Arrakis Finance Modular Audit Report

Mar 27, 2023

Table of Contents

Summary

This report has been prepared for Arrakis Modular 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 Modular**
Codebase	**https://github.com/ArrakisFinance/arrakis-modular**
Commit	**33f270590fe7f0188d57620081069d85721de4fb**
Language	**Solidity**

Audit Summary

Delivery Date	**Mar 27, 2023**
Audit Methodology	**Static Analysis, Manual Review**
Total Isssues	**5**

[WP-H1] Insufficient Precision with `PIPS = 1_000_000` Leading to Financial Loss on User Withdrawal

High

Issue Description

Both the minting and burning of shares will convert the high precision shares into low precision (1e6) `proportion` before converting into asset amounts.

As a result, the user who mints may get more shares with fewer assets, and when burning shares, they may also get fewer assets.

PoC

Given:

- totalSupply = 1e8
- balanceOf token0 = 1e8
- balanceOf token1 = 1e8

1. Alice mints with `shares_ = 199`

proportion = 1

alice pays:

- amount0 = 1e2
- amount1 = 1e2

alice receives:

shares: 199

1. Alice borrows 1 share from someone and burns:

proportion = 2

alice receives:

- amount0 = 1e4
- amount1 = 1e4

https://github.com/ArrakisFinance/arrakis-modular/blob/b0a703fe14cfd4b172b40adaca5a9870707b0b7f/src/ArrakisMetaVaultPublic.sol#L75-L93

```
75  function burn(
76      uint256 shares_,
77      address receiver_
78  ) external returns (uint256 amount0, uint256 amount1) {
79      if (shares_ == 0) revert BurnZero();
80      uint256 supply = totalSupply();
81      if (shares_ > supply) revert BurnOverflow();
82
83      uint256 proportion = FullMath.mulDiv(shares_, PIPS, supply);
84
85      if (proportion == 0) revert CannotBurnProportionZero();
86      if (receiver_ == address(0)) revert AddressZero("Receiver");
87
88      _burn(msg.sender, shares_);
89
90      (amount0, amount1) = _withdraw(receiver_, proportion);
91
92      emit LogBurn(shares_, receiver_, amount0, amount1);
93  }
```

Due to the precision loss from rounding down in the calculation $proportion = \left\lfloor \frac{shares\_}{totalSupply()} \times PIPS \right\rfloor$, the computed $proportion$ may have an error close to 1 (representing $\frac{1}{PIPS}$) which is ***less*** than the actual $\frac{shares\_}{totalSupply()} \times PIPS$. This can lead to `receiver_` receiving fewer token0 and token1 at line 90.

Note: When $balanceOf(user) < \frac{1}{PIPS} \times totalSupply()$, the user cannot `burn()` . Since `shares_` must be less than or equal to `balanceOf(user)` (restricted at L88) and $proportion = \frac{shares\_}{totalSupply()} \times PIPS$ must be greater than 0 (restricted at L85), these two restrictions cannot be satisfied simultaneously when $balanceOf(user) < \frac{1}{PIPS} \times totalSupply()$.

https://github.com/ArrakisFinance/arrakis-modular/blob/b0a703fe14cfd4b172b40adaca5a9870707b0b7f/src/ArrakisMetaVaultPublic.sol#L48-L68

```solidity
function mint(
    uint256 shares_,
    address receiver_
) external payable returns (uint256 amount0, uint256 amount1) {
    if (shares_ == 0) revert MintZero();
    uint256 supply = totalSupply();

    uint256 proportion = FullMath.mulDiv(
        shares_, PIPS, supply > 0 ? supply : 1 ether
    );

    if (proportion == 0) revert CannotMintProportionZero();

    if (receiver_ == address(0)) revert AddressZero("Receiver");

    _mint(receiver_, shares_);

    (amount0, amount1) = _deposit(proportion);

    emit LogMint(shares_, receiver_, amount0, amount1);
}
```

https://github.com/ArrakisFinance/arrakis-modular/blob/b0a703fe14cfd4b172b40adaca5a9870707b0b7f/src/modules/ValantisSOTModulePublic.sol#L31-L97

```solidity
function deposit(
    address depositor_,
    uint256 proportion_
)
    external
    payable
    onlyMetaVault
    whenNotPaused
    nonReentrant
    returns (uint256 amount0, uint256 amount1)
{
    if (msg.value > 0) revert NoNativeToken();
    if (depositor_ == address(0)) revert AddressZero();
    if (proportion_ == 0) revert ProportionZero();

```

```
        // #region effects.

        {
            (uint256 _amt0, uint256 _amt1) = pool.getReserves();

            if (_amt0 == 0 && _amt1 == 0) {
                _amt0 = _init0;
                _amt1 = _init1;
            }

            amount0 = FullMath.mulDiv(proportion_, _amt0, PIPS);
            amount1 = FullMath.mulDiv(proportion_, _amt1, PIPS);
        }

        // #endregion effects.

        uint256 balance0 = token0.balanceOf(address(this));
        uint256 balance1 = token1.balanceOf(address(this));

        // #region interactions.

        // #region get the tokens from the depositor.

        token0.safeTransferFrom(depositor_, address(this), amount0);
        token1.safeTransferFrom(depositor_, address(this), amount1);

        // #endregion get the tokens from the depositor.

        // #region increase allowance to alm.

        token0.safeIncreaseAllowance(address(alm), amount0);
        token1.safeIncreaseAllowance(address(alm), amount1);

        // #endregion increase allowance to alm.

        alm.depositLiquidity(amount0, amount1, 0, 0);

        // #endregion interactions.

        // #region assertions.

        if (token0.balanceOf(address(this)) - balance0 > 0) {
            revert Deposit0();
```

```
    }
    if (token1.balanceOf(address(this)) - balance1 > 0) {
        revert Deposit1();
    }

    // #endregion assertions.

    emit LogDeposit(depositor_, proportion_, amount0, amount1);
}
```

Status

✓ Fixed

[WP-H2] Precision Loss in Rebasing Token Transfers Causing Unexpected Reverts in `ValantisSOTModule.withdraw()`

High

Issue Description

Inconsistencies between the `rabase token transfer amount` and `balanceOf()` (due to precision loss in the conversion between share amount and balance amount within the rabase token accounting) may cause `ValantisSOTModule.withdraw()` to unexpectedly revert.

The Valantis pool is designed to support rebasing tokens, which typically encounter a precision issue. For instance, attempting to transfer 100 might result in the receiver only seeing 99.99999 in their wallet. This discrepancy arises because the transfer involves shares, and the conversion from the nominal value to shares and back can lead to a loss in precision.

However, the `ValantisSOTModule#withdraw()` function explicitly checks the delta amounts of the balance, and these must exactly match the calculated expected amounts (L224-229). This strict requirement may lead to a revert if the balance delta does not precisely align with the calculated amounts, due to the aforementioned precision issue.

The impact is that users would be able to deposit, but they will almost always get a revert due to the precision issue when trying to withdraw.

https://github.com/ArrakisFinance/arrakis-modular/blob/33f270590fe7f0188d57620081069d85721de4fb/src/ArrakisMetaVaultPublic.sol#L70-L93

```
@@ 70,74 @@
function burn(
    uint256 shares_,
    address receiver_
) external returns (uint256 amount0, uint256 amount1) {
    if (shares_ == 0) revert BurnZero();
    uint256 supply = totalSupply();
    if (shares_ > supply) revert BurnOverflow();

    uint256 proportion = FullMath.mulDiv(shares_, PIPS, supply);

    if (proportion == 0) revert CannotBurnProportionZero();
86      if (receiver_ == address(0)) revert AddressZero("Receiver");
87
88      _burn(msg.sender, shares_);
89
90      (amount0, amount1) = _withdraw(receiver_, proportion);
91
92      emit LogBurn(shares_, receiver_, amount0, amount1);
93  }
```

https://github.com/ArrakisFinance/arrakis-modular/blob/33f270590fe7f0188d57620081069d85721de4fb/src/abstracts/ArrakisMetaVault.sol#L244-L249

```
244  function _withdraw(
245      address receiver_,
246      uint256 proportion_
247  ) internal returns (uint256 amount0, uint256 amount1) {
248      (amount0, amount1) = module.withdraw(receiver_, proportion_);
249  }
```

https://github.com/ArrakisFinance/arrakis-modular/blob/33f270590fe7f0188d57620081069d85721de4fb/src/abstracts/ValantisSOTModule.sol#L173-L234

```
173  /// @notice function used by metaVault to withdraw tokens from the strategy.
174  /// @param receiver_ address that will receive tokens.
175  /// @param proportion_ number of share needed to be withdrawn.
176  /// @return amount0 amount of token0 withdrawn.
177  /// @return amount1 amount of token1 withdrawn.
178  function withdraw(
179      address receiver_,
180      uint256 proportion_
181  )
182      external
183      onlyMetaVault
184      nonReentrant
185      returns (uint256 amount0, uint256 amount1)
186  {
187      // #region checks.
188
189      if (receiver_ == address(0)) revert AddressZero();
```

```
    if (proportion_ == 0) revert ProportionZero();
    if (proportion_ > PIPS) revert ProportionGtPIPS();

    // #endregion checks.

    // #region effects.

    {
        (uint256 _amt0, uint256 _amt1) = pool.getReserves();

        amount0 = FullMath.mulDiv(proportion_, _amt0, PIPS);
        amount1 = FullMath.mulDiv(proportion_, _amt1, PIPS);
    }

    if (amount0 == 0 && amount1 == 0) revert AmountsZeros();

    // #endregion effects.

    // NOTE:  check with Ed for rebase tokens.

    uint256 balance0 = token0.balanceOf(receiver_);
    uint256 balance1 = token1.balanceOf(receiver_);

    // #region interactions.

    alm.withdrawLiquidity(amount0, amount1, receiver_, 0, 0);

    // #endregion interactions.

    uint256 _actual0 = token0.balanceOf(receiver_) - balance0;
    uint256 _actual1 = token1.balanceOf(receiver_) - balance1;

    // #region assertions.

    if (_actual0 != amount0) {
        revert Actual0DifferentExpected(_actual0, amount0);
    }
    if (_actual1 != amount1) {
        revert Actual1DifferentExpected(_actual1, amount1);
    }

    // #endregion assertions.

    emit LogWithdraw(receiver_, proportion_, amount0, amount1);
}
```

Status

✓ **Fixed**

[WP-M3] Not setting dead shares may expose the system to first minter PPS inflation attacks.

Medium

Issue Description

This is a classic attack vector that affects nearly every vault system involving share issuance. In such scenarios, the initial minter can intentionally create a condition where the total supply of shares is minimal and manipulate the share price to an excessively high value. Consequently, future shareholders will receive a reduced number of shares due to precision loss when converting deposited assets into shares.

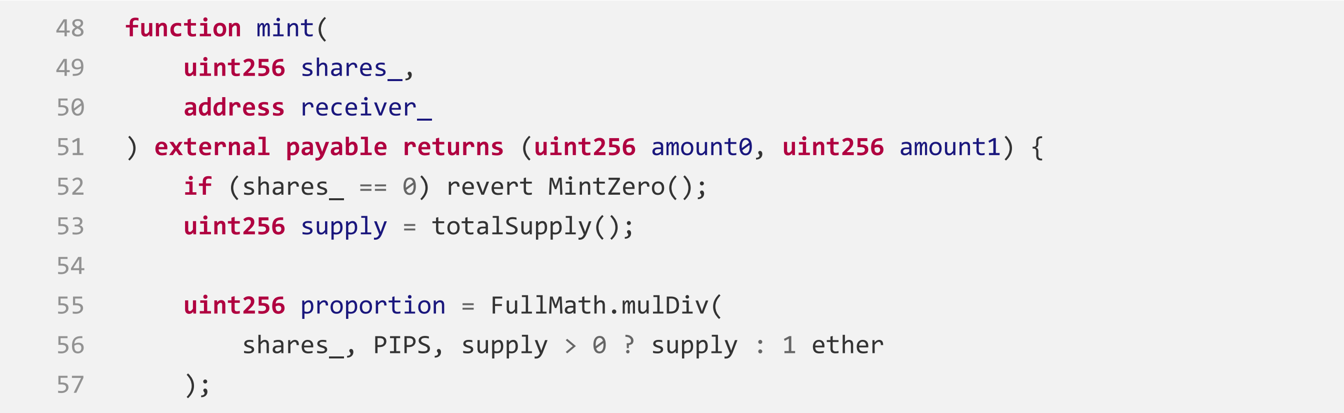
A common solution is to mandate a specific initial mint amount and send a substantial number of shares to a dead address as a permanent reserve. This approach significantly increases the cost of manipulating share prices.

In the current implementation, there is no such permanent reserve, even though the first minter must at least mint 1e12 shares, they can burn two times, each time burning (1e6-1)/1e6 of the total supply, then the totalSupply will be reduced to 1 wei.

The attacker can substantially send funds to the module directly to inflate the PPS.

Note: The impact of this issue is limited by [WP-H1], however, it may become more impactful once changes are made to [WP-H1].

https://github.com/ArrakisFinance/arrakis-modular/blob/33f270590fe7f0188d57620081069d85721de4fb/src/ArrakisMetaVaultPublic.sol#L48-L68



```
function mint(
    uint256 shares_,
    address receiver_
) external payable returns (uint256 amount0, uint256 amount1) {
    if (shares_ == 0) revert MintZero();
    uint256 supply = totalSupply();

    uint256 proportion = FullMath.mulDiv(
        shares_, PIPS, supply > 0 ? supply : 1 ether
    );
```

```

        if (proportion == 0) revert CannotMintProportionZero();

        if (receiver_ == address(0)) revert AddressZero("Receiver");

        _mint(receiver_, shares_);

        (amount0, amount1) = _deposit(proportion);

        emit LogMint(shares_, receiver_, amount0, amount1);
    }
```

https://github.com/ArrakisFinance/arrakis-modular/blob/b0a703fe14cfd4b172b40adaca5a9870707b0b7f/src/ArrakisMetaVaultPublic.sol#L75-L93

```
    function burn(
        uint256 shares_,
        address receiver_
    ) external returns (uint256 amount0, uint256 amount1) {
        if (shares_ == 0) revert BurnZero();
        uint256 supply = totalSupply();
        if (shares_ > supply) revert BurnOverflow();

        uint256 proportion = FullMath.mulDiv(shares_, PIPS, supply);

        if (proportion == 0) revert CannotBurnProportionZero();
        if (receiver_ == address(0)) revert AddressZero("Receiver");

        _burn(msg.sender, shares_);

        (amount0, amount1) = _withdraw(receiver_, proportion);

        emit LogBurn(shares_, receiver_, amount0, amount1);
    }
```

Status

✓ Fixed

[WP-M4] `RouterSwapExecutor#swap()` should not pull tokens from `router` .

Medium

Issue Description

`router` has already pushed the tokens to `swapper` , and since there is no allowance, the `safeTransferFrom()` call will revert.

https://github.com/ArrakisFinance/arrakis-modular/blob/b0a703fe14cfd4b172b40adaca5a9870707b0b7f/src/ArrakisPublicVaultRouter.sol#L933-L1006

```
    function _swapAndAddLiquidity(
@@ 934,947 @@
    {
        uint256 valueToSend;
        if (params_.swapData.zeroForOne) {
            if (token0_ != nativeToken) {
                IERC20(token0_).safeTransfer(
                    address(swapper), params_.swapData.amountInSwap
                );
            } else {
                valueToSend = params_.swapData.amountInSwap;
            }
        } else {
            if (token1_ != nativeToken) {
                IERC20(token1_).safeTransfer(
                    address(swapper), params_.swapData.amountInSwap
                );
            } else {
                valueToSend = params_.swapData.amountInSwap;
            }
        }

        (amount0Diff, amount1Diff) =
            swapper.swap{value: valueToSend}(params_);

@@ 971,1005 @@
    }
```

https://github.com/ArrakisFinance/arrakis-modular/blob/b0a703fe14cfd4b172b40adaca5a9870707b0b7f/src/RouterSwapExecutor.sol#L41-L127

```
    function swap(SwapAndAddData memory params_)
        external
        payable
        onlyRouter
        returns (uint256 amount0Diff, uint256 amount1Diff)
    {
        address token0 =
            IArrakisMetaVault(params_.addData.vault).token0();
        address token1 =
            IArrakisMetaVault(params_.addData.vault).token1();
        uint256 balanceBefore;
        uint256 valueToSend;
        if (params_.swapData.zeroForOne) {
            if (token0 != nativeToken) {
                IERC20(token0).safeTransferFrom(
                    router,
                    address(this),
                    params_.swapData.amountInSwap
                );
                balanceBefore =
                    IERC20(token0).balanceOf(address(this));
                IERC20(token0).safeIncreaseAllowance(
                    params_.swapData.swapRouter,
                    params_.swapData.amountInSwap
                );
            } else {
                balanceBefore = address(this).balance;
                valueToSend = params_.swapData.amountInSwap;
            }
        } else {
            if (token1 != nativeToken) {
                IERC20(token1).safeTransferFrom(
                    router,
                    address(this),
                    params_.swapData.amountInSwap
                );
                balanceBefore =
                    IERC20(token1).balanceOf(address(this));
                IERC20(token1).safeIncreaseAllowance(
                    params_.swapData.swapRouter,
                    params_.swapData.amountInSwap
                );
            } else {
                balanceBefore = address(this).balance;
                valueToSend = params_.swapData.amountInSwap;
            }
        }
        (bool success,) = params_.swapData.swapRouter.call{
            value: valueToSend
        }(params_.swapData.swapPayload);
        if (!success) revert SwapCallFailed();

@@ 93,125 @@
    }
}
```

Status

✓ Fixed

[WP-N5] Consider `_disableInitializers()` in `constructor()`

Issue Description

It is a best practice to call `_disableInitializers()` in the constructor function of an upgradeable contract.

See: https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#initializing_the_implementation_contract

https://github.com/ArrakisFinance/arrakis-modular/blob/33f270590fe7f0188d57620081069d85721de4fb/src/ArrakisStandardManager.sol#L107-L125

```
    constructor(
        uint256 defaultFeePIPS_,
        address nativeToken_,
        uint8 nativeTokenDecimals_,
        address guardian_
    ) {
        if (nativeToken_ == address(0)) revert AddressZero();
        if (nativeTokenDecimals_ == 0) {
            revert NativeTokenDecimalsZero();
        }
        if (guardian_ == address(0)) revert AddressZero();
        /// @dev we are not checking if the default fee pips is not zero, to have
        /// the option to set 0 as default fee pips.

        defaultFeePIPS = defaultFeePIPS_;
        nativeToken = nativeToken_;
        nativeTokenDecimals = nativeTokenDecimals_;
        _guardian = guardian_;
    }
```

https://github.com/ArrakisFinance/arrakis-modular/blob/33f270590fe7f0188d57620081069d85721de4fb/src/modules/ValantisSOTModulePublic.sol#L22

```
    constructor(address guardian_) ValantisModule(guardian_) {}
```

https://github.com/ArrakisFinance/arrakis-modular/blob/33f270590fe7f0188d57620081069d85721de4fb/src/modules/ValantisSOTModulePrivate.sol#L19

```
    constructor(address guardian_) ValantisModule(guardian_) {}
```

Recommendation

Consider add `_disableInitializers()` into `constructor()` of upgradeable contracts.

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.