Arrakis Finance v2 Follow up II Audit Report

Dec 16, 2022

WATCHPUG

Table of Contents

Summary

This report has been prepared for Arrakis Finance v2 Follow up II Audit Report smart contract, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

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

Overview

Project Summary

Project Name	**Arrakis Finance v2 Follow up II Audit Report**
Codebase	**https://github.com/ArrakisFinance/vault-v2-core**
Commit	**026d9f346394b02b691be2b9259509abe386eab9**
Language	**Solidity**

Audit Summary

Delivery Date	**Dec 16, 2022**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**15**

[WP-M1] `_burnBuffer` mishandled the fee which could result in some users being unable to withdraw

Medium

Issue Description

L244-245 and L250-251 have not taken into account the fees belonging to the shareholders, `fee0` and `fee1` .

As a result, the additional amount to `leftOver` can be higher than `_burnBuffer` for small shareholders.

Thus, they may not be able to withdraw until `rebalance()` or until other users claim the fees first.

https://github.com/ArrakisFinance/vault-v2-core/blob/026d9f346394b02b691be2b9259509abe386eab9/contracts/ArrakisV2.sol#L135-L260

```
135  function burn(
136      BurnLiquidity[] calldata burns_,
137      uint256 burnAmount_,
138      address receiver_
139  ) external nonReentrant returns (uint256 amount0, uint256 amount1) {
     @@ 140,193 @@
194
195      Withdraw memory total;
196      {
197          for (uint256 i; i < burns_.length; i++) {
     @@ 198,224 @@
225          }
226
227          _applyFees(total.fee0, total.fee1);
228      }
229
230      if (amount0 > 0) {
231          token0.safeTransfer(receiver_, amount0);
232      }
233
234        if (amount1 > 0) {
235            token1.safeTransfer(receiver_, amount1);
236        }
237
238        // intentional underflow revert if managerBalance > contract's token balance
239        uint256 leftover0 = token0.balanceOf(address(this)) - managerBalance0;
240        uint256 leftover1 = token1.balanceOf(address(this)) - managerBalance1;
241
242        require(
243            (leftover0 <= underlying.leftOver0) ||
244                ((leftover0 - underlying.leftOver0) <=
245                    FullMath.mulDiv(total.burn0, _burnBuffer, hundredPercent)),
246            "L0"
247        );
248        require(
249            (leftover1 <= underlying.leftOver1) ||
250                ((leftover1 - underlying.leftOver1) <=
251                    FullMath.mulDiv(total.burn1, _burnBuffer, hundredPercent)),
252            "L1"
253        );
254
255        // For monitoring how much user burn LP token for getting their token back.
256        emit LPBurned(msg.sender, total.burn0, total.burn1);
257        emit LogUncollectedFees(underlying.fee0, underlying.fee1);
258        emit LogCollectedFees(total.fee0, total.fee1);
259        emit LogBurn(receiver_, burnAmount_, amount0, amount1);
260    }
```

https://github.com/ArrakisFinance/vault-v2-core/blob/026d9f346394b02b691be2b9259509abe386eab9/contracts/ArrakisV2.sol#L457-L479

```
457        function _withdraw(
458            IUniswapV3Pool pool_,
459            int24 lowerTick_,
460            int24 upperTick_,
461            uint128 liquidity_
462        ) internal returns (Withdraw memory withdraw) {
463            (withdraw.burn0, withdraw.burn1) = pool_.burn(
464                lowerTick_,
465                upperTick_,
466                liquidity_
```

```
        );

        (uint256 collect0, uint256 collect1) = pool_.collect(
            address(this),
            lowerTick_,
            upperTick_,
            type(uint128).max,
            type(uint128).max
        );

        withdraw.fee0 = collect0 - withdraw.burn0;
        withdraw.fee1 = collect1 - withdraw.burn1;
    }
```

PoC

Given:

- The total token0 holdings of the vault is `1000e18` ;
- The total unclaimed token0 fee is: `20e18` ;
- The token0 balance of the vault is: `1e18` , ie, `underlying.leftOver0 = 1e18` ;
- The total token0 holdings of Alice is `10e18` .
- `_burnBuffer` : 20%

1. Alice calls `burn()` to retrieve all her deposit. When `_withdraw()` is called, The vault receives `20e18` in fees while withdrawing `10e18` in liquidity, `total.burn0 = 10e18` ;
2. The current balance of the Vault becomes `20e18 + 10e18 + 1e18 == 31e18` ; After transfered `10e18` to Alice:
   `leftOver0 = 1e18(underlying.leftover0) + 10e18(burn0) + 20e18(fee0) - 10e18(Alice withdrawal) =`
   .

Unfortunately, this means that Alice cannot retrieve her money,

1. `leftover0 <= underlying.leftOver0` can not be satisfied.
2. `(leftover0 - underlying.leftOver0) == 20e18 <= FullMath.mulDiv(total.burn0, _burnBuffer, hundred`
   can not be satisfied, because `total.burn0` doesn't contain fee earned before, but leftover0 contains fee earned before.

Recommendation

Consider changing to:

```
fee0AfterManagerFee = (fee0_ * (hundredPercent - managerFeeBPS)) / hundredPercent;
fee1AfterManagerFee = (fee1_ * (hundredPercent - managerFeeBPS)) / hundredPercent;
require(
    (   fee0AfterManagerFee >= leftover0 ||
        leftover0 - fee0AfterManagerFee <= underlying.leftOver0) ||
        ((leftover0 - fee0AfterManagerFee - underlying.leftOver0) <=
            FullMath.mulDiv(total.burn0, _burnBuffer, hundredPercent)),
    "L0"
);
require(
    (   fee1AfterManagerFee >= leftover1 ||
        leftover1 - fee1AfterManagerFee <= underlying.leftOver1) ||
        ((leftover1 - fee1AfterManagerFee - underlying.leftOver1) <=
            FullMath.mulDiv(total.burn1, _burnBuffer, hundredPercent)),
    "L1"
);
```

Status

✓ Fixed

[WP-M2] `_rebalance()` Lack of slippage control for `burns`

Medium

Issue Description

https://github.com/ArrakisFinance/vault-v2-core/blob/026d9f346394b02b691be2b9259509abe386eab9/contracts/ArrakisV2.sol#L336-L455

```
function _rebalance(Rebalance calldata rebalanceParams_)
    internal
    nonReentrant
{
    // Burns.
    uint256 aggregator0 = 0;
    uint256 aggregator1 = 0;
    IUniswapV3Factory mFactory = factory;
    address mToken0Addr = address(token0);
    address mToken1Addr = address(token1);
    for (uint256 i; i < rebalanceParams_.removes.length; i++) {
        address poolAddr = mFactory.getPool(
            mToken0Addr,
            mToken1Addr,
            rebalanceParams_.removes[i].range.feeTier
        );
        IUniswapV3Pool pool = IUniswapV3Pool(poolAddr);

        Withdraw memory withdraw = _withdraw(
            pool,
            rebalanceParams_.removes[i].range.lowerTick,
            rebalanceParams_.removes[i].range.upperTick,
            rebalanceParams_.removes[i].liquidity
        );

        aggregator0 += withdraw.fee0;
        aggregator1 += withdraw.fee1;
    }

    if (aggregator0 > 0 || aggregator1 > 0) {
        _applyFees(aggregator0, aggregator1);

        emit LogCollectedFees(aggregator0, aggregator1);
    }

    // Swap.
@@ 372,452 @@

    emit LogRebalance(rebalanceParams_);
}
```

The swap (the 2nd step) in `_rebalance` includes slippage control with `expectedMinReturn` .

However, the `Burns` are not controlled.

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

Specifically, a different `amountsOut` from the `burns` .

As a reference, the corresponding Uniswap v3 periphery `burn()` do have proper slippage control:

https://github.com/Uniswap/v3-periphery/blob/6cce88e63e176af1ddb6cc56e029110289622317/contracts/interfaces/INonfungiblePositionManager.sol#L139-L165

```
struct DecreaseLiquidityParams {
    uint256 tokenId;
    uint128 liquidity;
    uint256 amount0Min;
    uint256 amount1Min;
    uint256 deadline;
}

/// @notice Decreases the amount of liquidity in a position and accounts it to the position
/// @param params tokenId The ID of the token for which liquidity is being decreased,
/// amount The amount by which liquidity will be decreased,
/// amount0Min The minimum amount of token0 that should be accounted for the burned liquidity,
/// amount1Min The minimum amount of token1 that should be accounted for the burned liquidity,
/// deadline The time by which the transaction must be included to effect the change
/// @return amount0 The amount of token0 accounted to the position's tokens owed
/// @return amount1 The amount of token1 accounted to the position's tokens owed
function decreaseLiquidity(DecreaseLiquidityParams calldata params)
    external
    payable
    returns (uint256 amount0, uint256 amount1);

struct CollectParams {
    uint256 tokenId;
    address recipient;
    uint128 amount0Max;
    uint128 amount1Max;
}
```

Recommendation

Consider adding proper slippage control to the `burns` , similar to Uniswap v3's `NonfungiblePositionManager.sol` .

Status

✓ Fixed

[WP-I3] Incorrect/Misleading comment

Informational

Issue Description

https://github.com/ArrakisFinance/vault-v2-core/blob/026d9f346394b02b691be2b9259509abe386eab9/contracts/ArrakisV2.sol#L60-L126

```
    function mint(uint256 mintAmount_, address receiver_)
        external
        nonReentrant
        returns (uint256 amount0, uint256 amount1)
    {
        require(mintAmount_ > 0, "MA");
        require(
            restrictedMint == address(0) || msg.sender == restrictedMint,
            "R"
        );
        address me = address(this);
        uint256 ts = totalSupply();
        bool isTotalSupplyGtZero = ts > 0;
        (
            uint256 current0,
            uint256 current1,
            uint256 fee0,
            uint256 fee1
        ) = isTotalSupplyGtZero
                ? UnderlyingHelper.totalUnderlyingWithFees(
                    UnderlyingPayload({
                        ranges: ranges,
                        factory: factory,
                        token0: address(token0),
                        token1: address(token1),
                        self: me
                    })
                )
                : (init0, init1, 0, 0);
        uint256 denominator = isTotalSupplyGtZero ? ts : 1 ether;

        /// @dev current0 and current1 include fees and left over (but not manager balances)
```

```
92          amount0 = FullMath.mulDivRoundingUp(mintAmount_, current0, denominator);
93          amount1 = FullMath.mulDivRoundingUp(mintAmount_, current1, denominator);
94
95          // #region check amount0 is a multiple of current0.
96
97          if (!isTotalSupplyGtZero) {
98              uint256 amount0Mint = current0 != 0
99                  ? FullMath.mulDiv(amount0, denominator, current0)
100                 : type(uint256).max;
101             uint256 amount1Mint = current1 != 0
102                 ? FullMath.mulDiv(amount1, denominator, current1)
103                 : type(uint256).max;
104
105             require(
106                 (amount0Mint < amount1Mint ? amount0Mint : amount1Mint) ==
107                     mintAmount_,
108                 "A0&A1"
109             );
110         }
111
112         // #endregion check amount0 is a multiple of current0.
113
114         _mint(receiver_, mintAmount_);
115
116         // transfer amounts owed to contract
117         if (amount0 > 0) {
118             token0.safeTransferFrom(msg.sender, me, amount0);
119         }
120         if (amount1 > 0) {
121             token1.safeTransferFrom(msg.sender, me, amount1);
122         }
123
124         emit LogUncollectedFees(fee0, fee1);
125         emit LogMint(receiver_, mintAmount_, amount0, amount1);
126     }
```

The comment at L95/L112 is incorrect and misleading:

> #region check amount0 is a multiple of current0.

However, it does not required that `amount0` is a multiple of `current0`, in fact `amount0` is often not a multiple of `current0` and can still pass the check in L105.

Recommendation

Consider changing the comment to something like:

> #region check `amount0 * denominator - mintAmount * current0 < current0`

Status

✓ Fixed

[WP-M4] `VaultV2.burn()` may revert as the `BurnLiquidity[]` burns returned by `ArrakisV2Resolver.standardBurnParams()` can be slightly smaller than expected

Medium

Issue Description

If the total outAmounts from the burns (`BurnLiquidity[]`) returned by `ArrakisV2Resolver.standardBurnParams(amountToBurn_, vaultV2_)` is not enough, it may cause `vaultV2.burn()` to revert.

When all the `token0` and `token1` of the vault are in the liquidity of UniswapV3Pool (i.e., the vault contract itself has no token0 and token1 in its contract balance, and there is no pending fee in UniswapV3Pool), due to the accumulated precision loss of ArrakisV2Resolver at line 227, the total number of `token0` and `token1` taken out from UniswapV3Pool may not be enough, resulting in a revert at lines 231 and 235.

Furthermore, if ArrakisV2Resolver L227 rounds down to 0, `vaultV2.burn()` will revert at ArrakisV2 L198 as well.

https://github.com/ArrakisFinance/vault-v2-core/blob/026d9f346394b02b691be2b9259509abe386eab9/contracts/ArrakisV2Resolver.sol#L145-L238

```
145  function standardBurnParams(uint256 amountToBurn_, IArrakisV2 vaultV2_)
146      external
147      view
148      returns (BurnLiquidity[] memory burns)
149  {
     @@ 150,220 @@
221      burns = new BurnLiquidity[](len);
222      uint256 idx;
223      for (uint256 j; j < ranges.length; j++) {
224          if (liquidities[j] > 0) {
225              burns[idx] = BurnLiquidity({
226                  liquidity: SafeCast.toUint128(
227                      FullMath.mulDiv(
228                          liquidities[j],
229                         amountToBurn_,
230                         totalSupply
231                     )
232                 ),
233                 range: ranges[j]
234             });
235             ++idx;
236         }
237     }
238 }
```

https://github.com/ArrakisFinance/vault-v2-core/blob/026d9f346394b02b691be2b9259509abe386eab9/contracts/ArrakisV2.sol#L135-L260

```
135 function burn(
136     BurnLiquidity[] calldata burns_,
137     uint256 burnAmount_,
138     address receiver_
139 ) external nonReentrant returns (uint256 amount0, uint256 amount1) {
    @@ 140,192 @@
193     _burn(msg.sender, burnAmount_);
194
195     Withdraw memory total;
196     {
197         for (uint256 i; i < burns_.length; i++) {
198             require(burns_[i].liquidity != 0, "LZ");
199             {
200                 (bool exist, ) = Position.rangeExist(
201                     ranges,
202                     burns_[i].range
203                 );
204                 require(exist, "RRNE");
205             }
206
207             Withdraw memory withdraw = _withdraw(
208                 IUniswapV3Pool(
209                     factory.getPool(
210                         address(token0),
211                         address(token1),
212                         burns_[i].range.feeTier
```

```
                    )
                ),
                burns_[i].range.lowerTick,
                burns_[i].range.upperTick,
                burns_[i].liquidity
            );

            total.fee0 += withdraw.fee0;
            total.fee1 += withdraw.fee1;

            total.burn0 += withdraw.burn0;
            total.burn1 += withdraw.burn1;
        }

        _applyFees(total.fee0, total.fee1);
    }

    if (amount0 > 0) {
        token0.safeTransfer(receiver_, amount0);
    }

    if (amount1 > 0) {
        token1.safeTransfer(receiver_, amount1);
    }

@@ 238,259 @@
}
```

Recommendation

Consider changing ArrakisV2Resolver L227 to `mulDivRoundingUp()` :

```
224 if (liquidities[j] > 0) {
225     burns[idx] = BurnLiquidity({
226         liquidity: SafeCast.toUint128(
227             FullMath.mulDivRoundingUp(
228                 liquidities[j],
229                 amountToBurn_,
230                 totalSupply
231             )
```

```
            ),
            range: ranges[j]
        });
        ++idx;
    }
```

Status

✓ Fixed

[WP-I5] Consider adding `nonReentrant` modifier to `withdrawManagerBalance()`

Informational

Issue Description

https://github.com/ArrakisFinance/vault-v2-core/blob/d958ffd0e9ed7890b55d8ade4fdc26eae9640ab3/contracts/ArrakisV2.sol#L317-L333

```
function withdrawManagerBalance() external {
    uint256 amount0 = managerBalance0;
    uint256 amount1 = managerBalance1;

    managerBalance0 = 0;
    managerBalance1 = 0;

    if (amount0 > 0) {
        token0.safeTransfer(manager, amount0);
    }

    if (amount1 > 0) {
        token1.safeTransfer(manager, amount1);
    }

    emit LogWithdrawManagerBalance(amount0, amount1);
}
```

The manager can reenter `burn()` if one of the tokens is a hookable token (ERC777) in `withdrawManagerBalance()`, and using the abnormal `pricePerShare` to withdraw more `token0` or token1 than expected.

Status

✓ Fixed

[WP-I6] Inconsistent `address(0)` check in `upgradeVaults()` and `upgradeVaultsAndCall()`

Informational

Issue Description

The `upgradeVaults()` function has been updated with an `implementation != address(0)` check, but the `upgradeVaultsAndCall()` function has not been updated.

By the way, consider using CAS to prevent the `arrakisV2Beacon.implementation()` from changing between the time the `upgradeVaults` transaction is sent and the time the transaction is minted.

https://github.com/ArrakisFinance/vault-v2-core/blob/026d9f346394b02b691be2b9259509abe386eab9/contracts/abstract/ArrakisV2FactoryStorage.sol#L49-L55

```
    function upgradeVaults(address[] memory vaults_) external onlyOwner {
        address implementation = arrakisV2Beacon.implementation();
        require(implementation != address(0), "implementation is address zero");
        for (uint256 i = 0; i < vaults_.length; i++) {
            ITransparentUpgradeableProxy(vaults_[i]).upgradeTo(implementation);
        }
    }
```

https://github.com/ArrakisFinance/vault-v2-core/blob/026d9f346394b02b691be2b9259509abe386eab9/contracts/abstract/ArrakisV2FactoryStorage.sol#L62-L73

```
    function upgradeVaultsAndCall(
        address[] memory vaults_,
        bytes[] calldata datas_
    ) external onlyOwner {
        require(vaults_.length == datas_.length, "mismatching array length");
        for (uint256 i = 0; i < vaults_.length; i++) {
            ITransparentUpgradeableProxy(vaults_[i]).upgradeToAndCall(
                arrakisV2Beacon.implementation(),
                datas_[i]
            );
        }
    }
```

Recommendation

Change to:

```
function upgradeVaults(address[] memory vaults_, address implementation_) external
onlyOwner {
    address implementation = arrakisV2Beacon.implementation();
    require(implementation == implementation_, "implementation mismatch");
    for (uint256 i = 0; i < vaults_.length; i++) {
        ITransparentUpgradeableProxy(vaults_[i]).upgradeTo(implementation);
    }
}
```

```
function upgradeVaultsAndCall(
    address[] memory vaults_,
    bytes[] calldata datas_,
    address implementation_
) external onlyOwner {
    address implementation = arrakisV2Beacon.implementation();
    require(implementation == implementation_, "implementation mismatch");
    require(vaults_.length == datas_.length, "mismatching array length");
    for (uint256 i = 0; i < vaults_.length; i++) {
        ITransparentUpgradeableProxy(vaults_[i]).upgradeToAndCall(
            arrakisV2Beacon.implementation(),
            datas_[i]
        );
    }
}
```

Status

✓ Fixed

[WP-N7] Unused imports

Issue Description

ArrakisFinance/vault-v2-core

contracts/abstract/ArrakisV2FactoryStorage.sol

- ”Initializable” is unused

contracts/ArrakisV2.sol

- ”ERC20” is unused
- ”SafeCast” is unused

contracts/ArrakisV2Factory.sol

- ”ERC1967Upgrade” is unused

contracts/ArrakisV2Resolver.sol

- ”ERC20” is unused
- ”SwapPayload” is unused

PALM/PALMManager.sol

- ”VaultInfo” is unused

ArrakisFinance/v2-palm

contracts/abstracts/PALMManagerStorage.sol

- ”Ownable” is unused

contracts/PALMManager.sol

- ”VaultInfo” is unused

contracts/structs/SPALMTerms.sol

- ”BurnLiquidity” is unused

Status

✓ Fixed

[WP-D8] Improve the NatSpec comments for `totalUnderlyingWithFees()` about the returns

Issue Description

- `amount0` , `amount1` : the total amount of underlying tokens (`token0` and `token1`) in the ArrakisV2 vault corresponding to the share holders
- `fee0` , `fee1` : the amount of fees in the underlying UniswapV3Pool, where the `managerFeeBPS_` portion goes to the manager and the rest goes to the share holders.

Because, the return variable names are not specific and the scope of the `amount` and `fee` are not consistent, which can lead to misunderstandings or omissions of what is being returned.

Status

ⓘ Acknowledged

[WP-G9] `Position#rangeExist()` can be optimized

Gas

Issue Description

The MSTORE at L45 is only needed when `ok == true` :

https://github.com/ArrakisFinance/vault-v2-core/blob/d958ffd0e9ed7890b55d8ade4fdc26eae9640ab3/contracts/libraries/Position.sol#L35-L48

```
35      function rangeExist(Range[] memory currentRanges_, Range memory range_)
36          public
37          pure
38          returns (bool ok, uint256 index)
39      {
40          for (uint256 i; i < currentRanges_.length; i++) {
41              ok =
42                  range_.lowerTick == currentRanges_[i].lowerTick &&
43                  range_.upperTick == currentRanges_[i].upperTick &&
44                  range_.feeTier == currentRanges_[i].feeTier;
45              index = i;
46              if (ok) break;
47          }
48      }
```

Recommendation

```
35      function rangeExist(Range[] memory currentRanges_, Range memory range_)
36          public
37          pure
38          returns (bool ok, uint256 index)
39      {
40          for (uint256 i; i < currentRanges_.length; i++) {
41              ok =
42                  range_.lowerTick == currentRanges_[i].lowerTick &&
43                  range_.upperTick == currentRanges_[i].upperTick &&
44                  range_.feeTier == currentRanges_[i].feeTier;
45              if (ok) {
```

```
                index = i;
                break;
            }
        }
    }
```

Status

✓ Fixed

[WP-G10] `safeApprove(0)` to the newly created vault is unnecessary

Gas

Issue Description

https://github.com/ArrakisFinance/v2-palm/blob/46546698096f0f4cd86381cd06be28d43a50a2ea/contracts/PALMTerms.sol#L56-L128

```
function openTerm(SetupPayload calldata setup_)
    external
    payable
    override
    collectLeftOver(setup_.token0, setup_.token1)
    returns (address vault)
{
@@ 63,107 @@
    setup_.token0.safeTransferFrom(
        msg.sender,
        address(this),
        setup_.amount0
    );
    setup_.token1.safeTransferFrom(
        msg.sender,
        address(this),
        setup_.amount1
    );

    setup_.token0.safeApprove(vault, 0);
    setup_.token1.safeApprove(vault, 0);

    setup_.token0.safeApprove(vault, setup_.amount0);
    setup_.token1.safeApprove(vault, setup_.amount1);

    vaultV2.mint(mintAmount, address(this));

    emit SetupVault(setup_.owner, vault);
}
```

The **vault** is a newly created address, therefore the existing allowance must be **zero**.

The `safeApprove` at L119-120 is redundant.

Recommendation

Remove L119-120.

Status

✓ **Fixed**

[WP-G11] `PALMManagerStorage._withdrawVaultBalance()` can be optimized

Gas

Issue Description

https://github.com/ArrakisFinance/v2-palm//blob/46546698096f0f4cd86381cd06be28d43a50a2ea/contracts/abstracts/PALMManagerStorage.sol#L480-L493

```
    function _withdrawVaultBalance(
        address vault_,
        uint256 amount_,
        address payable to_
    ) internal {
        require(
            vaults[vault_].balance >= amount_,
            "PALMManager: amount exceeds available balance"
        );
        vaults[vault_].balance -= amount_;
        to_.sendValue(amount_);

        emit WithdrawVaultBalance(vault_, amount_, to_, vaults[vault_].balance);
    }
```

Recommendation

Consider changing to:

```
    function _withdrawVaultBalance(
        address vault_,
        uint256 amount_,
        address payable to_
    ) internal {
        uint256 oldBalance = vaults[vault_].balance;
        require(
            oldBalance >= amount_,
            "PALMManager: amount exceeds available balance"
        );
        uint256 newBalance = oldBalance - amount_;
        vaults[vault_].balance = newBalance;
        to_.sendValue(amount_);

        emit WithdrawVaultBalance(vault_, amount_, to_, newBalance);
    }
```

Status

✓ Fixed

[WP-G12] Removing `rangesToRemove` in `rebalance()` can be optimized

Gas

Issue Description

https://github.com/ArrakisFinance/vault-v2-core/blob/026d9f346394b02b691be2b9259509abe386eab9/contracts/ArrakisV2.sol#L270-L313

```
270  function rebalance(
271      Range[] calldata rangesToAdd_,
272      Rebalance calldata rebalanceParams_,
273      Range[] calldata rangesToRemove_
274  ) external onlyManager {
     @@ 275,292 @@
293      for (uint256 i; i < rangesToRemove_.length; i++) {
294          (bool exist, uint256 index) = Position.rangeExist(
295              ranges,
296              rangesToRemove_[i]
297          );
298          require(exist, "RRNE");
299
300          Position.requireNotActiveRange(
301              factory,
302              address(this),
303              address(token0),
304              address(token1),
305              rangesToRemove_[i]
306          );
307
308          for (uint256 j = index; j < ranges.length - 1; j++) {
309              ranges[j] = ranges[j + 1];
310          }
311          ranges.pop();
312      }
313  }
```

The current implementation to remove the `rangesToRemove_` from `ranges` is very gas-expensive.

And based on the context, it seems unnecessary to maintain the order in `ranges` .

Recommendation

```
for (uint256 i; i < rangesToRemove_.length; i++) {
    (bool exist, uint256 index) = Position.rangeExist(
        ranges,
        rangesToRemove_[i]
    );
    require(exist, "RRNE");

    Position.requireNotActiveRange(
        factory,
        address(this),
        address(token0),
        address(token1),
        rangesToRemove_[i]
    );

    ranges[index] = ranges[ranges.length - 1];
    ranges.pop();
}
```

Status

✓ Fixed

[WP-G13] `ArrakisV2Resolver.sol#standardBurnParams()` Cache external call results can save gas

Gas

Issue Description

Every call to an external contract costs a decent amount of gas. For optimization of gas usage, external call results should be cached if they are being used for more than one time.

https://github.com/ArrakisFinance/vault-v2-core/blob/026d9f346394b02b691be2b9259509abe386eab9/contracts/ArrakisV2Resolver.sol#L145-L238

```
function standardBurnParams(uint256 amountToBurn_, IArrakisV2 vaultV2_)
    external
    view
    returns (BurnLiquidity[] memory burns)
{
    uint256 totalSupply = vaultV2_.totalSupply();
    require(totalSupply > 0, "total supply");

    Range[] memory ranges = vaultV2_.getRanges();

    {
        UnderlyingOutput memory underlying;
        (
            underlying.amount0,
            underlying.amount1,
            underlying.fee0,
            underlying.fee1
        ) = UnderlyingHelper.totalUnderlyingWithFees(
            UnderlyingPayload({
                ranges: ranges,
                factory: factory,
                token0: address(vaultV2_.token0()),
                token1: address(vaultV2_.token1()),
                self: address(vaultV2_)
            })
        );
        underlying.leftOver0 =
            vaultV2_.token0().balanceOf(address(vaultV2_)) -
```

```
                vaultV2_.managerBalance0();
            underlying.leftOver1 =
                vaultV2_.token1().balanceOf(address(vaultV2_)) -
                vaultV2_.managerBalance1();

            {
                uint256 amount0 = FullMath.mulDiv(
                    underlying.amount0,
                    amountToBurn_,
                    totalSupply
                );
                uint256 amount1 = FullMath.mulDiv(
                    underlying.amount1,
                    amountToBurn_,
                    totalSupply
                );

                if (
                    amount0 <= underlying.leftOver0 &&
                    amount1 <= underlying.leftOver1
                ) return burns;
            }
        }
        // #endregion get amount to burn.

        uint128[] memory liquidities = new uint128[](ranges.length);
        uint256 len;
        for (uint256 i; i < ranges.length; i++) {
            uint128 liquidity;
            {
                (liquidity, , , , ) = IUniswapV3Pool(
                    factory.getPool(
                        address(vaultV2_.token0()),
                        address(vaultV2_.token1()),
                        ranges[i].feeTier
                    )
                ).positions(
                        PositionHelper.getPositionId(
                            address(vaultV2_),
                            ranges[i].lowerTick,
                            ranges[i].upperTick
                        )
                    );
```

```solidity
        }
        liquidities[i] = liquidity;

        if (liquidity != 0) ++len;
    }
    burns = new BurnLiquidity[](len);
    uint256 idx;
    for (uint256 j; j < ranges.length; j++) {
        if (liquidities[j] > 0) {
            burns[idx] = BurnLiquidity({
                liquidity: SafeCast.toUint128(
                    FullMath.mulDiv(
                        liquidities[j],
                        amountToBurn_,
                        totalSupply
                    )
                ),
                range: ranges[j]
            });
            ++idx;
        }
    }
}
```

`vaultV2_.token0()` , `vaultV2_.token1()` can be cached in storage to save the external call.

While the `ArrakisV2Resolver.sol#standardBurnParams()` is an external view function, there are on-chain invocations as well, eg:

https://github.com/ArrakisFinance/vault-v2-agreement/blob/46546698096f0f4cd86381cd06be28d43a50a2ea/contracts/functions/FPALMTerms.sol#L27-L30

```solidity
27  BurnLiquidity[] memory burnPayload = resolver.standardBurnParams(
28      balance,
29      vault_
30  );
```

Status

ⓘ **Acknowledged**

[WP-G14] Avoiding unnecessary storage read can save gas

Gas

Issue Description

https:
//github.com/ArrakisFinance/v2-palm//blob/0c4572718328e9aa0061c959f658e4d1c4ba4568/
contracts/abstracts/PALMTermsStorage.sol#L241-L277

```
function setVaultData(address vault_, bytes calldata data_)
    external
    override
    requireAddressNotZero(vault_)
    requireDelegateWhenDelegateExistsOtherwiseRequireIsOwner(vault_)
{
    IPALMManager(manager).setVaultData(vault_, data_);

    emit LogSetVaultData(
        delegateByVaults[vault_] != address(0)
            ? delegateByVaults[vault_]
            : msg.sender,
        vault_,
        data_
    );
}

/// @notice set Arrakis V2 vault strategy
/// @param vault_ Arrakis V2 vault
/// @param strat_ strategy to apply during market making
/// @dev only callable by delegate of the vault by default or otherwise owner
function setVaultStratByName(address vault_, string calldata strat_)
    external
    override
    requireAddressNotZero(vault_)
    requireDelegateWhenDelegateExistsOtherwiseRequireIsOwner(vault_)
{
    IPALMManager(manager).setVaultStratByName(vault_, strat_);

    emit LogSetVaultStratByName(
        delegateByVaults[vault_] != address(0)
            ? delegateByVaults[vault_]
```

```
                : msg.sender,
            vault_,
            strat_
        );
}
```

Recommendation

```
function setVaultData(address vault_, bytes calldata data_)
    external
    override
    requireAddressNotZero(vault_)
    requireDelegateWhenDelegateExistsOtherwiseRequireIsOwner(vault_)
{
    IPALMManager(manager).setVaultData(vault_, data_);
    address delegate = delegateByVaults[vault_];
    emit LogSetVaultData(
        delegate != address(0)
            ? delegate
            : msg.sender,
        vault_,
        data_
    );
}
```

Status

✓ Fixed

[WP-G15] Storage variables can be cached in memory to save gas

Gas

Issue Description

https://github.com/ArrakisFinance/vault-v2-core/blob/026d9f346394b02b691be2b9259509abe386eab9/contracts/ArrakisV2.sol#L481-L484

```
    function _applyFees(uint256 fee0_, uint256 fee1_) internal {
        managerBalance0 += (fee0_ * managerFeeBPS) / hundredPercent;
        managerBalance1 += (fee1_ * managerFeeBPS) / hundredPercent;
    }
```

`managerFeeBPS` is a storage variable that is being read multiple times, they can be cached to save ~100 gas for each extra `SLOAD` s.

Recommendation

Change to:

```
    function _applyFees(uint256 fee0_, uint256 fee1_) internal {
        uint16 mManagerFeeBPS = managerFeeBPS;
        managerBalance0 += (fee0_ * mManagerFeeBPS) / hundredPercent;
        managerBalance1 += (fee1_ * mManagerFeeBPS) / hundredPercent;
    }
```

Status

✓ Fixed

Appendix

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by WatchPug; however, WatchPug does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Disclaimer