



Arrakis Finance v2 Vault Core Audit Report

Jan 31, 2023



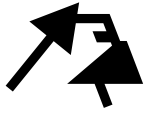


Table of Contents

Summary	2
Overview	3
Issues	4
[WP-H1] <code>ArrakisV2#rebalance()</code> Dangerous arbitrary external call can be used by the manager to steal funds from the users who have approved tokens to the vault contract	4
[WP-M2] Swap requires all the amountIn to be spent precisely making the transaction prone to revert	5
[WP-H3] <code>ArrakisV2#rebalance()</code> may spend part of the managerBalance + arrakisBalance in the balance and cause <code>burn()</code> to revert	8
[WP-H4] <code>amount0</code> , <code>amount1</code> returned from <code>Underlying.totalUnderlyingWithFees()</code> is larger than the actual amounts as the admin and protocol fees are not deducted from the uncollected fees	10
[WP-H5] <code>ArrakisV2Resolver#standardBurnParams()</code> Double counting for balances in underlying amounts	13
[WP-I6] <code>init0</code> and <code>init1</code> can both be set to 0 at the same time using <code>setInits()</code>	20
[WP-M7] <code>_burnBuffer</code> mishandled the fee which could result in some users being unable to withdraw	22
[WP-M8] <code>_rebalance()</code> Lack of slippage control for <code>burns</code>	26
[WP-M9] <code>VaultV2.burn()</code> may revert as the <code>BurnLiquidity[]</code> burns returned by <code>ArrakisV2Resolver.standardBurnParams()</code> can be slightly smaller than expected	29
[WP-I10] Consider adding <code>nonReentrant</code> modifier to <code>withdrawManagerBalance()</code>	33
[WP-I11] Inconsistent <code>address(0)</code> check in <code>upgradeVaults()</code> and <code>upgradeVaultsAndCall()</code>	34
[WP-H12] Attacker can manipulate the price (<code>tick</code>) of the Uniswap V3 pool and <code>burn()</code> vault shares at a higher price to steal funds	36
Appendix	45
Disclaimer	46



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.



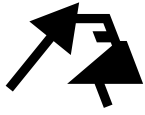
Overview

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 Issues	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>

```
382 (bool success, ) = rebalanceParams_.swap.router.call(
383     rebalanceParams_.swap.payload
384 );
385 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.

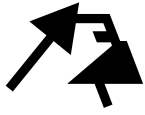
Recommendation

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         @@ 315,335 @@  
316  
336         uint256 balance0 = token0.balanceOf(address(this));  
337         uint256 balance1 = token1.balanceOf(address(this));  
338
```

```

339         if (_swapData.zeroForOne) {
340             amount0Diff = balance0Before - balance0;
341             amount1Diff = balance1 - balance1Before;
342             require(
343                 (amount0Diff == _swapData.amountInSwap) &&
344                 (amount1Diff >= _swapData.amountOutSwap),
345                 "Token0 swap failed!"
346             );
347         } else {
348             amount0Diff = balance0 - balance0Before;
349             amount1Diff = balance1Before - balance1;
350             require(
351                 (amount0Diff >= _swapData.amountOutSwap) &&
352                 (amount1Diff == _swapData.amountInSwap),
353                 "Token1 swap failed!"
354             );
355         }
356
357         @@ 357,362 @@
363     }

```

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

```

2087 {
2088     bytes memory callData = abi.encodePacked(caller.callBytes.selector,
2089         bytes12(0), msg.sender, data);
2089     // solhint-disable-next-line avoid-low-level-calls
2090     (bool success, bytes memory result) = address(caller).call{value:
2091         msg.value}(callData);
2091     if (!success) {
2092         revert(RevertReasonParser.parse(result, "callBytes failed: "));
2093     }
2094 }
2095
2096 spentAmount = desc.amount;
2097 returnAmount = dstToken.uniBalanceOf(address(this));
2098
2099 if (flags & _PARTIAL_FILL != 0) {
2100     uint256 unspentAmount = srcToken.uniBalanceOf(address(this));

```

```
2101     if (unspentAmount > 0) {
2102         spentAmount = spentAmount.sub(unspentAmount);
2103         srcToken.uniTransfer(msg.sender, unspentAmount);
2104     }
2105     require(returnAmount.mul(desc.amount) >=
desc.minReturnAmount.mul(spentAmount), "Return amount is not enough");
2106 } else {
2107     require(returnAmount >= desc.minReturnAmount, "Return amount is not enough");
2108 }
2109
2110 address payable dstReceiver = (desc.dstReceiver == address(0)) ? msg.sender :
desc.dstReceiver;
2111 dstToken.uniTransfer(dstReceiver, returnAmount);
```

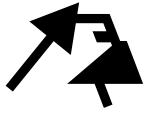
Requiring the balance after strictly equals the amountIn means that any unspent amount will revert the whole transaction.

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

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

```

24  function totalUnderlyingWithFees(
25      UnderlyingPayload memory underlyingPayload_
26  )
27      public
28      view
29      returns (
30          uint256 amount0,
31          uint256 amount1,
32          uint256 fee0,
33          uint256 fee1
34      )
35  {

```

```

36     for (uint256 i = 0; i < underlyingPayload_.ranges.length; i++) {
37         {
38             IUniswapV3Pool pool = IUniswapV3Pool(
39                 underlyingPayload_.factory.getPool(
40                     underlyingPayload_.token0,
41                     underlyingPayload_.token1,
42                     underlyingPayload_.ranges[i].feeTier
43                 )
44             );
45             (uint256 a0, uint256 a1, uint256 f0, uint256 f1) = underlying(
46                 RangeData({
47                     self: underlyingPayload_.self,
48                     range: underlyingPayload_.ranges[i],
49                     pool: pool
50                 })
51             );
52             amount0 += a0 + f0;
53             amount1 += a1 + f1;
54             fee0 += f0;
55             fee1 += f1;
56         }
57     }
58
59     IArrakisV2 arrakisV2 = IArrakisV2(underlyingPayload_.self);
60
61     amount0 +=
62         IERC20(underlyingPayload_.token0).balanceOf(
63             underlyingPayload_.self
64         ) -
65         arrakisV2.managerBalance0() -
66         arrakisV2.arrakisBalance0();
67     amount1 +=
68         IERC20(underlyingPayload_.token1).balanceOf(
69             underlyingPayload_.self
70         ) -
71         arrakisV2.managerBalance1() -
72         arrakisV2.arrakisBalance1();
73 }

```



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/fefa7ddbf7c984a5925c58b163b88cb007d9ae5/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             );

```

```

323         underlying.leftOver0 = vaultV2_.token0().balanceOf(
324             address(vaultV2_)
325         );
326         underlying.leftOver1 = vaultV2_.token1().balanceOf(
327             address(vaultV2_)
328         );
329
330         {
331             (uint256 fee0, uint256 fee1) = UniswapV3Amounts
332                 .subtractAdminFees(
333                     underlying.fee0,
334                     underlying.fee1,
335                     vaultV2_.manager().managerFeeBPS(),
336                     vaultV2_.arrakisFeeBPS()
337                 );
338             underlying.amount0 += underlying.leftOver0 + fee0;
339             underlying.amount1 += underlying.leftOver1 + fee1;
340         }
341
342         {
343             uint256 amount0 = FullMath.mulDiv(
344                 underlying.amount0,
345                 amountToBurn_,
346                 totalSupply
347             );
348             uint256 amount1 = FullMath.mulDiv(
349                 underlying.amount1,
350                 amountToBurn_,
351                 totalSupply
352             );
353
354             if (
355                 amount0 <= underlying.leftOver0 &&
356                 amount1 <= underlying.leftOver1
357             ) return burns;
358         }
359     }
360     // #endregion get amount to burn.
361
362     burns = new BurnLiquidity[](ranges.length);
363
364     for (uint256 i = 0; i < ranges.length; i++) {
365         uint128 liquidity;

```

```

366         {
367             (liquidity, , , , ) = IUniswapV3Pool(
368                 vaultV2_.factory().getPool(
369                     address(vaultV2_.token0()),
370                     address(vaultV2_.token1()),
371                     ranges[i].feeTier
372                 )
373             ).positions(
374                 PositionHelper.getPositionId(
375                     address(vaultV2_),
376                     ranges[i].lowerTick,
377                     ranges[i].upperTick
378                 )
379             );
380         }
381
382         burns[i] = BurnLiquidity({
383             liquidity: SafeCast.toUint128(
384                 FullMath.mulDiv(liquidity, amountToBurn_, totalSupply)
385             ),
386             range: ranges[i]
387         });
388     }
389 }

```

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

```

24     function totalUnderlyingWithFees(
25         UnderlyingPayload memory underlyingPayload_
26     )
27     public
28     view
29     returns (
30         uint256 amount0,
31         uint256 amount1,
32         uint256 fee0,
33         uint256 fee1
34     )
35     {
36         for (uint256 i = 0; i < underlyingPayload_.ranges.length; i++) {

```



```

37         {
38             IUniswapV3Pool pool = IUniswapV3Pool(
39                 underlyingPayload_.factory.getPool(
40                     underlyingPayload_.token0,
41                     underlyingPayload_.token1,
42                     underlyingPayload_.ranges[i].feeTier
43                 )
44             );
45             (uint256 a0, uint256 a1, uint256 f0, uint256 f1) = underlying(
46                 RangeData({
47                     self: underlyingPayload_.self,
48                     range: underlyingPayload_.ranges[i],
49                     pool: pool
50                 })
51             );
52             amount0 += a0 + f0;
53             amount1 += a1 + f1;
54             fee0 += f0;
55             fee1 += f1;
56         }
57     }
58
59     IArrakisV2 arrakisV2 = IArrakisV2(underlyingPayload_.self);
60
61     amount0 +=
62         IERC20(underlyingPayload_.token0).balanceOf(
63             underlyingPayload_.self
64         ) -
65         arrakisV2.managerBalance0() -
66         arrakisV2.arrakisBalance0();
67     amount1 +=
68         IERC20(underlyingPayload_.token1).balanceOf(
69             underlyingPayload_.self
70         ) -
71         arrakisV2.managerBalance1() -
72         arrakisV2.arrakisBalance1();
73 }

```

Recommendation

```

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             );
323             underlying.leftOver0 = vaultV2_.token0().balanceOf(
324                 address(vaultV2_)
325             );
326             underlying.leftOver1 = vaultV2_.token1().balanceOf(
327                 address(vaultV2_)
328             );
329
330             {
331                 (uint256 fee0, uint256 fee1) = UniswapV3Amounts
332                     .subtractAdminFees(
333                         underlying.fee0,
334                         underlying.fee1,
335                         vaultV2_.manager().managerFeeBPS(),
336                         vaultV2_.arrakisFeeBPS()

```

```

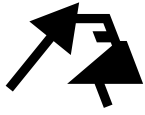
337         );
338         underlying.amount0 += fee0;
339         underlying.amount1 += fee1;
340     }
341
342     {
343         uint256 amount0 = FullMath.mulDiv(
344             underlying.amount0,
345             amountToBurn_,
346             totalSupply
347         );
348         uint256 amount1 = FullMath.mulDiv(
349             underlying.amount1,
350             amountToBurn_,
351             totalSupply
352         );
353
354         if (
355             amount0 <= underlying.leftOver0 &&
356             amount1 <= underlying.leftOver1
357         ) return burns;
358     }
359 }
360 // #endregion get amount to burn.
361
362 burns = new BurnLiquidity[](ranges.length);
363
364 for (uint256 i = 0; i < ranges.length; i++) {
365     uint128 liquidity;
366     {
367         (liquidity, , , , ) = IUniswapV3Pool(
368             vaultV2_.factory().getPool(
369                 address(vaultV2_.token0()),
370                 address(vaultV2_.token1()),
371                 ranges[i].feeTier
372             )
373         ).positions(
374             PositionHelper.getPositionId(
375                 address(vaultV2_),
376                 ranges[i].lowerTick,
377                 ranges[i].upperTick
378             )
379         );

```

```
380         }
381
382         burns[i] = BurnLiquidity({
383             liquidity: SafeCast.toUint128(
384                 FullMath.mulDiv(liquidity, amountToBurn_, totalSupply)
385             ),
386             range: ranges[i]
387         });
388     }
389 }
```

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>

```
178     function initialize(  
179         string calldata name_,  
180         string calldata symbol_,  
181         InitializePayload calldata params_  
182     ) external initializer {  
183         require(params_.feeTiers.length > 0, "NFT");  
184         require(params_.token0 != address(0), "T0");  
185         require(params_.token0 < params_.token1, "WTO");  
186  
187         require(params_.init0 > 0 || params_.init1 > 0, "I");  
188  
189         @@ 189,222 @@  
223     }
```

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

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

```
226     function setInits(uint256 init0_, uint256 init1_) external onlyOwner {  
227         require(totalSupply() == 0, "total supply");  
228         emit LogSetInits(address(this), init0 = init0_, init1 = init1_);  
229     }
```

Recommendation

Change to:

```

226     function setInits(uint256 init0_, uint256 init1_) external onlyOwner {
227         require(totalSupply() == 0, "total supply");
228         require(init0_ > 0 || init1_ > 0, "I");
229         emit LogSetInits(address(this), init0 = init0_, init1 = init1_);
230     }

```

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>

```

246     function setManager(IManagerProxyV2 manager_) external onlyOwner {
247         emit LogSetManager(
248             address(this),
249             address(manager),
250             address(manager = manager_)
251         );
252     }

```

```

1     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

```

```

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_

```



```

467         );
468
469         (uint256 collect0, uint256 collect1) = pool_.collect(
470             address(this),
471             lowerTick_,
472             upperTick_,
473             type(uint128).max,
474             type(uint128).max
475         );
476
477         withdraw.fee0 = collect0 - withdraw.burn0;
478         withdraw.fee1 = collect1 - withdraw.burn1;
479     }

```

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:

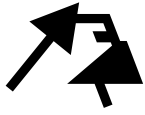
```

240 fee0AfterManagerFee = (fee0_ * (hundredPercent - managerFeeBPS)) / hundredPercent;
241 fee1AfterManagerFee = (fee1_ * (hundredPercent - managerFeeBPS)) / hundredPercent;
242 require(
243     ( fee0AfterManagerFee >= leftover0 ||
244       leftover0 - fee0AfterManagerFee <= underlying.leftOver0) ||
245     ((leftover0 - fee0AfterManagerFee - underlying.leftOver0) <=
246       FullMath.mulDiv(total.burn0, _burnBuffer, hundredPercent)),
247     "L0"
248 );
249 require(
250     ( fee1AfterManagerFee >= leftover1 ||
251       leftover1 - fee1AfterManagerFee <= underlying.leftOver1) ||
252     ((leftover1 - fee1AfterManagerFee - underlying.leftOver1) <=
253       FullMath.mulDiv(total.burn1, _burnBuffer, hundredPercent)),
254     "L1"
255 );

```

Status

✓ Fixed



[WP-M8] `_rebalance()` Lack of slippage control for burns

Medium

Issue Description

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

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

```
336 function _rebalance(Rebalance calldata rebalanceParams_)
337     internal
338     nonReentrant
339 {
340     // Burns.
341     uint256 aggregator0 = 0;
342     uint256 aggregator1 = 0;
343     IUniswapV3Factory mFactory = factory;
344     address mToken0Addr = address(token0);
345     address mToken1Addr = address(token1);
346     for (uint256 i; i < rebalanceParams_.removes.length; i++) {
347         address poolAddr = mFactory.getPool(
348             mToken0Addr,
349             mToken1Addr,
350             rebalanceParams_.removes[i].range.feeTier
351         );
352         IUniswapV3Pool pool = IUniswapV3Pool(poolAddr);
353
354         Withdraw memory withdraw = _withdraw(
355             pool,
356             rebalanceParams_.removes[i].range.lowerTick,
357             rebalanceParams_.removes[i].range.upperTick,
358             rebalanceParams_.removes[i].liquidity
359         );
360
361         aggregator0 += withdraw.fee0;
362         aggregator1 += withdraw.fee1;
363     }
364
365     if (aggregator0 > 0 || aggregator1 > 0) {
366         _applyFees(aggregator0, aggregator1);
367
368         emit LogCollectedFees(aggregator0, aggregator1);
```

```

369     }
370
371     // Swap.
    @@ 372,452 @@
453
454     emit LogRebalance(rebalanceParams_);
455 }

```

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>

```

139 struct DecreaseLiquidityParams {
140     uint256 tokenId;
141     uint128 liquidity;
142     uint256 amount0Min;
143     uint256 amount1Min;
144     uint256 deadline;
145 }
146
147 /// @notice Decreases the amount of liquidity in a position and accounts it to the
    position
148 /// @param params tokenId The ID of the token for which liquidity is being
    decreased,
149 /// amount The amount by which liquidity will be decreased,
150 /// amount0Min The minimum amount of token0 that should be accounted for the
    burned liquidity,
151 /// amount1Min The minimum amount of token1 that should be accounted for the
    burned liquidity,

```



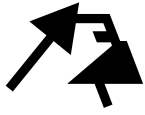
```
152  /// deadline The time by which the transaction must be included to effect the
153  change
154  /// @return amount0 The amount of token0 accounted to the position's tokens owed
155  /// @return amount1 The amount of token1 accounted to the position's tokens owed
156  function decreaseLiquidity(DecreaseLiquidityParams calldata params)
157      external
158      payable
159      returns (uint256 amount0, uint256 amount1);
160
161  struct CollectParams {
162      uint256 tokenId;
163      address recipient;
164      uint128 amount0Max;
165      uint128 amount1Max;
166  }
```

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      @@ 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     @@ 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

```

```

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 @@ 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             )

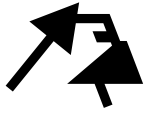
```

```
232         ),  
233         range: ranges[j]  
234     });  
235     ++idx;  
236 }
```

Status

✓ Fixed



[WP-I10] Consider adding `nonReentrant` modifier to `withdrawManagerBalance()`

Informational

Issue Description

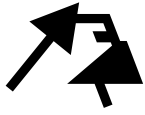
<https://github.com/ArrakisFinance/vault-v2-core/blob/d958ffd0e9ed7890b55d8ade4fdc26eae9640ab3/contracts/ArrakisV2.sol#L317-L333>

```
317 function withdrawManagerBalance() external {
318     uint256 amount0 = managerBalance0;
319     uint256 amount1 = managerBalance1;
320
321     managerBalance0 = 0;
322     managerBalance1 = 0;
323
324     if (amount0 > 0) {
325         token0.safeTransfer(manager, amount0);
326     }
327
328     if (amount1 > 0) {
329         token1.safeTransfer(manager, amount1);
330     }
331
332     emit LogWithdrawManagerBalance(amount0, amount1);
333 }
```

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>

```
49     function upgradeVaults(address[] memory vaults_) external onlyOwner {
50         address implementation = arrakisV2Beacon.implementation();
51         require(implementation != address(0), "implementation is address zero");
52         for (uint256 i = 0; i < vaults_.length; i++) {
53             ITransparentUpgradeableProxy(vaults_[i]).upgradeTo(implementation);
54         }
55     }
```

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

```
62     function upgradeVaultsAndCall(
63         address[] memory vaults_,
64         bytes[] calldata datas_
65     ) external onlyOwner {
66         require(vaults_.length == datas_.length, "mismatching array length");
67         for (uint256 i = 0; i < vaults_.length; i++) {
68             ITransparentUpgradeableProxy(vaults_[i]).upgradeToAndCall(
69                 arrakisV2Beacon.implementation(),
```

```

70         datas_[i]
71     );
72 }
73 }

```

Recommendation

Change to:

```

49 function upgradeVaults(address[] memory vaults_, address implementation_) external
    onlyOwner {
50     address implementation = arrakisV2Beacon.implementation();
51     require(implementation == implementation_, "implementation mismatch");
52     for (uint256 i = 0; i < vaults_.length; i++) {
53         ITransparentUpgradeableProxy(vaults_[i]).upgradeTo(implementation);
54     }
55 }

```

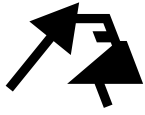
```

62 function upgradeVaultsAndCall(
63     address[] memory vaults_,
64     bytes[] calldata datas_,
65     address implementation_
66 ) external onlyOwner {
67     address implementation = arrakisV2Beacon.implementation();
68     require(implementation == implementation_, "implementation mismatch");
69     require(vaults_.length == datas_.length, "mismatching array length");
70     for (uint256 i = 0; i < vaults_.length; i++) {
71         ITransparentUpgradeableProxy(vaults_[i]).upgradeToAndCall(
72             arrakisV2Beacon.implementation(),
73             datas_[i]
74         );
75     }
76 }

```

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>

```
129     function burn(  
130         BurnLiquidity[] calldata burns_,  
131         uint256 burnAmount_,  
132         address receiver_  
133     ) external nonReentrant returns (uint256 amount0, uint256 amount1) {  
134         require(burnAmount_ > 0, "BA");  
135  
136         uint256 ts = totalSupply();  
137         require(ts > 0, "TS");  
138  
139         UnderlyingOutput memory underlying;  
140         (  
141             underlying.amount0,  
142             underlying.amount1,  
143             underlying.fee0,  
144             underlying.fee1  
145         ) = UnderlyingHelper.totalUnderlyingWithFees(  
146             UnderlyingPayload({  
147                 ranges: ranges,  
148                 factory: factory,  
149                 token0: address(token0),  
150                 token1: address(token1),  
151                 self: address(this)  
152             })  
153         );  
154         underlying.leftOver0 =  
155             token0.balanceOf(address(this)) -  
156             managerBalance0;  
157         underlying.leftOver1 =
```

```

158         token1.balanceOf(address(this)) -
159         managerBalance1;
160
161     {
162         // the proportion of user balance.
163         amount0 = FullMath.mulDiv(underlying.amount0, burnAmount_, ts);
164         amount1 = FullMath.mulDiv(underlying.amount1, burnAmount_, ts);
165     }
166
167     if (
168         underlying.leftOver0 >= amount0 && underlying.leftOver1 >= amount1
169     ) {
170         _burn(msg.sender, burnAmount_);
171
172         if (amount0 > 0) {
173             token0.safeTransfer(receiver_, amount0);
174         }
175
176         if (amount1 > 0) {
177             token1.safeTransfer(receiver_, amount1);
178         }
179
180         emit LogBurn(receiver_, burnAmount_, amount0, amount1);
181         return (amount0, amount1);
182     }
183
184     @@ 183,275 @@
185
276 }

```

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

```

25     function totalUnderlyingWithFees(
26         UnderlyingPayload memory underlyingPayload_
27     )
28     public
29     view
30     returns (
31         uint256 amount0,
32         uint256 amount1,
33         uint256 fee0,

```

```

34         uint256 fee1
35     )
36     {
37         for (uint256 i; i < underlyingPayload_.ranges.length; i++) {
38             {
39                 IUniswapV3Pool pool = IUniswapV3Pool(
40                     underlyingPayload_.factory.getPool(
41                         underlyingPayload_.token0,
42                         underlyingPayload_.token1,
43                         underlyingPayload_.ranges[i].feeTier
44                     )
45                 );
46                 (uint256 a0, uint256 a1, uint256 f0, uint256 f1) = underlying(
47                     RangeData({
48                         self: underlyingPayload_.self,
49                         range: underlyingPayload_.ranges[i],
50                         pool: pool
51                     })
52                 );
53                 amount0 += a0;
54                 amount1 += a1;
55                 fee0 += f0;
56                 fee1 += f1;
57             }
58         }
59     }
60 }

```

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

```

82     function underlying(RangeData memory underlying_)
83     public
84     view
85     returns (
86         uint256 amount0,
87         uint256 amount1,
88         uint256 fee0,
89         uint256 fee1
90     )

```

```

91     {
92         (uint160 sqrtPriceX96, int24 tick, , , , ) = underlying_.pool.slot0();
93         bytes32 positionId = Position.getPositionId(
94             underlying_.self,
95             underlying_.range.lowerTick,
96             underlying_.range.upperTick
97         );
98         PositionUnderlying memory positionUnderlying = PositionUnderlying({
99             positionId: positionId,
100             sqrtPriceX96: sqrtPriceX96,
101             tick: tick,
102             lowerTick: underlying_.range.lowerTick,
103             upperTick: underlying_.range.upperTick,
104             pool: underlying_.pool
105         });
106         (amount0, amount1, fee0, fee1) = getUnderlyingBalances(
107             positionUnderlying
108         );
109     }

```

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

```

112     function getUnderlyingBalances(
113         PositionUnderlying memory positionUnderlying_
114     )
115     public
116     view
117     returns (
118         uint256 amount0Current,
119         uint256 amount1Current,
120         uint256 fee0,
121         uint256 fee1
122     )
123     {
124         @@ 124,144 @@
125         // compute current holdings from liquidity
126         (amount0Current, amount1Current) = LiquidityAmounts
127             .getAmountsForLiquidity(
128                 positionUnderlying_.sqrtPriceX96,

```



```

149         TickMath.getSqrtRatioAtTick(positionUnderlying_.lowerTick),
150         TickMath.getSqrtRatioAtTick(positionUnderlying_.upperTick),
151         liquidity
152     );
153
154     fee0 += uint256(tokensOwed0);
155     fee1 += uint256(tokensOwed1);
156 }

```

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.

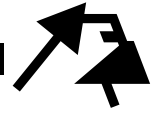
In essence, this issue allows the attacker to manipulate the price and get underlying assets (on the vault contract's balance) at a lower rate through `burn()`.

PoC

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

Given:

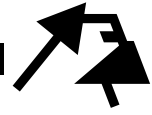
- Pool: USDC / WETH
- Current ETH price: 2000 USDC / ETH
- Arrakisv2 vault totalSupply: 3.0
- Arrakisv2 vault's underlying assets:
 - vault contract balance:



- * 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 `sqrPriceX96` 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.

```


1  function simpleBurn(
2      uint256 burnAmount_,
3      address receiver_
4  ) external nonReentrant returns (uint256 amount0, uint256 amount1) {
5      require(burnAmount_ > 0, "BA");
6
7      uint256 ts = totalSupply();
8      require(ts > 0, "TS");
9
10     _burn(msg.sender, burnAmount_);

```

```

11
12     Withdraw memory total;
13     {
14         for (uint256 i; i < ranges.length; i++) {
15             uint256 liquidity = getLiquidityByRange(ranges[i]);
16             if (liquidity == 0) continue;
17
18             Withdraw memory withdraw = _withdraw(
19                 IUniswapV3Pool(
20                     factory.getPool(
21                         address(token0),
22                         address(token1),
23                         range.feeTier
24                     )
25                 ),
26                 range.lowerTick,
27                 range.upperTick,
28                 FullMath.mulDiv(liquidity, burnAmount_, ts);
29             );
30
31             total.fee0 += withdraw.fee0;
32             total.fee1 += withdraw.fee1;
33
34             total.burn0 += withdraw.burn0;
35             total.burn1 += withdraw.burn1;
36         }
37
38         _applyFees(total.fee0, total.fee1);
39     }
40
41     uint256 leftOver0 = token0.balanceOf(address(this)) - managerBalance0 -
total.burn0;
42     uint256 leftOver1 = token1.balanceOf(address(this)) - managerBalance1 -
total.burn1;
43
44     // the proportion of user balance.
45     amount0 = FullMath.mulDiv(leftOver0, burnAmount_, ts);
46     amount1 = FullMath.mulDiv(leftOver0, burnAmount_, ts);
47
48     amount0 += total.burn0;
49     amount1 += total.burn1;
50
51     if (amount0 > 0) {

```



```
52         token0.safeTransfer(receiver_, amount0);
53     }
54
55     if (amount1 > 0) {
56         token1.safeTransfer(receiver_, amount1);
57     }
58
59     // TODO: ADD EVENTS
60 }
```



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

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Smart Contract technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.