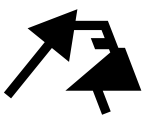
Arrakis Finance v2 Vault Core Audit Report

Oct 7, 2022



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	**27004a99dc61dc19502538434841ae72433200be**
Language	**Solidity**

Audit Summary

Delivery Date	**Oct 7, 2022**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**6**

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


```
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
    @@ 357,362 @@
363     }
```

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

```
2087 {
2088     bytes memory callData = abi.encodePacked(caller.callBytes.selector,
     bytes12(0), msg.sender, data);
2089     // solhint-disable-next-line avoid-low-level-calls
2090     (bool success, bytes memory result) = address(caller).call{value:
     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/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/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             );
```

```
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 

@@ 189,222 @@

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

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.