Report

v. 1.0

Customer

Deltaprime

Smart Contract Audit

Primeloans

26th June 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	26.06.24	A. Zveryanskaya	Initial Draft
0.2	26.06.24	A. Zveryanskaya	Minor revision
1.0	26.06.24	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.

DeltaPrime is a decentralized borrowing and investing ecosystem. Customers can lend or invest borrowed funds across the most popular protocols on Avalanche and Arbitrum, with a minimum collateral ratio of 20%.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

/			
	Prime_L2.sol	Prime.sol	PrimeBridge.sol
	sPrime.sol	sPrimeUniswap.sol	vPrime.sol
	vPrimeController.sol		
NonfungibleNFT/			
	PositionManager.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.
- **Recommendations** contain code style, best practices and other suggestions.

5 Our findings

We found 5 major, and a few less important issues. All identified Major issues have been fixed.

Fixed 5 out of 5 issues

6 Major Issues

CVF-2 FIXED

- **Category** Flaw
- **Source** sPrimeUniswap.sol

Description There is no check to ensure that the pool actually does exist.

Recommendation Consider adding an appropriate check.

```
69 address poolAddress = IUniswapV3Factory(getUniV3FactoryAddress()).
      ↪ getPool(tokenX_, tokenY_, feeTier_);
```

CVF-3 FIXED

- **Category** Overflow/Underflow
- **Source** sPrimeUniswap.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.

```
143 fullyVestedBalance += locks[account][i].amount * locks[account][i].
      ↪ lockPeriod / MAX_LOCK_TIME;
```

```
368 liquidity: uint128(liquidity * share / balanceOf(_msgSender())),
```

CVF-4 FIXED

- **Category** Overflow/Underflow
- **Source** vPrimeController.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.

```
124 fullyVestedDollarValue += fullyVestedBalance * prices[i] * 1e10 / 10
       ↪  ** whitelistedPools[i].decimals();
    nonVestedDollarValue += nonVestedBalance * prices[i] * 1e10 / 10 **
       ↪ whitelistedPools[i].decimals();
```

```
141 uint256 poolDollarValue = poolBorrowedAmount * prices[i] * 1e10 / 10
       ↪  ** whitelistedPools[i].decimals();
```

```
159 fullyVestedDollarValue += userSPrimeValueInTokenY *
       ↪ sPrimeTokenYPrice * 1e10 * fullyVestedDollarValue /
       ↪ sPrimeBalance / 10 ** sPrimeTokenYDecimals;
160 nonVestedBalance += userSPrimeValueInTokenY * sPrimeTokenYPrice * 1
       ↪ e10 * nonVestedDollarValue / sPrimeBalance / 10 **
       ↪ sPrimeTokenYDecimals;
```

CVF-5 FIXED

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

Recommendation The value “10” should be a named constant.

```
178     (userBorrowedDollarValue / 10,
```

```
204 uint256 depositVestedPairsCount = Math.min(
       ↪ userDepositFullyVestedDollarValue / 10,
       ↪ userSPrimeDollarValueFullyVested) / 1e18;
    uint256 depositNonVestedPairsCount = Math.min(
       ↪ userDepositNonVestedDollarValue / 10,
       ↪ userSPrimeDollarValueNonVested) / 1e18;
```

CVF-6 FIXED

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

Description Overflow is possible when converting to “int256”.

Recommendation Consider using safe conversion.

```
54 int256 votesDiff = int256(currentVotesBalance) - int256(
   ↪ lastRecordedVotes);
```

7 Moderate Issues

CVF-1 INFO

- **Category** Flaw
- **Source** PrimeBridge.sol

Description Relying on token balance changes of an untrusted address could be very dangerous, as tokens may implement various kinds of inbound transfer hooks, that would allow the recipient to adjust the balance before the sender could check it.

Recommendation Consider not relying on the "_to" balance here.

Client Comment *The bridge contract will only be used with our PRIME token contract. We know the exact code of this contract so there is no need to treat balanceOf() method as unreliable as PrimeBridge will not be used with any other tokens.*

```
88 uint before = innerToken.balanceOf(_to);
```

```
94 return innerToken.balanceOf(_to) - before;
```

CVF-7 INFO

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

Description In case the bin already have some deposits, the deposit IDs and liquidity configs passed within the "params" argument are silently ignored.

Recommendation Consider ensuring that the passed arrays are empty in such a case.

```
91 if(binInfo.depositIds.length == 0) {
       binInfo.depositIds = params.depositIds;
       binInfo.liquidityConfigs = params.liquidityConfigs;
   }
```

CVF-8 INFO

- **Category** Flaw
- **Source** PositionManager.sol

Description There is no length check for “params.liquidityAmounts”.

Recommendation Consider adding such a check.

```
106 position.liquidityMinted[i] += params.liquidityAmounts[i];
```

```
111 position.liquidityMinted[i] -= params.liquidityAmounts[i];
```

CVF-9 INFO

- **Category** Suboptimal
- **Source** Prime_L2.sol

Description Handling the _from == spender case is a special way makes the contract behavior more complicated and less predictable. Also, it makes the contract itself more complicated and less efficient.

Recommendation Consider not handling this case separately.

```
44 if (_from != spender) _spendAllowance(_from, spender, _amount);
```

CVF-10 INFO

- **Category** Suboptimal
- **Source** Prime_L2.sol

Description This check makes the contract behavior more complicated and less predictable.

Recommendation Consider always spending allowance.

```
65 if (_from != address(this) && _from != spender) _spendAllowance(
   ↪ _from, spender, _amount);
```

CVF-11 INFO

- **Category** Suboptimal
- **Source** Prime_L2.sol

Description The signature of this function assumes that the rate is always an integer, which is inflexible.

Recommendation Consider supporting fractional rate at the interface level even if in this particular implementation the rate is always integer.

```
70 function _ld2sdRate() internal view virtual override returns (uint)
       ↪ {
```

CVF-12 INFO

- **Category** Suboptimal
- **Source** PrimeBridge.sol

Description This code looks like an overkill. It would make sense if in case the token contract doesn't implement the "decimals" function, some kind of default decimals would be used. But here transaction is just reverted in such a case.

Recommendation Consider just calling "decimals" normally.

```
24 (bool success, bytes memory data) = _token.staticcall(abi.
       ↪ encodeWithSignature("decimals()"));
   require(success, "ProxyOFT:␣failed␣to␣get␣token␣decimals");
   uint8 decimals = abi.decode(data, (uint8));
```

CVF-13 INFO

- **Category** Suboptimal
- **Source** PrimeBridge.sol

Description The signature of this function assumes that the rate is always an integer, which is inflexible.

Recommendation Consider supporting fractional rate at the interface level even if in this particular implementation the rate is always integer.

```
97 function _ld2sdRate() internal view virtual override returns (uint)
       ↪ {
```

CVF-14 INFO

- **Category** Suboptimal
- **Source** sPrime.sol

Description Hardcoding mainnet addresses makes it harder testing the code.

Recommendation Consider passing the address as a constructor argument and storing in an immutable variable.

```
88 return 0xb4315e873dBcf96Ffd0acd8EA43f689D8c20fB30;
```

CVF-15 INFO

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

Description This will revert in case the “depositIds” array is empty.

Recommendation Consider returning false in such a case.

```
116 if (depositIds[0] <= activeId && depositIds[depositIds.length - 1]
    ↪ >= activeId) {
```

CVF-16 INFO

- **Category** Suboptimal
- **Source** sPrime.sol

Description The expression “locks[account]” is calculated multiple times.

Recommendation Consider calculating once and reusing.

```
141 for (uint256 i = 0; i < locks[account].length; i++) {
        if (locks[account][i].unlockTime > block.timestamp) {
            fullyVestedBalance += locks[account][i].amount * locks[
                ↪ account][i].lockPeriod / MAX_LOCK_TIME;
```

```
155 for (uint256 i = 0; i < locks[account].length; i++) {
        if (locks[account][i].unlockTime > block.timestamp) {
            lockedBalance += locks[account][i].amount;
```

CVF-17 INFO

- **Category** Overflow/Underflow
- **Source** sPrime.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.

```
143         fullyVestedBalance += locks[account][i].amount * locks[
                ↪ account][i].lockPeriod / MAX_LOCK_TIME;
```

```
216     amountY = amountX * (10 ** IERC20Metadata(address(tokenY)).
            ↪ decimals()) / price;
```

```
232 if(amountY * _REBALANCE_MARGIN / 100 < diff && (amountXToY > 0 ||
        ↪ amountX == 0)) {
```

```
237         amountIn = (diff / 2) * price / (10 ** IERC20Metadata(
                ↪ address(tokenY)).decimals());
```

```
265     amountOut = ILBRouter(getJoeV2RouterAddress()).
            ↪ swapExactTokensForTokens(amountIn, amountOut * (100 -
            ↪ swapSlippage) / 100, path, address(this), block.timestamp)
            ↪ ;
```

```
310     liquidityAmounts[i] = liquidityMinted[i] * share / totalShare;
```

```
544             liquidityMinted[i] = liquidityMinted[i] * amount /
                    ↪ balanceOf(from);
```

CVF-18 INFO

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

Description In case the user doesn’t have any tokens, this function silently returns zero.

Recommendation Consider reverting in such a case.

```
168 function getUserTokenId(address user) public view returns(uint256
        ↪ tokenId){
```

CVF-19 INFO

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

Description In case the “liquidityMinted” array is longer than “depositIds”, the extra elements are silently ignored.

Recommendation Consider reverting in case the arrays have different lengths.

181
```
function _getLiquidityTokenAmounts(uint256[] memory depositIds,
    ↪ uint256[] memory liquidityMinted) internal view returns(
    ↪ uint256 amountX, uint256 amountY) {
```

CVF-20 INFO

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

Description In case the “liquidityConfigs” or “depositIds” array is longer than “deltaIds”, the extra elements are silently ignored.

Recommendation Consider reverting in case the array lengths are different.

277
```
function _encodeDepositConfigs(uint256 centerId) internal view
    ↪ returns (bytes32[] memory liquidityConfigs, uint256[] memory
    ↪ depositIds) {
```

CVF-21 INFO

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

Description In case the “liquidityMinted” array is longer than “depositIds”, the remaining elements are silently ignored.

Recommendation Consider reverting in case the array lengths are different.

297
```
function _withdrawFromLB(uint256[] memory depositIds, uint256[]
    ↪ memory liquidityMinted, uint256 share) internal returns (
    ↪ uint256 balanceX, uint256 balanceY, uint256[] memory
    ↪ liquidityAmounts) {
```

CVF-22 INFO

- **Category** Suboptimal
- **Source** sPrime.sol

Description The value “balanceOf(from)” is calculated multiple times.

Recommendation Consider calculating once and reusing.

```
533 require(balanceOf(from) >= amount + lockedBalance, "Insufficient␣
      ↪ Balance");
```

```
538 if(balanceOf(from) == amount) {
```

```
544         liquidityMinted[i] = liquidityMinted[i] * amount / balanceOf
              ↪ (from);
```

CVF-23 INFO

- **Category** Suboptimal
- **Source** sPrimeUniswap.sol

Recommendation Unchained initializers should be used to prevent double initialization of base contracts.

```
58 __PendingOwnable_init();
   __ReentrancyGuard_init();
60 __ERC20_init(name_, "sPrime");
   __ERC721Holder_init();
```

CVF-24 INFO

- **Category** Procedural
- **Source** sPrimeUniswap.sol

Description Hardcoding mainnet addresses is a bad practice, as it makes it harder testing the code.

Recommendation Consider passing the addresses as constructor arguments and storing in immutable variables.

```
81 return 0x1F98431c8aD98523631AE4a59f267346ea31F984;
```

```
89 return 0xC36442b4a4522E871399CD717aBDD847Ab11FE88;
```

```
97 return 0x68b3465833fb72A70ecDF485E0e4C7bD8665Fc45;
```

CVF-25 INFO

- **Category** Suboptimal
- **Source** sPrimeUniswap.sol

Description The expression “locks[account]” is calculated multiple times.

Recommendation Consider calculating once and reusing.

```
141 for (uint256 i = 0; i < locks[account].length; i++) {
        if (locks[account][i].unlockTime > block.timestamp) {
            fullyVestedBalance += locks[account][i].amount * locks[
               ↪ account][i].lockPeriod / MAX_LOCK_TIME;
```

```
155 for (uint i = 0; i < locks[account].length; i++) {
        if (locks[account][i].unlockTime > block.timestamp) {
            lockedBalance += locks[account][i].amount;
```

CVF-26 INFO

- **Category** Suboptimal
- **Source** sPrimeUniswap.sol

Description The value “balanceOf(from)” is calculated multiple times.

Recommendation Consider calculating once and reusing.

```
441 require(balanceOf(from) >= amount + balance, "Insufficient␣Balance")
      ↪ ;
```

```
446 if(balanceOf(from) == amount) {
```

```
456             liquidity: uint128(liquidity * amount / balanceOf(from))
                  ↪ ,
```

CVF-27 INFO

- **Category** Suboptimal
- **Source** vPrimeController.sol

Description Using mocks in production code looks weird, as usually mocks are used only in tests and experiments.

Recommendation Consider not using mocks or choosing different term.

```
15 SPrimeMock[] public whitelistedSPrimeContracts;
```

CVF-28 INFO

- **Category** Suboptimal
- **Source** vPrimeController.sol

Description In EVM time is measured in seconds. Using other time units makes code more complicated, harder to read and more error prone.

Recommendation Consider measuring all time intervals in seconds.

```
21 uint256 public constant MAX_V_PRIME_VESTING_YEARS = 3;
   uint256 public constant V_PRIME_DETERIORATION_DAYS = 14;
```

8 Minor Issues

CVF-29 INFO

- **Category** Procedural
- **Source** PositionManager.sol

Description Specifying a particular compiler version makes it harder upgrading to newer versions.

Recommendation Consider specifying as “^0.8.0” or as “^0.8.17” if there is something special regarding this particular version. Also relevant for: Prime_L2.sol, Prime.sol, PrimeBridge.sol, sPrime.sol, sPrimeUniswap.sol, vPrimeController.sol, vPrime.sol.

```
3 pragma solidity 0.8.17;
```

CVF-30 INFO

- **Category** Procedural
- **Source** PositionManager.sol

Description We didn’t review these files.

```
8  import "../../interfaces/ISPrime.sol";
   import "../../interfaces/IPositionManager.sol";
10 import "../../lib/joe-v2/LiquidityAmounts.sol";
```

CVF-31 INFO

- **Category** Procedural
- **Source** PositionManager.sol

Description We didn’t review this interface.

```
15 IPositionManager,
```

CVF-32 INFO

- **Category** Procedural
- **Source** PositionManager.sol

Description We didn't review the "Position" structure.

```
19 mapping(uint256 => Position) private _positions;
```

CVF-33 INFO

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

Description The comment uses term "bin id", while variable is named "centerId".

Recommendation Consider using a consistent naming.

```
20 // bin id => Bin Information
   mapping(uint256 => DepositConfig) private _binInfo;
```

```
42     return _binInfo[centerId];
```

CVF-34 INFO

- **Category** Procedural
- **Source** PositionManager.sol

Description We didn't review the "DepositConfig" structure.

```
21 mapping(uint256 => DepositConfig) private _binInfo;
```

CVF-35 INFO

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

Recommendation The type for this variable should be "ISPrime".

```
23 address public sPrime;
```

CVF-36 INFO

- **Category** Procedural
- **Source** PositionManager.sol

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

```
25 constructor() ERC721('SPrime␣Position␣NFT', 'SPRIME-POS') {}
```

CVF-37 INFO

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

Description This function should emit some event.

```
32 function setSPrime(address sPrime_) external onlyOwner {
```

CVF-38 INFO

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

Recommendation The argument type should be “ISPrime”.

```
32 function setSPrime(address sPrime_) external onlyOwner {
```

CVF-39 INFO

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

Recommendation The type for these returned values should be “IERC20”.

```
50 address token0,
   address token1,
```

CVF-40 INFO

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

Recommendation The type for this returned value should be more specific.

```
52 address pairAddr,
```

CVF-41 INFO

- **Category** Procedural
- **Source** PositionManager.sol

Recommendation The brackets around assignment are redundant.

```
82 _mint(params.recipient, (tokenId = _nextId++));
```

CVF-42 INFO

- **Category** Suboptimal
- **Source** PositionManager.sol

Recommendation This check should be replaced with the “onlySPrime” modifier.

```
122 require(sPrime == _msgSender(), "Only␣allowed␣SPrime");
```

```
128 require(sPrime == _msgSender(), "Only␣allowed␣SPrime");
```

CVF-43 INFO

- **Category** Procedural
- **Source** Prime_L2.sol

Description We didn’t review this file.

```
7 import "@layerzerolabs/solidity-examples/contracts/token/oft/v2/
    ↪ BaseOFTV2.sol";
```

CVF-44 INFO

- **Category** Procedural
- **Source** Prime_L2.sol

Description We didn't review the "BaseOFTV2" contract.

```
9 contract Prime_L2 is BaseOFTV2, ERC20 {
```

CVF-45 INFO

- **Category** Bad naming
- **Source** Prime_L2.sol

Description The variable name in confusing.

Recommendation Consider using a more descriptive name and/or documenting.

```
10 uint internal immutable ld2sdRate;
```

CVF-46 INFO

- **Category** Procedural
- **Source** Prime_L2.sol

Description We didn't review the "decimals" function.

```
18 uint8 decimals = decimals();
```

CVF-47 INFO

- **Category** Procedural
- **Source** Prime_L2.sol

Description In ERC20 the decimals property is used by UI to render token amounts in human readable way. Using this property in smart contracts is discouraged.

Recommendation Consider treating all token amounts as integers.

```
18 uint8 decimals = decimals();
```

CVF-48 INFO

- **Category** Bad datatype
- **Source** Prime_L2.sol

Recommendation The return type should be “IERC20”.

```
30 function token() public view virtual override returns (address) {
```

CVF-49 INFO

- **Category** Procedural
- **Source** PrimeBridge.sol

Description We didn’t review this file.

```
5 import "@layerzerolabs/solidity-examples/contracts/token/oft/v2/
    ↪ BaseOFTV2.sol";
```

CVF-50 INFO

- **Category** Procedural
- **Source** PrimeBridge.sol

Description We didn’t review the “BaseOFTV2” contract.

```
8 contract PrimeBridge is BaseOFTV2{
```

CVF-51 INFO

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

Description The variable name in confusing.

Recommendation Consider using a more descriptive name and/or documenting.

```
12 uint internal immutable ld2sdRate;
```

CVF-52 INFO

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

Recommendation The type for this argument should be “IERC20”.

```
18 address _token,
```

CVF-53 INFO

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

Recommendation The type for this argument should be more specific.

```
20 address _lzEndpoint
```

CVF-54 INFO

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

Recommendation The return type should be “IERC20”.

```
39 function token() public view virtual override returns (address) {
```

CVF-55 INFO

- **Category** Suboptimal
- **Source** PrimeBridge.sol

Description This check makes the “_from” argument redundant.

Recommendation Consider removing it.

```
52 require(_from == _msgSender(), "ProxyOFT:␣owner␣is␣not␣send␣caller")
   ↪ ;
```

CVF-56 INFO

- **Category** Procedural
- **Source** PrimeBridge.sol

Description We didn't review the "_removeDust" function.

```
57 (uint amount, uint dust) = _removeDust(_amount);
```

CVF-57 INFO

- **Category** Readability
- **Source** PrimeBridge.sol

Recommendation Should be "else return".

```
80 return _transferFrom(address(this), _toAddress, _amount);
```

CVF-58 INFO

- **Category** Procedural
- **Source** sPrime.sol

Description We didn't review these files.

```
 7 import "../lib/joe-v2/math/SafeCast.sol";
   import "../interfaces/ISPrime.sol";
   import "../interfaces/joe-v2/ILBRouter.sol";
10 import "../interfaces/IPositionManager.sol";
   import "../lib/joe-v2/LiquidityAmounts.sol";
   import "../lib/joe-v2/math/Uint256x256Math.sol";
   import "../lib/joe-v2/math/LiquidityConfigurations.sol";
   import "../abstract/PendingOwnableUpgradeable.sol";
```

```
20 import "@redstone-finance/evm-connector/contracts/core/
     ↪ ProxyConnector.sol";
```

CVF-59 INFO

- **Category** Procedural
- **Source** sPrime.sol

Description We didn't review the "ISPrime", "PendingOwnableUpgradeable", and "ProxyConnector" contracts.

```
23 contract SPrime is ISPrime, ReentrancyGuardUpgradeable,
       ↪ PendingOwnableUpgradeable, ERC20Upgradeable, ProxyConnector {
```

CVF-60 INFO

- **Category** Procedural
- **Source** sPrime.sol

Description We didn't review the "LockDetails" structure.

```
37 mapping(address => LockDetails[]) public locks;
```

CVF-61 INFO

- **Category** Suboptimal
- **Source** sPrime.sol

Recommendation It would be more efficient to store a single array of structs with three fields, rather than three parallel arrays. This would also make the length checks unnecessary.

```
47 int256[] private deltaIds;
   uint256[] private distributionX;
   uint256[] private distributionY;
```

CVF-62 INFO

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

Recommendation The type for the “positionManager_” argument should be “IPositionManager”.

```
62 function initialize(address tokenX_, address tokenY_, string memory
   ↪ name_, uint256[] memory distributionX_, uint256[] memory
   ↪ distributionY_, int256[] memory deltaIds_, address
   ↪ positionManager_, address vPrimeController_) external
   ↪ initializer {
```

CVF-63 INFO

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

Recommendation The return type should be “ILBPair”.

```
91 function getLBPair() public view returns (address) {
```

CVF-64 INFO

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

Recommendation The return type should be “IERC20”.

```
95 function getTokenX() public view returns (address) {
```

```
99 function getTokenY() public view returns (address) {
```

CVF-65 INFO

- **Category** Suboptimal
- **Source** sPrime.sol

Description The expression “locks[account][i]” is calculated several times.

Recommendation Consider calculating once and reusing.

142
```
if (locks[account][i].unlockTime > block.timestamp) {
    fullyVestedBalance += locks[account][i].amount * locks[account][
        ↪ i].lockPeriod / MAX_LOCK_TIME;
```

156
```
if (locks[account][i].unlockTime > block.timestamp) {
    lockedBalance += locks[account][i].amount;
```

CVF-66 INFO

- **Category** Suboptimal
- **Source** sPrime.sol

Recommendation It would be more efficient to pass a single array of structs with two elements, instead of two parallel arrays. This would also make the length check unnecessary.

181
```
function _getLiquidityTokenAmounts(uint256[] memory depositIds,
    ↪ uint256[] memory liquidityMinted) internal view returns(
    ↪ uint256 amountX, uint256 amountY) {
```

CVF-67 INFO

- **Category** Suboptimal
- **Source** sPrime.sol

Description The value “10 ** IERC20Metadata(address(tokenY)).decimals()” is calculated on every invocation.

Recommendation Consider calculating once during contract initialization.

216
```
amountY = amountX * (10 ** IERC20Metadata(address(tokenY)).decimals
    ↪ ()) / price;
```

237
```
    amountIn = (diff / 2) * price / (10 ** IERC20Metadata(address(
        ↪ tokenY)).decimals());
```

CVF-68 INFO

- **Category** Procedural
- **Source** sPrime.sol

Recommendation The brackets around power operator are redundant.

```
216 amountY = amountX * (10 ** IERC20Metadata(address(tokenY)).decimals
    ↪ ()) / price;
```

```
237     amountIn = (diff / 2) * price / (10 ** IERC20Metadata(address(
        ↪ tokenY)).decimals());
```

CVF-69 INFO

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

Recommendation The value “100” should be a named constant.

```
232 if(amountY * _REBALANCE_MARGIN / 100 < diff && (amountXToY > 0 ||
    ↪ amountX == 0)) {
```

```
265     amountOut = ILBRouter(getJoeV2RouterAddress()).
        ↪ swapExactTokensForTokens(amountIn, amountOut * (100 -
        ↪ swapSlippage) / 100, path, address(this), block.timestamp)
        ↪ ;
```

CVF-70 INFO

- **Category** Suboptimal
- **Source** sPrime.sol

Description Using denominator as little as 100 leads to quite coarse precision.

Recommendation Consider using bigger denominator, such as 10000 or 10^18.

```
232 if(amountY * _REBALANCE_MARGIN / 100 < diff && (amountXToY > 0 ||
    ↪ amountX == 0)) {
```

```
265     amountOut = ILBRouter(getJoeV2RouterAddress()).
        ↪ swapExactTokensForTokens(amountIn, amountOut * (100 -
        ↪ swapSlippage) / 100, path, address(this), block.timestamp)
        ↪ ;
```

CVF-71 INFO

- **Category** Suboptimal
- **Source** sPrime.sol

Recommendation It would be more efficient to pass a single array of structs with two elements, rather than two parallel arrays. This would also make the length check unnecessary.

```
277 function _encodeDepositConfigs(uint256 centerId) internal view
    ↪ returns (bytes32[] memory liquidityConfigs, uint256[] memory
    ↪ depositIds) {
```

CVF-72 INFO

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

Description Overflow is possible when converting “centerId” to “int256”.

Recommendation Consider using safe conversion.

```
282 int256 _id = int256(centerId) + deltaIds[i];
```

CVF-73 INFO

- **Category** Suboptimal
- **Source** sPrime.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary.

```
297 function _withdrawFromLB(uint256[] memory depositIds, uint256[]
    ↪ memory liquidityMinted, uint256 share) internal returns (
    ↪ uint256 balanceX, uint256 balanceY, uint256[] memory
    ↪ liquidityAmounts) {
```

CVF-74 INFO

- **Category** Procedural
- **Source** sPrimeUniswap.sol

Description We didn't review these files.

```
6  import "../interfaces/ISPrimeUniswap.sol";
   import "../interfaces/ITokenManager.sol";

12 import "../lib/local/DeploymentConstants.sol";
   import "../abstract/PendingOwnableUpgradeable.sol";

19 import "@redstone-finance/evm-connector/contracts/core/
      ↪ ProxyConnector.sol";
```

CVF-75 INFO

- **Category** Procedural
- **Source** sPrimeUniswap.sol

Description We didn't review some of the base contracts.

```
23 contract sPrimeUniswap is ISPrimeUniswap, ReentrancyGuardUpgradeable
      ↪ , PendingOwnableUpgradeable, ERC20Upgradeable,
      ↪ ERC721HolderUpgradeable, ProxyConnector {
```

CVF-76 INFO

- **Category** Documentation
- **Source** sPrimeUniswap.sol

Description The number format for these values is unclear.

Recommendation Consider documenting.

```
27 uint256 private constant _REBALANCE_MARGIN = 5;
   uint256 private constant _MAX_SLIPPAGE = 10;
   int24 private constant _TICK_RANGE = 10;
```

CVF-77 INFO

- **Category** Procedural
- **Source** sPrimeUniswap.sol

Description We didn't review the "LockDetails" structure.

```
33 mapping(address => LockDetails[]) public locks;
```

CVF-78 INFO

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

Description The semantics of the keys in this mapping is unclear.

Recommendation Consider giving a descriptive name to the key and/or documenting.

```
33 mapping(address => LockDetails[]) public locks;
```

CVF-79 INFO

- **Category** Documentation
- **Source** sPrimeUniswap.sol

Description Strictly speaking this is not a constructor.

Recommendation Consider using term "initializer".

```
49 * @dev Constructor of the contract.
```

CVF-80 INFO

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

Recommendation The type for the token arguments should be "IERC20".

```
57 function initialize(address tokenX_, address tokenY_, string memory
   ↪ name_, uint24 feeTier_, int24[2] memory deltaIds_, address
   ↪ vPrimeController_) external initializer {
```

CVF-81 INFO

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

Recommendation The return type should be “IUniswapV3Factory”.

```
80 function getUniV3FactoryAddress() public view virtual returns (
   ↪ address){
```

CVF-82 INFO

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

Recommendation The return type should be “INonfungiblePositionManager”.

```
88 function getNonfungiblePositionManagerAddress() public view virtual
   ↪ returns (address){
```

CVF-83 INFO

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

Recommendation The return type should be “ISwapRouter”.

```
96 function getSwapRouter() public view virtual returns (address){
```

CVF-84 INFO

- **Category** Suboptimal
- **Source** sPrimeUniswap.sol

Recommendation This could be simplified as: return tickLower <= tick && tock <= tickUpper;

```
112 if (tickLower <= tick && tick <= tickUpper) {
        return true;
    }
    return false;
```

CVF-85 INFO

- **Category** Suboptimal
- **Source** sPrimeUniswap.sol

Description The expression “locks[account][i]” is calculated several times.

Recommendation Consider calculating once and reusing.

```
142 if (locks[account][i].unlockTime > block.timestamp) {
        fullyVestedBalance += locks[account][i].amount * locks[account][
            ↪ i].lockPeriod / MAX_LOCK_TIME;
```

```
156 if (locks[account][i].unlockTime > block.timestamp) {
        lockedBalance += locks[account][i].amount;
```

CVF-86 INFO

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

Recommendation The value “100” should be a named constant.

```
194 if(amountY * _REBALANCE_MARGIN / 100 < diff && (amountXToY > 0 ||
       ↪ amountX == 0)) {
```

```
226                 amountOutMinimum: amountOut * (100 - swapSlippage) /
                       ↪ 100,
```

CVF-87 INFO

- **Category** Suboptimal
- **Source** sPrimeUniswap.sol

Recommendation Double space makes formatting inconsistent.

```
340 tickLower = currenTick -  _TICK_RANGE * deltaIds[0];
    tickUpper = currenTick + _TICK_RANGE * deltaIds[1];
```

CVF-88 INFO

- **Category** Procedural
- **Source** vPrimeController.sol

Description We didn't review these files.

```
5  import "@redstone-finance/evm-connector/contracts/core/
       ↪ RedstoneConsumerNumericBase.sol";

7  import "../abstract/PendingOwnableUpgradeable.sol";
   import "../interfaces/ITokenManager.sol";
   import "../interfaces/IBorrowersRegistry.sol";
10 import "../interfaces/IPool.sol";
   import "./mock/sPrimeMock.sol";
```

CVF-89 INFO

- **Category** Procedural
- **Source** vPrimeController.sol

Description We didn't review the base contracts.

```
14 abstract contract vPrimeController is PendingOwnableUpgradeable,
       ↪ RedstoneConsumerNumericBase {
```

CVF-90 INFO

- **Category** Documentation
- **Source** vPrimeController.sol

Description The number format for these values is unclear.

Recommendation Consider documenting.

```
19 uint256 public constant BORROWER_YEARLY_V_PRIME_RATE = 1;
20 uint256 public constant DEPOSITOR_YEARLY_V_PRIME_RATE = 5;
```

CVF-91 INFO

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

Recommendation The value “1e10” should be a named constant.

```
124 fullyVestedDollarValue += fullyVestedBalance * prices[i] * 1e10 / 10
       ↪  ** whitelistedPools[i].decimals();
    nonVestedDollarValue += nonVestedBalance * prices[i] * 1e10 / 10 **
       ↪ whitelistedPools[i].decimals();
```

```
141 uint256 poolDollarValue = poolBorrowedAmount * prices[i] * 1e10 / 10
       ↪  ** whitelistedPools[i].decimals();
```

```
159 fullyVestedDollarValue += userSPrimeValueInTokenY *
       ↪ sPrimeTokenYPrice * 1e10 * fullyVestedDollarValue /
       ↪ sPrimeBalance / 10 ** sPrimeTokenYDecimals;
160 nonVestedBalance += userSPrimeValueInTokenY * sPrimeTokenYPrice * 1
       ↪ e10 * nonVestedDollarValue / sPrimeBalance / 10 **
       ↪ sPrimeTokenYDecimals;
```

CVF-92 INFO

- **Category** Procedural
- **Source** vPrimeController.sol

Recommendation The values 10**decimals for whitelisted pools and tokens should be precomputed.

```
124 fullyVestedDollarValue += fullyVestedBalance * prices[i] * 1e10 / 10
       ↪  ** whitelistedPools[i].decimals();
    nonVestedDollarValue += nonVestedBalance * prices[i] * 1e10 / 10 **
       ↪ whitelistedPools[i].decimals();
```

```
141 uint256 poolDollarValue = poolBorrowedAmount * prices[i] * 1e10 / 10
       ↪  ** whitelistedPools[i].decimals();
```

```
159 fullyVestedDollarValue += userSPrimeValueInTokenY *
       ↪ sPrimeTokenYPrice * 1e10 * fullyVestedDollarValue /
       ↪ sPrimeBalance / 10 ** sPrimeTokenYDecimals;
160 nonVestedBalance += userSPrimeValueInTokenY * sPrimeTokenYPrice * 1
       ↪ e10 * nonVestedDollarValue / sPrimeBalance / 10 **
       ↪ sPrimeTokenYDecimals;
```

CVF-93 INFO

- **Category** Suboptimal
- **Source** vPrimeController.sol

Description The expression “10 ** whitelistedPools[i].decimals()” is calculated twice.

Recommendation Consider calculating once and reusing.

```
124 fullyVestedDollarValue += fullyVestedBalance * prices[i] * 1e10 / 10
       ↪  ** whitelistedPools[i].decimals();
    nonVestedDollarValue += nonVestedBalance * prices[i] * 1e10 / 10 **
       ↪ whitelistedPools[i].decimals();
```

CVF-94 INFO

- **Category** Suboptimal
- **Source** vPrimeController.sol

Description The expression “10 ** sPrimeTokenYDecimals” is calculated twice.

Recommendation Consider calculating once and reusing.

```
159 fullyVestedDollarValue += userSPrimeValueInTokenY *
       ↪ sPrimeTokenYPrice * 1e10 * fullyVestedDollarValue /
       ↪ sPrimeBalance / 10 ** sPrimeTokenYDecimals;
160 nonVestedBalance += userSPrimeValueInTokenY * sPrimeTokenYPrice * 1
       ↪ e10 * nonVestedDollarValue / sPrimeBalance / 10 **
       ↪ sPrimeTokenYDecimals;
```

CVF-95 INFO

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

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

```
180 ) / 1e18;
```

```
186 uint256 vPrimeMaxCap = vPrimePairsCount *
    ↪ BORROWER_YEARLY_V_PRIME_RATE * MAX_V_PRIME_VESTING_YEARS * 1
    ↪ e18;
```

```
204 uint256 depositVestedPairsCount = Math.min(
    ↪ userDepositFullyVestedDollarValue / 10,
    ↪ userSPrimeDollarValueFullyVested) / 1e18;
uint256 depositNonVestedPairsCount = Math.min(
    ↪ userDepositNonVestedDollarValue / 10,
    ↪ userSPrimeDollarValueNonVested) / 1e18;
```

```
212 uint256 vPrimeMaxCap = (vPrimePairsCountVested +
    ↪ vPrimePairsCountNonVested) * DEPOSITOR_YEARLY_V_PRIME_RATE *
    ↪ MAX_V_PRIME_VESTING_YEARS * 1e18;
uint256 alreadyVestedVPrimeBalance = vPrimePairsCountVested *
    ↪ DEPOSITOR_YEARLY_V_PRIME_RATE * MAX_V_PRIME_VESTING_YEARS * 1
    ↪ e18;
```

CVF-96 INFO

- **Category** Suboptimal
- **Source** vPrimeController.sol

Recommendation The “timestamp” parameters are redundant as every emitted event is bound to a block that already has a timestamp.

```
235 event WhitelistedSPrimeContractsUpdated(SPrimeMock[]
    ↪ newWhitelistedSPrimeContracts, address userAddress, uint256
    ↪ timestamp);
event TokenManagerUpdated(ITokenManager newTokenManager, address
    ↪ userAddress, uint256 timestamp);
```

CVF-97 INFO

- **Category** Procedural
- **Source** vPrime.sol

Description We didn't review this file.

```
7 import "../interfaces/IBorrowersRegistry.sol";
  import "../abstract/PendingOwnableUpgradeable.sol";
```

CVF-98 INFO

- **Category** Suboptimal
- **Source** vPrime.sol

Description This value could be either max or min balance cap.

Recommendation Consider renaming to "balanceLimit" or something like this.

```
17 uint256 maxVPrimeCap;
```

CVF-99 INFO

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

Description This function should emit some event.

```
46 function setVPrimeControllerAddress(address _vPrimeControllerAddress
   ↪ ) external onlyOwner {
```

CVF-100 INFO

- **Category** Suboptimal
- **Source** vPrime.sol

Description The value “_checkpoints[account]” is calculated several times.

Recommendation Consider calculating once and reusing.

93
```
if (_checkpoints[account].length == 0) {
```

96
```
Checkpoint memory cp = _checkpoints[account][_checkpoints[account].
    ↪ length - 1];
```

CVF-101 INFO

- **Category** Suboptimal
- **Source** vPrime.sol

Description This square root calculation is too precise and thus may cost lots of gas.

Recommendation Consider performing only a few iterations to just estimate square root. Another way is to calculate “2^(nbits(length) / 2)”, where “nbits(length)” is the number of bits in “length”.

275
```
uint256 mid = length - Math.sqrt(length);
```

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