solmate/tokens/ERC20.sol";
```

CVF-37. FIXED

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

Description This function should emit some event and such event should be declared in this interface.

```
20 function setManager(IAssetManager manager) external;
```

CVF-38. FIXED

- **Category** Documentation
- **Source** IReservoirCallee.sol

Description The arguments ”sender” and ”data” are not documented.

Recommendation Consider documenting.

```
7 function reservoirCall(address sender, int256 amount0, int256
    ↪ amount1, bytes calldata data) external;
```

CVF-39. INFO

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

Description The return type should be more specific.

Client Comment *Added stableMintBurn return type, however, left getPair & createPair as address as these are instantiated from bytecode blobs and are not guaranteed to conform to a particular interface.*

```
5 function stableMintBurn() external view returns (address);
```

```
12 function allPairs() external view returns (address[] memory);
function getPair(address tokenA, address tokenB, uint256 curveId)
    ↪ external view returns (address);
function createPair(address tokenA, address tokenB, uint256 curveId)
    ↪  external returns (address);
```

CVF-40. FIXED

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

Recommendation The type of the token arguments should be more specific.

```
13 function getPair(address tokenA, address tokenB, uint256 curveId)
    ↪ external view returns (address);
function createPair(address tokenA, address tokenB, uint256 curveId)
    ↪  external returns (address);
```

CVF-41. INFO

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

Description We didn't review these files.

Client Comment *Acknowledged.*

```
5  import { ERC20 } from "solmate/tokens/ERC20.sol";

10 import { Bytes32Lib } from "src/libraries/Bytes32.sol";
   import { FactoryStoreLib } from "src/libraries/FactoryStore.sol";

15 import { StableOracleMath } from "src/libraries/StableOracleMath.sol
      ↪ ";
   import { ConstantProductOracleMath } from "src/libraries/
      ↪ ConstantProductOracleMath.sol";
```

CVF-42. FIXED

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

Description Declaring a top-level struct in a file named after a contract makes it harder to navigate through code.

Recommendation Consider either moving the declaration into the contract or moving it into a separate file.

Client Comment *We have moved the struct declaration to a new file, moving it within the contract would break tests.*

```
18 struct AmplificationData {
```

CVF-43. FIXED

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

Description The expression "_isStableMintBurn(aToken0, aToken1)" is calculated twice.

Recommendation Consider calculating once and reusing.

```
52 MINT_BURN_LOGIC = _isStableMintBurn(aToken0, aToken1) ? address(0) :
      ↪  factory.stableMintBurn();
```

```
54 if (!_isStableMintBurn(aToken0, aToken1)) {
```

CVF-44. FIXED

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

Description Division may lead to precision loss.

Recommendation Consider calculating as: lFutureAPrecise * 1 days<= lCurrentAPrecise * duration * StableMath.MAX_AMP_UPDATE_DAILY_RATE

```
85 uint256 dailyRate = lFutureAPrecise > lCurrentAPrecise
       ? Math.ceilDiv(lFutureAPrecise * 1 days, lCurrentAPrecise *
          ↪ duration)
       : Math.ceilDiv(lCurrentAPrecise * 1 days, lFutureAPrecise *
          ↪ duration);
   require(dailyRate <= StableMath.MAX_AMP_UPDATE_DAILY_RATE, "SP:
      ↪ AMP_RATE_TOO_HIGH");
```

CVF-45. INFO

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

Recommendation Per second rate would be more efficient.

Client Comment *Acknowledged but will leave as is for readability because the code is rarely run.*

```
88 require(dailyRate <= StableMath.MAX_AMP_UPDATE_DAILY_RATE, "SP:
      ↪ AMP_RATE_TOO_HIGH");
```

CVF-46. FIXED

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

Description This logic is executed in both branches.

Recommendation Consider executing before the conditional statement.

```
121     returndatacopy(0, 0, returndatasize())
```

```
125 returndatacopy(0, 0, returndatasize())
```

CVF-47. INFO

- **Category** Procedural
- **Source** StableMintBurn.sol

Recommendation This contract could be turned into a library to make usage pattern simpler and safer.

Client Comment *Due to the need of most functions to read contract storage slots, implementation as a library was deemed non-feasible.*

```
13 contract StableMintBurn is StablePair {
```

CVF-48. FIXED

- **Category** Procedural
- **Source** StableMintBurn.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
20 constructor() StablePair(ERC20(address(0)), ERC20(address(0))) { }
```

CVF-49. FIXED

- **Category** Procedural
- **Source** StableMintBurn.sol

Recommendation Brackets around multiplication are redundant.

```
45  uint256 amount1Optimal = (aAmount0 * aReserve1) / aReserve0;
```

```
50      uint256 amount0Optimal = (aAmount1 * aReserve0) / aReserve1;
        rToken0Fee = (swapFee * (aAmount0 - amount0Optimal)) / (2 *
           ↪ FEE_ACCURACY);
```

```
79      rLiquidity = ((lNewLiq - lOldLiq) * lTotalSupply) / lOldLiq;
```

```
106 rAmount0 = (liquidity * lReserve0) / lTotalSupply;
    rAmount1 = (liquidity * lReserve1) / lTotalSupply;
```

CVF-50. FIXED

- **Category** Suboptimal
- **Source** StableMintBurn.sol

Description This conditional statement doesn't save gas.

```
152 } else if (lastInvariant != 0) {
```

CVF-51. INFO

- **Category** Procedural
- **Source** ConstantProductPair.sol

Description We didn't review these files.

Client Comment *Acknowledged.*

```
5  import { FixedPointMathLib } from "solady/utils/FixedPointMathLib.
       ↪ sol";
   import { ERC20 } from "solmate/tokens/ERC20.sol";
```

```
8  import { Bytes32Lib } from "src/libraries/Bytes32.sol";
   import { FactoryStoreLib } from "src/libraries/FactoryStore.sol";
```

```
11 import { ConstantProductOracleMath } from "src/libraries/
       ↪ ConstantProductOracleMath.sol";
```

CVF-52. FIXED

- **Category** Procedural
- **Source** ConstantProductPair.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

```
32 constructor(ERC20 aToken0, ERC20 aToken1) ReservoirPair(aToken0,
       ↪ aToken1, PAIR_SWAP_FEE_NAME, true) { }
```

CVF-53. FIXED

- **Category** Suboptimal
- **Source** ConstantProductPair.sol

Recommendation This conditional statement doesn't save any gas, as overwriting a storage slot with the value this slot already have costs the same as reading the slot.

```
87 } else if (kLast != 0) {
```

CVF-54. FIXED

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

Description The name ”aInOrOut” is confusing, as it is unclear whether ”true” means ”in” or ”out”.

Recommendation Consider renaming.

```
147 function swap(int256 aAmount, bool aInOrOut, address aTo, bytes
      ↪ calldata aData)
```

CVF-55. FIXED

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

Description Phantom overflow is possible here.

Recommendation Consider surrounding with an ”unchecked” block.

```
183 rAmountOut = uint256(-aAmount);
```

CVF-56. FIXED

- **Category** Procedural
- **Source** ConstantProductPair.sol

Description This comment should be either resolved or removed. Also, it is unclear, what is ”clamped” price.

Recommendation Consider clarifying.

Client Comment *Removed comment, clamp price is documented in ReservoirPair.sol*

```
225 // perf: see if we can avoid using prevClampedPrice and read the two
      ↪  previous oracle observations
    // to figure out the previous clamped price
```

CVF-57. INFO

- **Category** Procedural
- **Source** AaveManager.sol

Description We didn't review these files.

Client Comment *Acknowledged.*

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

CVF-58. INFO

- **Category** Procedural
- **Source** AaveManager.sol

Description We didn't review these files.

Client Comment *Acknowledged.*

```
12 import { IPoolAddressesProvider } from "src/interfaces/aave/
     ↪ IPoolAddressesProvider.sol";
   import { IPool } from "src/interfaces/aave/IPool.sol";
   import { IAaveProtocolDataProvider } from "src/interfaces/aave/
     ↪ IAaveProtocolDataProvider.sol";
   import { IRewardsController } from "src/interfaces/aave/
     ↪ IRewardsController.sol";
```

CVF-59. FIXED

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

Recommendation Events are usually named via nouns, such as "Investment".

```
20 event FundsInvested(IAssetManagedPair pair, ERC20 token, uint256
     ↪ shares);
   event FundsDivested(IAssetManagedPair pair, ERC20 token, uint256
     ↪ shares);
```

CVF-60. FIXED

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

Recommendation The argument type should be ”IPoolAddressProvider”.

```
56 constructor(address aPoolAddressesProvider) {
```

CVF-61. FIXED

- **Category** Suboptimal
- **Source** AaveManager.sol

Recommendation This check is redundant, as it is anyway possible to pass a dead address provider.

```
57 require(aPoolAddressesProvider != address(0), "AM:
   ↪ PROVIDER_ADDRESS_ZERO");
```

CVF-62. FIXED

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

Description These functions should emit some events.

```
67 function updatePoolAddress() public onlyOwner {
```

```
73 function updateDataProviderAddress() public onlyOwner {
```

```
79 function setRewardSeller(address aRewardSeller) external onlyOwner {
```

```
84 function setRewardsController(address aRewardsController) external
   ↪ onlyOwner {
```

```
89 function setWindDownMode(bool aWindDown) external onlyOwner {
```

```
93 function setUpperThreshold(uint128 aUpperThreshold) external
   ↪ onlyOwner {
```

```
98 function setLowerThreshold(uint128 aLowerThreshold) external
   ↪ onlyOwner {
```

CVF-63. FIXED

- **Category** Suboptimal
- **Source** AaveManager.sol

Recommendation The first par of the condition is redundant.

```
99 require(aLowerThreshold <= 1e18 && aLowerThreshold <= upperThreshold
     ↪ , "AM: INVALID_THRESHOLD");
```

CVF-64. FIXED

- **Category** Suboptimal
- **Source** AaveManager.sol

Recommendation It would be more precise to calculate this as: aAmount * lTotalShares / lAaveUnderlying

```
124 rShares = aAmount.divWad(_getExchangeRate(aAaveToken));
```

```
133 rShares = aAmount.divWadUp(_getExchangeRate(aAaveToken));
```

CVF-65. FIXED

- **Category** Procedural
- **Source** AaveManager.sol

Description The value ”shares[aPair][aToken]” is read from the storage several times.

Recommendation Consider reading once and reusing.

```
136 if (rShares > shares[aPair][aToken]) {
        rShares = shares[aPair][aToken];
```

```
140 shares[aPair][aToken] -= rShares;
```

CVF-66. FIXED

- **Category** Suboptimal
- **Source** AaveManager.sol

Recommendation It would be more precise to calculate this as: shares[aOwner][aToken] * lAaveUnderlying / lTotalShares

```
168 rTokenBalance = shares[aOwner][aToken].mulWad(_getExchangeRate(
        ↪ lAaveToken));
```

CVF-67. INFO

- **Category** Suboptimal
- **Source** AaveManager.sol

Recommendation Actually, the cast won't overflow. Negation will overflow, but this could be solved with "unchecked" block like this: int256 x; if (x < 0) { uint256 absX; unchecked { absX = uint256 (-x); } }

```
187 require(aAmount0Change != type(int256).min && aAmount1Change != type
        ↪ (int256).min, "AM: CAST_WOULD_OVERFLOW");
```

CVF-68. INFO

- **Category** Procedural
- **Source** ReservoirERC20.sol

Description We didn't review this file.

Client Comment *Acknowledged.*

```
4 import { ERC20 } from "solmate/tokens/ERC20.sol";
```

CVF-69. FIXED

- **Category** Procedural
- **Source** ReservoirERC20.sol

Recommendation It is a good practice to put a comment into empty block to explain why the block is empty.

```
7 contract ReservoirERC20 is ERC20("Reservoir LP Token", "RES-LP", 18)
    ↪ { }
```

CVF-70. INFO

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

Description We didn't review these files.

Client Comment *Acknowledged.*

```
5 import { SafeTransferLib } from "solady/utils/SafeTransferLib.sol";
```

```
8 import { FactoryStoreLib } from "src/libraries/FactoryStore.sol";
9 import { Bytes32Lib } from "src/libraries/Bytes32.sol";
10 import { LogCompression } from "src/libraries/LogCompression.sol";
```

CVF-71. FIXED

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

Description Declaring a top-level struct in a file named after a contract makes it harder to navigate through the code.

Recommendation Consider either moving the struct declarations into the contract or moving them into a separate file.

```
18 struct Slot0 {
```

```
25 struct Observation {
```

CVF-72. FIXED

- **Category** Readability
- **Source** ReservoirPair.sol

Recommendation ”1e3” or ”1000” would be shorter.

```
49 uint256 public constant MINIMUM_LIQUIDITY = 10 ** 3;
```

CVF-73. FIXED

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

Recommendation The ”encodePacked” call is redundant. Just cast ”aSwapFeeName” to ”bytes”.

```
66 swapFeeName = keccak256(abi.encodePacked(aSwapFeeName));
```

CVF-74. FIXED

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

Recommendation Binary ”AND” would be more efficient here.

```
119 return uint32(block.timestamp % 2 ** 31);
```

CVF-75. FIXED

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

Recommendation Only underflow is possible here, not overflow.

```
168 lTimeElapsed = lBlockTimestamp - aBlockTimestampLast; // overflow is
    ↪ desired
```

CVF-76. FIXED

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

Recommendation Events are usually named via nouns, such as ”SwapFee”, ”CustomSwapFee”, etc.

```
217 event SwapFeeChanged(uint256 oldSwapFee, uint256 newSwapFee);
    event CustomSwapFeeChanged(uint256 oldCustomSwapFee, uint256
       ↪ newCustomSwapFee);
    event PlatformFeeChanged(uint256 oldPlatformFee, uint256
       ↪ newPlatformFee);
220 event CustomPlatformFeeChanged(uint256 oldCustomPlatformFee, uint256
       ↪  newCustomPlatformFee);
```

```
355 event ProfitReported(ERC20 token, uint256 amount);
    event LossReported(ERC20 token, uint256 amount);
```

```
488 event OracleCallerUpdated(address oldCaller, address newCaller);
    event MaxChangeRateUpdated(uint256 oldMaxChangePerSecond, uint256
       ↪ newMaxChangePerSecond);
```

CVF-77. FIXED

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

Recommendation The old values are redundant, as they could be derived from the previous events.

```
217 event SwapFeeChanged(uint256 oldSwapFee, uint256 newSwapFee);
    event CustomSwapFeeChanged(uint256 oldCustomSwapFee, uint256
       ↪ newCustomSwapFee);
    event PlatformFeeChanged(uint256 oldPlatformFee, uint256
       ↪ newPlatformFee);
220 event CustomPlatformFeeChanged(uint256 oldCustomPlatformFee, uint256
       ↪  newCustomPlatformFee);
```

```
488 event OracleCallerUpdated(address oldCaller, address newCaller);
    event MaxChangeRateUpdated(uint256 oldMaxChangePerSecond, uint256
       ↪ newMaxChangePerSecond);
```

CVF-78. FIXED

- **Category** Readability
- **Source** ReservoirPair.sol

Recommendation ”0.02e6” would be more readable.

```
229 uint256 public constant MAX_SWAP_FEE = 20_000;
```

CVF-79. FIXED

- **Category** Readability
- **Source** ReservoirPair.sol

Recommendation ”1e6” would be more readable.

```
236 uint256 public constant MAX_PLATFORM_FEE = 1_000_000;
```

CVF-80. INFO

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

Description These events are emitted even if nothing actually changed.

Client Comment *Acknowledged, as this function is authenticated we are not worried about spam.*

```
243 emit CustomSwapFeeChanged(customSwapFee, aCustomSwapFee);
```

```
250 emit CustomPlatformFeeChanged(customPlatformFee, aCustomPlatformFee)
    ↪ ;
```

CVF-81. FIXED

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

Recommendation The argument type should be ”ERC20”.

```
277 function recoverToken(address aToken) external {
```

CVF-82. FIXED

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

Recommendation The type of the ”aToken” argument should be ”ERC20”.

```
291 function _safeTransfer(address aToken, address aTo, uint256 aValue)
      ↪ internal returns (bool) {
```

CVF-83. FIXED

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

Description This function should emit some event.

```
360 function setManager(IAssetManager manager) external onlyFactory {
```

CVF-84. INFO

- **Category** Procedural
- **Source** GenericFactory.sol

Description We didn’t review these files.

Client Comment *Acknowledged.*

```
6 import { SSTORE2 } from "solady/utils/SSTORE2.sol";
import { Owned } from "solmate/auth/Owned.sol";
```

```
9 import { Bytes32Lib } from "src/libraries/Bytes32.sol";
```

CVF-85. FIXED

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

Recommendation The type of this variable should be ”StableMintBurn”.

```
19 address public immutable stableMintBurn;
```

CVF-86. FIXED

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

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

```
131 event PairCreated(address indexed token0, address indexed token1,
    ↪ uint256 curveId, address pair);
```

CVF-87. FIXED

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

Recommendation The type of the token parameters should be ”ERC20”.

```
131 event PairCreated(address indexed token0, address indexed token1,
    ↪ uint256 curveId, address pair);
```

CVF-88. FIXED

- **Category** Documentation
- **Source** GenericFactory.sol

Description The semantics of the keys is unclear.

Recommendation Consider documenting.

```
133 mapping(address => mapping(address => mapping(uint256 => address)))
    ↪ public getPair;
```

CVF-89. INFO

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

Recommendation The value type should be ”IAssetManagedPair”.

Client Comment *Because pairs are not guaranted to confirm to any particular interface, we have left it as address.*

```
133 mapping(address => mapping(address => mapping(uint256 => address)))
    ↪ public getPair;
```

CVF-90. INFO

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

Recommendation The type of the ”pair” parameter should be ”IAssetManagedPair”.

Client Comment *Because pairs are not guaranted to confirm to any particular interface, we have left it as address.*

```
131 event PairCreated(address indexed token0, address indexed token1,
      ↪ uint256 curveId, address pair);
```

CVF-91. INFO

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

Recommendation The type of this array should be ”IAssetManagedPair[]”.

Client Comment *Because pairs are not guaranted to confirm to any particular interface, we have left it as address.*

```
134 address[] private _allPairs;
```

CVF-92. FIXED

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

Recommendation The type of the arguments and the returned values should be ”ERC20”.

```
140 function _sortAddresses(address a, address b) private pure returns (
      ↪ address r0, address r1) {
```

CVF-93. FIXED

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

Recommendation The type of the token arguments should be ”ERC20”.

```
144 function createPair(address aTokenA, address aTokenB, uint256
      ↪ aCurveId) external returns (address rPair) {
```

CVF-94. FIXED

- **Category** Procedural
- **Source** ReservoirTimelock.sol

Recommendation These checks could be done via a modifier.

```
12 require(msg.sender == admin, "RT: ADMIN");
```

```
19 require(msg.sender == admin, "RT: ADMIN");
```

```
26 require(msg.sender == admin, "RT: ADMIN");
```

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