Arrakis Finance v2 Vault Core Audit Report

Jan 31, 2023

WATCHPUG

Table of Contents

Summary

This report has been prepared for Arrakis Finance v2 Vault Core 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 Vault Core Audit Report**
Codebase	**https://github.com/ArrakisFinance/vault-v2-core**
Commit	**903e55c0ac61e37f8ce92b6a46d7fe1ffc5e4fc9**
Language	**Solidity**

Audit Summary

Delivery Date	**Jan 31, 2023**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**12**

[WP-H1] `ArrakisV2#rebalance()` Dangerous arbitrary external call can be used by the manager to steal funds from the users who have approved tokens to the vault contract

High

Issue Description

https://github.com/ArrakisFinance/vault-v2-core/blob/27004a99dc61dc19502538434841ae72433200be/contracts/ArrakisV2.sol#L382-L385

```
(bool success, ) = rebalanceParams_.swap.router.call(
    rebalanceParams_.swap.payload
);
require(success, "swap");
```

For the users who approved the vault contract to `mint()` directly without using the router, `manager` can rebalance with `token0` or `token1` 's address as `rebalanceParams_.swap.router` and `transferFrom(victim, attacker, amount)` as payload to steal funds from the victim.

Besides, the manager can also use `transfer(attacker, amount)` as the payload and sweep the amounts in the balance to rug all users.

Recommendation

Consider blacklist token0 and token1 as `_swapData.swapRouter` .

Furthermore, consider requiring the `rebalanceParams_.swap.router` to be an address whitelisted on the factory.

Status

✓ Fixed

[WP-M2] Swap requires all the amountIn to be spent precisely making the transaction prone to revert

Medium

Issue Description

https://github.com/ArrakisFinance/vault-v2-core/blob/27004a99dc61dc19502538434841ae72433200be/contracts/ArrakisV2.sol#L436-L444

```
436  require(
437      (balance0After >=
438          balance0Before +
439              rebalanceParams_.swap.expectedMinReturn) &&
440          (balance1After ==
441              balance1Before -
442                  rebalanceParams_.swap.amountIn),
443      "SF"
444  );
```

`ArrakisV2Router._swap()` :

https://github.com/ArrakisFinance/vault-v2-periphery/blob/29c2d050d232be109ef0ac49698a0bafbb283f14/contracts/ArrakisV2Router.sol#L306-L363

```
306      function _swap(AddAndSwapData memory _swapData)
307          internal
308          returns (uint256 amount0Diff, uint256 amount1Diff)
309      {
310          IERC20 token0 = _swapData.vault.token0();
311          IERC20 token1 = _swapData.vault.token1();
312          uint256 balance0Before = token0.balanceOf(address(this));
313          uint256 balance1Before = token1.balanceOf(address(this));
314

@@ 315,335 @@

336
337          uint256 balance0 = token0.balanceOf(address(this));
338          uint256 balance1 = token1.balanceOf(address(this));
```

```
        if (_swapData.zeroForOne) {
            amount0Diff = balance0Before - balance0;
            amount1Diff = balance1 - balance1Before;
            require(
                (amount0Diff == _swapData.amountInSwap) &&
                    (amount1Diff >= _swapData.amountOutSwap),
                "Token0 swap failed!"
            );
        } else {
            amount0Diff = balance0 - balance0Before;
            amount1Diff = balance1Before - balance1;
            require(
                (amount0Diff >= _swapData.amountOutSwap) &&
                    (amount1Diff == _swapData.amountInSwap),
                "Token1 swap failed!"
            );
        }

@@ 357,362 @@
    }
```

Certain swap aggregators (routers) like 1inch's `AggregationRouterV4` , will not spend all the `amountIn` , the unspent amount will be returned:

```
{
    bytes memory callData = abi.encodePacked(caller.callBytes.selector,
bytes12(0), msg.sender, data);
    // solhint-disable-next-line avoid-low-level-calls
    (bool success, bytes memory result) = address(caller).call{value:
msg.value}(callData);
    if (!success) {
        revert(RevertReasonParser.parse(result, "callBytes failed: "));
    }
}

spentAmount = desc.amount;
returnAmount = dstToken.uniBalanceOf(address(this));

if (flags & _PARTIAL_FILL != 0) {
    uint256 unspentAmount = srcToken.uniBalanceOf(address(this));
```

```
        if (unspentAmount > 0) {
            spentAmount = spentAmount.sub(unspentAmount);
            srcToken.uniTransfer(msg.sender, unspentAmount);
        }
        require(returnAmount.mul(desc.amount) >=
    desc.minReturnAmount.mul(spentAmount), "Return amount is not enough");
    } else {
        require(returnAmount >= desc.minReturnAmount, "Return amount is not enough");
    }

    address payable dstReceiver = (desc.dstReceiver == address(0)) ? msg.sender :
    desc.dstReceiver;
    dstToken.uniTransfer(dstReceiver, returnAmount);
```

Requiring the balance after strictly equals the amountIn means that any unspent amount will revert the whole transaction.

Recommendation

Change to checks for `balanceAfter > balanceBefore - amountIn` .

Status

✓ Fixed

[WP-H3] `ArrakisV2#rebalance()` may spend part of the managerBalance + arrakisBalance in the balance and cause `burn()` to revert

High

Issue Description

https://github.com/ArrakisFinance/vault-v2-core/blob/1338a6cfdb1b5f22666209e3763aa8a096b905b7/contracts/ArrakisV2.sol#L111-L141

```
111 function burn(
112     BurnLiquidity[] calldata burns_,
113     uint256 burnAmount_,
114     address receiver_
115 ) external nonReentrant returns (uint256 amount0, uint256 amount1) {
116     uint256 totalSupply = totalSupply();
117     require(totalSupply > 0, "TS");
118 
119     UnderlyingOutput memory underlying;
120     (
121         underlying.amount0,
122         underlying.amount1,
123         underlying.fee0,
124         underlying.fee1
125     ) = UnderlyingHelper.totalUnderlyingWithFees(
126         UnderlyingPayload({
127             ranges: ranges,
128             factory: factory,
129             token0: address(token0),
130             token1: address(token1),
131             self: address(this)
132         })
133     );
134     underlying.leftOver0 =
135         token0.balanceOf(address(this)) -
136         (managerBalance0 + arrakisBalance0);
137     underlying.leftOver1 =
138         token1.balanceOf(address(this)) -
139         (managerBalance1 + arrakisBalance1);
```

The tokens in the balance MUST be greater than or equal to `managerBalance + arrakisBalance` for both token0 and token1 to ensure `burn()` can work properly.

However, there is no such restriction in `rebalance()` to prevent the manager from consuming more balance.

Recommendation

Consider adding `token.balanceOf(address(this)) >= managerBalance + arrakisBalance` in the end of `rebalance()` .

Status

✓ Fixed

[WP-H4] `amount0` , `amount1` returned from `Underlying.totalUnderlyingWithFees()` is larger than the actual amounts as the admin and protocol fees are not deducted from the uncollected fees

High

Issue Description

A recent update: 6860862472ab060f370e9f6b60d4e58c79d5ef93 has rendered this issue invalid. We leave the issue as it is, especially the `Recommendation` section, to provide a reference of an alternative resolution.

In `Underlying.totalUnderlyingWithFees()` , all the `f0` , `f1` from the underlying pool are added to `amount0` , `amount1` directly.

However, not all the `f0` , `f1` belongs to the share holders. There is a portion of the fees belongs to the `manager` and the Arrakis protocol as managerFee and arrakisFee.

As a result, the `amount0` and `amount1` returned from `Underlying.totalUnderlyingWithFees()` is larger than the actual amounts.

https://github.com/ArrakisFinance/vault-v2-core/blob/27004a99dc61dc19502538434841ae72433200be/contracts/libraries/Underlying.sol#L24-L73

```
function totalUnderlyingWithFees(
    UnderlyingPayload memory underlyingPayload_
)
    public
    view
    returns (
        uint256 amount0,
        uint256 amount1,
        uint256 fee0,
        uint256 fee1
    )
{
    for (uint256 i = 0; i < underlyingPayload_.ranges.length; i++) {
        {
            IUniswapV3Pool pool = IUniswapV3Pool(
                underlyingPayload_.factory.getPool(
                    underlyingPayload_.token0,
                    underlyingPayload_.token1,
                    underlyingPayload_.ranges[i].feeTier
                )
            );
            (uint256 a0, uint256 a1, uint256 f0, uint256 f1) = underlying(
                RangeData({
                    self: underlyingPayload_.self,
                    range: underlyingPayload_.ranges[i],
                    pool: pool
                })
            );
            amount0 += a0 + f0;
            amount1 += a1 + f1;
            fee0 += f0;
            fee1 += f1;
        }
    }

    IArrakisV2 arrakisV2 = IArrakisV2(underlyingPayload_.self);

    amount0 +=
        IERC20(underlyingPayload_.token0).balanceOf(
            underlyingPayload_.self
        ) -
        arrakisV2.managerBalance0() -
        arrakisV2.arrakisBalance0();
    amount1 +=
        IERC20(underlyingPayload_.token1).balanceOf(
            underlyingPayload_.self
        ) -
        arrakisV2.managerBalance1() -
        arrakisV2.arrakisBalance1();
}
```

Recommendation

Consider making `totalUnderlyingWithFees()` always returns the underlying amounts and fees, with the manager and protocol fees deducted from the uncollected fees.

Status

✓ **Fixed**

[WP-H5] `ArrakisV2Resolver#standardBurnParams()` Double counting for balances in underlying amounts

High

Issue Description

At L314, `totalUnderlyingWithFees()` already includes account balances in `amount0` and `amount1` (`Underlying.sol#L61-L72`), but `ArrakisV2Resolver.sol#L323-L328` added them again at `ArrakisV2Resolver.sol#L323-L328` and `L338-L339` .

https://github.com/ArrakisFinance/vault-v2-core/blob/fefa7ddbfe7c984a5925c58b163b88cb007d9ae5/contracts/ArrakisV2Resolver.sol#L297-L389

```
297     function standardBurnParams(uint256 amountToBurn_, IArrakisV2 vaultV2_)
298         external
299         view
300         returns (BurnLiquidity[] memory burns)
301     {
302         uint256 totalSupply = vaultV2_.totalSupply();
303         require(totalSupply > 0, "total supply");
304
305         Range[] memory ranges = helper.ranges(vaultV2_);
306
307         {
308             UnderlyingOutput memory underlying;
309             (
310                 underlying.amount0,
311                 underlying.amount1,
312                 underlying.fee0,
313                 underlying.fee1
314             ) = UnderlyingHelper.totalUnderlyingWithFees(
315                 UnderlyingPayload({
316                     ranges: ranges,
317                     factory: factory,
318                     token0: address(vaultV2_.token0()),
319                     token1: address(vaultV2_.token1()),
320                     self: address(vaultV2_)
321                 })
322             );
```

```
            underlying.leftOver0 = vaultV2_.token0().balanceOf(
                address(vaultV2_)
            );
            underlying.leftOver1 = vaultV2_.token1().balanceOf(
                address(vaultV2_)
            );

            {
                (uint256 fee0, uint256 fee1) = UniswapV3Amounts
                    .subtractAdminFees(
                        underlying.fee0,
                        underlying.fee1,
                        vaultV2_.manager().managerFeeBPS(),
                        vaultV2_.arrakisFeeBPS()
                    );
                underlying.amount0 += underlying.leftOver0 + fee0;
                underlying.amount1 += underlying.leftOver1 + fee1;
            }

            {
                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.

        burns = new BurnLiquidity[](ranges.length);

        for (uint256 i = 0; i < ranges.length; i++) {
            uint128 liquidity;
```

```
                {
                    (liquidity, , , , ) = IUniswapV3Pool(
                        vaultV2_.factory().getPool(
                            address(vaultV2_.token0()),
                            address(vaultV2_.token1()),
                            ranges[i].feeTier
                        )
                    ).positions(
                            PositionHelper.getPositionId(
                                address(vaultV2_),
                                ranges[i].lowerTick,
                                ranges[i].upperTick
                            )
                        );
                }

                burns[i] = BurnLiquidity({
                    liquidity: SafeCast.toUint128(
                        FullMath.mulDiv(liquidity, amountToBurn_, totalSupply)
                    ),
                    range: ranges[i]
                });
            }
        }
```

https://github.com/ArrakisFinance/vault-v2-core/blob/fefa7ddbfe7c984a5925c58b163b88cb007d9ae5/contracts/libraries/Underlying.sol#L24-L73

```
    function totalUnderlyingWithFees(
            UnderlyingPayload memory underlyingPayload_
        )
            public
            view
            returns (
                uint256 amount0,
                uint256 amount1,
                uint256 fee0,
                uint256 fee1
            )
        {
            for (uint256 i = 0; i < underlyingPayload_.ranges.length; i++) {
                {
                    IUniswapV3Pool pool = IUniswapV3Pool(
                        underlyingPayload_.factory.getPool(
                            underlyingPayload_.token0,
                            underlyingPayload_.token1,
                            underlyingPayload_.ranges[i].feeTier
                        )
                    );
                    (uint256 a0, uint256 a1, uint256 f0, uint256 f1) = underlying(
                        RangeData({
                            self: underlyingPayload_.self,
                            range: underlyingPayload_.ranges[i],
                            pool: pool
                        })
                    );
                    amount0 += a0 + f0;
                    amount1 += a1 + f1;
                    fee0 += f0;
                    fee1 += f1;
                }
            }

            IArrakisV2 arrakisV2 = IArrakisV2(underlyingPayload_.self);

            amount0 +=
                IERC20(underlyingPayload_.token0).balanceOf(
                    underlyingPayload_.self
                ) -
                arrakisV2.managerBalance0() -
                arrakisV2.arrakisBalance0();
            amount1 +=
                IERC20(underlyingPayload_.token1).balanceOf(
                    underlyingPayload_.self
                ) -
                arrakisV2.managerBalance1() -
                arrakisV2.arrakisBalance1();
        }
```

Recommendation

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

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

        {
            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_)
            );
            underlying.leftOver1 = vaultV2_.token1().balanceOf(
                address(vaultV2_)
            );

            {
                (uint256 fee0, uint256 fee1) = UniswapV3Amounts
                    .subtractAdminFees(
                        underlying.fee0,
                        underlying.fee1,
                        vaultV2_.manager().managerFeeBPS(),
                        vaultV2_.arrakisFeeBPS()
                    );
                underlying.amount0 += fee0;
                underlying.amount1 += fee1;
            }

            {
                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.

        burns = new BurnLiquidity[](ranges.length);

        for (uint256 i = 0; i < ranges.length; i++) {
            uint128 liquidity;
            {
                (liquidity, , , , ) = IUniswapV3Pool(
                    vaultV2_.factory().getPool(
                        address(vaultV2_.token0()),
                        address(vaultV2_.token1()),
                        ranges[i].feeTier
                    )
                ).positions(
                        PositionHelper.getPositionId(
                            address(vaultV2_),
                            ranges[i].lowerTick,
                            ranges[i].upperTick
                        )
                    );
            }

            burns[i] = BurnLiquidity({
                liquidity: SafeCast.toUint128(
                    FullMath.mulDiv(liquidity, amountToBurn_, totalSupply)
                ),
                range: ranges[i]
            });
        }
    }
```

Status

✓ Fixed

[WP-I6] `init0` and `init1` can both be set to 0 at the same time using `setInits()`

Informational

Issue Description

`initialize()` has the checks to ensure at least one is not 0:

https://github.com/ArrakisFinance/vault-v2-core/blob/27004a99dc61dc19502538434841ae72433200be/contracts/abstract/ArrakisV2Storage.sol#L178-L223

```
    function initialize(
        string calldata name_,
        string calldata symbol_,
        InitializePayload calldata params_
    ) external initializer {
        require(params_.feeTiers.length > 0, "NFT");
        require(params_.token0 != address(0), "T0");
        require(params_.token0 < params_.token1, "WTO");

        require(params_.init0 > 0 || params_.init1 > 0, "I");

@@ 189,222 @@
    }
```

However, `setInits()` allows both to be 0:

https://github.com/ArrakisFinance/vault-v2-core/blob/27004a99dc61dc19502538434841ae72433200be/contracts/abstract/ArrakisV2Storage.sol#L226-L229

```
    function setInits(uint256 init0_, uint256 init1_) external onlyOwner {
        require(totalSupply() == 0, "total supply");
        emit LogSetInits(address(this), init0 = init0_, init1 = init1_);
    }
```

Recommendation

Change to:

```
    function setInits(uint256 init0_, uint256 init1_) external onlyOwner {
        require(totalSupply() == 0, "total supply");
        require(init0_ > 0 || init1_ > 0, "I");
        emit LogSetInits(address(this), init0 = init0_, init1 = init1_);
    }
```

Similarly, `setManager()` can set manager to `address(0)` while this is not allowed in `initialize()` :

https://github.com/ArrakisFinance/vault-v2-core/blob/27004a99dc61dc19502538434841ae72433200be/contracts/abstract/ArrakisV2Storage.sol#L246-L252

```
    function setManager(IManagerProxyV2 manager_) external onlyOwner {
        emit LogSetManager(
            address(this),
            address(manager),
            address(manager = manager_)
        );
    }
```

```
    require(params_.manager != address(0), "NAZM");
```

Status

✓ Fixed

[WP-M7] `_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
```

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

    // intentional underflow revert if managerBalance > contract's token balance
    uint256 leftover0 = token0.balanceOf(address(this)) - managerBalance0;
    uint256 leftover1 = token1.balanceOf(address(this)) - managerBalance1;

    require(
        (leftover0 <= underlying.leftOver0) ||
            ((leftover0 - underlying.leftOver0) <=
                FullMath.mulDiv(total.burn0, _burnBuffer, hundredPercent)),
        "L0"
    );
    require(
        (leftover1 <= underlying.leftOver1) ||
            ((leftover1 - underlying.leftOver1) <=
                FullMath.mulDiv(total.burn1, _burnBuffer, hundredPercent)),
        "L1"
    );

    // For monitoring how much user burn LP token for getting their token back.
    emit LPBurned(msg.sender, total.burn0, total.burn1);
    emit LogUncollectedFees(underlying.fee0, underlying.fee1);
    emit LogCollectedFees(total.fee0, total.fee1);
    emit LogBurn(receiver_, burnAmount_, amount0, amount1);
}
```

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

```
    function _withdraw(
        IUniswapV3Pool pool_,
        int24 lowerTick_,
        int24 upperTick_,
        uint128 liquidity_
    ) internal returns (Withdraw memory withdraw) {
        (withdraw.burn0, withdraw.burn1) = pool_.burn(
            lowerTick_,
            upperTick_,
            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-M8] `_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-M9] `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],
                        amountToBurn_,
                        totalSupply
                    )
                ),
                range: ranges[j]
            });
            ++idx;
        }
    }
}
```

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

```
function burn(
    BurnLiquidity[] calldata burns_,
    uint256 burnAmount_,
    address receiver_
) external nonReentrant returns (uint256 amount0, uint256 amount1) {
@@ 140,192 @@
    _burn(msg.sender, burnAmount_);

    Withdraw memory total;
    {
        for (uint256 i; i < burns_.length; i++) {
            require(burns_[i].liquidity != 0, "LZ");
            {
                (bool exist, ) = Position.rangeExist(
                    ranges,
                    burns_[i].range
                );
                require(exist, "RRNE");
            }

            Withdraw memory withdraw = _withdraw(
                IUniswapV3Pool(
                    factory.getPool(
                        address(token0),
                        address(token1),
                        burns_[i].range.feeTier
213                     )
214                 ),
215                 burns_[i].range.lowerTick,
216                 burns_[i].range.upperTick,
217                 burns_[i].liquidity
218             );
219
220             total.fee0 += withdraw.fee0;
221             total.fee1 += withdraw.fee1;
222
223             total.burn0 += withdraw.burn0;
224             total.burn1 += withdraw.burn1;
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

    @@ 238,259 @@

260 }
```

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-I10] 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-I11] 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-H12] Attacker can manipulate the price (`tick`) of the Uniswap V3 pool and `burn()` vault shares at a higher price to steal funds

High

Issue Description

https://github.com/ArrakisFinance/vault-v2-core/blob/930e2f70fe8de6fab95c2bfa8f768a68489a19ee/contracts/ArrakisV2.sol#L129-L276

```
    function burn(
        BurnLiquidity[] calldata burns_,
        uint256 burnAmount_,
        address receiver_
    ) external nonReentrant returns (uint256 amount0, uint256 amount1) {
        require(burnAmount_ > 0, "BA");

        uint256 ts = totalSupply();
        require(ts > 0, "TS");

        UnderlyingOutput memory underlying;
        (
            underlying.amount0,
            underlying.amount1,
            underlying.fee0,
            underlying.fee1
        ) = UnderlyingHelper.totalUnderlyingWithFees(
            UnderlyingPayload({
                ranges: ranges,
                factory: factory,
                token0: address(token0),
                token1: address(token1),
                self: address(this)
            })
        );
        underlying.leftOver0 =
            token0.balanceOf(address(this)) -
            managerBalance0;
        underlying.leftOver1 =
                token1.balanceOf(address(this)) -
                managerBalance1;

        {
            // the proportion of user balance.
            amount0 = FullMath.mulDiv(underlying.amount0, burnAmount_, ts);
            amount1 = FullMath.mulDiv(underlying.amount1, burnAmount_, ts);
        }

        if (
            underlying.leftOver0 >= amount0 && underlying.leftOver1 >= amount1
        ) {
            _burn(msg.sender, burnAmount_);

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

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

            emit LogBurn(receiver_, burnAmount_, amount0, amount1);
            return (amount0, amount1);
        }
@@ 183,275 @@
    }
```

https://github.com/ArrakisFinance/vault-v2-core/blob/930e2f70fe8de6fab95c2bfa8f768a68489a19ee/contracts/libraries/Underlying.sol#L25-L80

```
    function totalUnderlyingWithFees(
        UnderlyingPayload memory underlyingPayload_
    )
        public
        view
        returns (
            uint256 amount0,
            uint256 amount1,
            uint256 fee0,
            uint256 fee1
        )
    {
        for (uint256 i; i < underlyingPayload_.ranges.length; i++) {
            {
                IUniswapV3Pool pool = IUniswapV3Pool(
                    underlyingPayload_.factory.getPool(
                        underlyingPayload_.token0,
                        underlyingPayload_.token1,
                        underlyingPayload_.ranges[i].feeTier
                    )
                );
                (uint256 a0, uint256 a1, uint256 f0, uint256 f1) = underlying(
                    RangeData({
                        self: underlyingPayload_.self,
                        range: underlyingPayload_.ranges[i],
                        pool: pool
                    })
                );
                amount0 += a0;
                amount1 += a1;
                fee0 += f0;
                fee1 += f1;
            }
        }
@@ 59,79 @@
    }
```

https://github.com/ArrakisFinance/vault-v2-core/blob/
930e2f70fe8de6fab95c2bfa8f768a68489a19ee/contracts/libraries/Underlying.sol#L82-L109

```
    function underlying(RangeData memory underlying_)
        public
        view
        returns (
            uint256 amount0,
            uint256 amount1,
            uint256 fee0,
            uint256 fee1
        )
```

```solidity
    {
        (uint160 sqrtPriceX96, int24 tick, , , , , ) = underlying_.pool.slot0();
        bytes32 positionId = Position.getPositionId(
            underlying_.self,
            underlying_.range.lowerTick,
            underlying_.range.upperTick
        );
        PositionUnderlying memory positionUnderlying = PositionUnderlying({
            positionId: positionId,
            sqrtPriceX96: sqrtPriceX96,
            tick: tick,
            lowerTick: underlying_.range.lowerTick,
            upperTick: underlying_.range.upperTick,
            pool: underlying_.pool
        });
        (amount0, amount1, fee0, fee1) = getUnderlyingBalances(
            positionUnderlying
        );
    }
```

https://github.com/ArrakisFinance/vault-v2-core/blob/930e2f70fe8de6fab95c2bfa8f768a68489a19ee/contracts/libraries/Underlying.sol#L112-L156

```solidity
    function getUnderlyingBalances(
        PositionUnderlying memory positionUnderlying_
    )
        public
        view
        returns (
            uint256 amount0Current,
            uint256 amount1Current,
            uint256 fee0,
            uint256 fee1
        )
    {
@@ 124,144 @@
        // compute current holdings from liquidity
        (amount0Current, amount1Current) = LiquidityAmounts
            .getAmountsForLiquidity(
                positionUnderlying_.sqrtPriceX96,
                TickMath.getSqrtRatioAtTick(positionUnderlying_.lowerTick),
                TickMath.getSqrtRatioAtTick(positionUnderlying_.upperTick),
                liquidity
            );

        fee0 += uint256(tokensOwed0);
        fee1 += uint256(tokensOwed1);
    }
```

When calculating the `token0` and `token1` amounts in the UniswapV3 pool (`getUnderlyingBalances()`), `pool.slot0` 's `sqrtPriceX96` is used directly.

However, the `sqrtPriceX96` may be manipulated (only in the current transaction) resulting in the calculated `token0` and `token1` amounts being distorted numbers that mismatch the actual fair market share price.

This is not an issue if the `token0` and `token1` used to pay for the burned shares are taken from the burned liquidity in the underlying UniswapV3 pool.

But ArrakisV2 has a certain amount of `token0` and `token1` on the balance of the vault contract, which works as a buffer for the burns.

As a result, there will be no change in liquidity on the underlying pool during the burn. This allows the attacker to reverse the price by swapping it back at a minimal cost.

In essence, this issue allows the attacker to manipulate the price and get underlying assets (on the vault contract's balance) at a lower rate through `burn()` .

PoC

To demonstrate the issue more easily, we are using smaller numbers in the following example.

Given:

- Pool: USDC / WETH
- Current ETH price: 2000 USDC / ETH
- Arrakisv2 vault totalSupply: 3.0
- Arrakisv2 vault's underlying assets:
    - vault contract balance:

    - usdc: 0.0
    - eth: 3.0
  - in the uniswap v3 pool:
    - usdc: 3000.0
    - eth: 0.0
  - total value: 3000.0 + 3.0 * 2000 = $9000
- Attacker's funds:
  - usdc: 33000.0
  - eth: 33.0
  - total value: 33000.0 + 33.0 * 2000 = $99000

When:

- attacker calls `arrakisV2.mint(33.0, attacker)`
  - Arrakisv2 vault totalSupply: 3.0 -> 36.0
  - Arrakisv2 vault's underlying assets:
    - in vault
      - usdc: 0.0 -> 33000.0
      - eth: 3.0 -> 36.0
    - in uniswap v3 pool: (unchanged)
- attacker manipulated `uniswapv3Pool.slot0` 's sqrtPriceX96 to 999 USDC / ETH (swap eth to USDC)
- attacker calls `arrakisV2.burn(_, 33.0, attacker)`
  - arrakisV2 `getUnderlyingBalances()` (using the manipulated price):
    - vault contract balance:
      - usdc: 33000.0
      - eth: 36.0
    - in uniswap v3 pool (1000 USDC / ETH) (unchanged)
      - usdc: 0.0
      - eth: 3.0
  - based on that, `arrakisV2.burn()` L163-L164 will result in:
    - amount0 (usdc): 33000.0 * 33.0 / 36.0 = 30250.0
    - amount1 (eth): 39.0 * 33.0 / 36.0 = 35.75
  - L168 will use vault balance of token0, token1 to pay:
    - Arrakisv2 vault totalSupply: 36.0 -> 3.0
    - vault contract balance:
      - usdc: 33000.0 -> 2750.0

        · eth: 36.0 -> 0.25
      * in uniswap v3 pool: (unchanged)
- attacker reverse `uniswapv3Pool.slot0` 's `sqrtPriceX96` back to 2000 USDC / ETH (swap USDC back to ETH)

Then:

- Attacker's funds:
  - usdc: 30250.0
  - eth: 35.75
  - total value: 30250.0 + 35.75 * 2000 = $101750
  - That's `$2750` more than the original `$99000`
- Arrakisv2 vault's underlying assets:
  - vault contract balance:
    * usdc: 2750.0
    * eth: 0.25
  - in uniswap v3 pool:
    * usdc: 3000.0
    * eth: 0.0
  - total value: 5750.0 + 0.25 * 2000 = $6250
  - That's `$2750` short than the original `$9000`

Recommendation

We recommend using a `simpleBurn` method.

This method would burn the user's shares and return the vault balances and underlying liquidities proportionally to the shares burnt.

```
function simpleBurn(
    uint256 burnAmount_,
    address receiver_
) external nonReentrant returns (uint256 amount0, uint256 amount1) {
    require(burnAmount_ > 0, "BA");

    uint256 ts = totalSupply();
    require(ts > 0, "TS");

    _burn(msg.sender, burnAmount_);

    Withdraw memory total;
    {
        for (uint256 i; i < ranges.length; i++) {
            uint256 liquidity = getLiquidityByRange(ranges[i]);
            if (liquidity == 0) continue;

            Withdraw memory withdraw = _withdraw(
                IUniswapV3Pool(
                    factory.getPool(
                        address(token0),
                        address(token1),
                        range.feeTier
                    )
                ),
                range.lowerTick,
                range.upperTick,
                FullMath.mulDiv(liquidity, burnAmount_, ts);
            );

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

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

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

    uint256 leftOver0 = token0.balanceOf(address(this)) - managerBalance0 -
total.burn0;
    uint256 leftOver1 = token1.balanceOf(address(this)) - managerBalance1 -
total.burn1;

    // the proportion of user balance.
    amount0 = FullMath.mulDiv(leftOver0, burnAmount_, ts);
    amount1 = FullMath.mulDiv(leftOver0, burnAmount_, ts);

    amount0 += total.burn0;
    amount1 += total.burn1;

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

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

    // TODO: ADD EVENTS
}
```

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