STATE MIND

Arrakis v2 periphery

06-04-2023 - 31-08-2023

Table of contents

1. Project Brief

Title	Description
Client	Arrakis
Project name	Arrakis v2 periphery
Timeline	06-04-2023 - 31-08-2023
Initial commit	cab630396506aad825d838f98d60d287ed49c0b9
Final commit	fdf8899b1feb8e5708695a67ae609b137b752933

Short Overview

ArrakisV2Router receives the approval from the users, transfers funds from users to itself, validate input data, wrap/unwrap eth, deposit/withdraw, stake/unstake, returns funds to users.

RouterSwapExecutor is responsible for executing swap payloads (prepared off-chain) passed to Router's swapAndAddLiquidity methods. This separation of contracts allows swap payloads to tap "arbitrary" liquidity sources and still be safe.

ArrakisV2GaugeFactory is the entry-point for deploying GaugeV4 for Arrakis vaults.

ArrakisV2StaticDeployer is a contract for auto-deploying vaults with static manager and renounced owner.

ArrakisV2Staticmanager is a contract for static managing ArrakisV2 vaults, using several functions.

Project Scope

The audit covered the following files:

ArrakisV2StaticManagerStorage.sol	ArrakisV2GaugeFactoryStorage.sol	ArrakisV2RouterStorage.sol
ArrakisV2Router.sol	ArrakisV2GaugeFactory.sol	Twap.sol
RouterSwapResolver.sol	SArrakisV2Router.sol	SPermit2.sol
SStaticManager.sol	SArrakisV2StaticDeployer.sol	RouterSwapExecutor.sol
ArrakisV2GaugeBeacon.sol	ArrakisV2StaticDeployer.sol	ArrakisV2StaticManager.sol

2. Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
🟥 Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds to be transferred to any party.
🟧 High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
🟨 Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss of funds.
🟩 Informational	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

3. Summary of findings

Severity	# of Findings
Critical	0 (0 fixed, 0 acknowledged)
High	1 (1 fixed, 0 acknowledged)
Medium	6 (6 fixed, 0 acknowledged)
Informational	37 (9 fixed, 28 acknowledged)
Total	44 (16 fixed, 28 acknowledged)

4. Conclusion

During the audit of Arrakis v2 periphery codebase, 44 issues were found in total:

- 1 high severity issue (1 fixed)
- 6 medium severity issues (6 fixed)
- 37 informational severity issues (9 fixed, 28 acknowledged)

The final reviewed commit is fdf8899b1feb8e5708695a67ae609b137b752933.

Contracts are deployed on ethereum, arbitrum, base, bsc, optimism, polygon networks under the same addresses.

Deployment

File name	Contract deployed on mainnet
ArrakisV2Router (proxy)	0x6aC8Bab8B775a03b8B72B2940251432442f61B94
ArrakisV2Router (implementation)	0xFe3D837317d420E9C40C30DCb49892aD9EF15e3d
DefaultProxyAdmin	0xdF4433A2b8850C49c2ef2cfF6447637002D8D8DD

RouterSwapResolver	0x2ce9E840b69a86C965f4D732e39e389fe69Fda51

5. Findings report

HIGH-01	DoS of RouterSwapExecutor with non zero approvals	Fixed at 3eaf25

Description

In the function **RouterSwapExecutor.swap()**, **swapRouter** and **swapPayload** are user chosen.

```
if (swapAndAddData_.swapData.zeroForOne) {
    balanceBefore = token0.balanceOf(address(this));
    token0.safeApprove(
        swapAndAddData_.swapData.swapRouter,
        swapAndAddData_.swapData.amountInSwap
    );
} else {
    balanceBefore = token1.balanceOf(address(this));
    token1.safeApprove(
        swapAndAddData_.swapData.swapRouter,
        swapAndAddData_.swapData.amountInSwap
    );
}
(bool success, ) = swapAndAddData_.swapData.swapRouter.call(
    swapAndAddData_.swapData.swapPayload
);
```

An attacker can choose **swapRouter** as any token different than **token0** and **token1** and approve non zero value for routers. Since the token chosen by the attacker is different than **token0** and **token1**, its allowance will not be reset to **0**. Then when a user tries to swap **safeApprove()** will revert since there is already a non zero approval.

```
function safeApprove(
    IERC20 token,
    address spender,
    uint256 value
) internal {
    // safeApprove should only be called when setting an initial allowance,
    // or when resetting it to zero. To increase and decrease it, use
    // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
    require(
        (value == 0) || (token.allowance(address(this), spender) == 0),
        "SafeERC20: approve from non-zero to non-zero allowance"
    );
    _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
}
```

This way an attacker can DoS every call to **swapAndAddLiquidity()** by frontrunning.

Recommendation

We recommend to use only whitelisted routers as **swapRouter**.

MEDIUM-01	Malicious vault can get a huge allowance for any token from router	Fixed at 3eaf25

Description

User can add liquidity to vault via **addLiquidity**, **swapAndAddLiquidity**, **addLiquidityPermit2**, **swapAndAddLiquidityPermit2** functions.
Attack **addLiquidity** scenario:

1. Attacker passes malicious vault to **addLiquidity** function.
   Malicious vault can manipulate **amount0**, **amount1**, **mintAmount** for calculation in router and resolver, because **totalSupply**, **init0**, **init1** getting from vault.
   So, attacker can make any **amount0**, **amount1**, **mintAmount**.
2. **ArrakisV2Router** call **IArrakisV2(vault_).token0()**, **IArrakisV2(vault_).token0()** functions for getting **token0**, **token1** addresses and transfer token from user to router.
   So, malicious vault can return malicious **token0**, **token1** and don't make transfer or transfer fake token.
   ArrakisV2Router have a lot of re-calls **IArrakisV2(vault_).token0()**, **IArrakisV2(vault_).token1()** in other places.

```
if (amount0 > 0 && (msg.value == 0 || !isToken0Weth)) {
    IERC20(IArrakisV2(params_.vault).token0()).safeTransferFrom(
//              ^-- token0 is requested from the router
//  malicious vault can return fake token0
        msg.sender, // <-- from attacker
        address(this), // <-- to router
        amount0
    );
}
if (amount1 > 0 && (msg.value == 0 || isToken0Weth)) {
    IERC20(IArrakisV2(params_.vault).token1()).safeTransferFrom(
//              ^-- token1 is requested from the router
//  malicious vault can return fake token1
        msg.sender, // <-- from attacker
        address(this), // <-- to router
        amount1
    );
}
```

3. Than, **ArrakisV2Router** again call **IArrakisV2(vault_).token0()**, **IArrakisV2(vault_).token1()** functions for getting **token0**, **token1** addresses and increase allowance in **_addLiquidity** function.

```
IERC20(IArrakisV2(vault_).token0()).safeIncreaseAllowance(
//         ^-- token0 is requested from the router
// there is no relation between the token transferred user -> router
// and the token approved router -> vault
    vault_,
    amount0In_
);
IERC20(IArrakisV2(vault_).token1()).safeIncreaseAllowance(
//         ^-- token1 is requested from the router
// there is no relation between the token transferred user -> router
// and the token approved router -> vault
    vault_,
    amount1In_
);
```

4. Finally, after call **IArrakisV2(vault_).mint(mintAmount_, receiver_)** malicious vault can try to steal a token if it is on the balance of the router. Even if router ever gets some tokens which are transferred by mistake or after work with an incorrectly configured vault, attacked can make approve and steal token. We haven't found any cases when fair vault unspent amounts, but we pay attention to approve.

Recommendation

All recommendations actual for **addLiquidity**, **swapAndAddLiquidity**, **addLiquidityPermit2**, **swapAndAddLiquidityPermit2** functions.

We recommend

- adding check and zeroing allowance after calling **mint** in **_addLiquidity** function, if vault doesn't spend all allowance amounts.
- using **token0**, **token1** address after first calling from vault. It's actual for all places in router contract. So, router will definitely transfer **token0**, **token1** from attacked.
- adding check and refund unspent tokens to **msg.sender** if it has.

MEDIUM-02	Growth factor calculation includes manager fees	Fixed at 3eaf25

Description

When calculating **totalUnderlyingWithFees**, the **Underlying** library subtracts admin fees from **underlying.fee0** and **underlying.fee1** Underlying.sol#L476-L493:

```
(uint256 fee0After, uint256 fee1After) = subtractAdminFees(
    fee0,
    fee1,
    arrakisV2.managerFeeBPS()
);

amount0 +=
    fee0After +
    IERC20(underlyingPayload_.token0).balanceOf(
        underlyingPayload_.self
    ) -
    arrakisV2.managerBalance0();
amount1 +=
    fee1After +
    IERC20(underlyingPayload_.token1).balanceOf(
        underlyingPayload_.self
    ) -
    arrakisV2.managerBalance1();
```

Thus,

```
underlying.amount0 = amount0 + underlying.leftOver0 + underlying.fee0 - managerFee0
```

But this is not done in the function **ArrakisV2StaticManager.compoundFees()**, **underlying.fee0** and **underlying.fee1** include fees of the manager.

```
uint256 liquidity0 = underlying.amount0 -
    (underlying.leftOver0 + underlying.fee0);
uint256 liquidity1 = underlying.amount1 -
    (underlying.leftOver1 + underlying.fee1);
uint256 proportion0 = liquidity0 > 0
    ? FullMath.mulDiv(
        underlying.leftOver0 + underlying.fee0,
        hundredPercent,
        liquidity0
    )
    : type(uint256).max;
```

Which means **proportion0**, **proportion1** and consequently **growthFactor** will be higher than they are supposed to be. When the rebalance withdraws tokens from the pool to the vault, it will lock manager fees and this could lead to mints reverting due to not having enough tokens.

Recommendation

We recommend to subtract manager fees from **underlying.fee0** and **underlying.fee1**.

MEDIUM-03 | Liquidity to mint can be incorrect due to a lack of precision | Fixed at 3eaf25

Description

In _getInits method of **ArrakisV2StaticDeployer** liquidity can be too small to make **in0** or **in1** greater than zero due to integer division in LiquidityAmounts.getAmountsForLiquidity but it shouldn't be zero in reality.

```
(uint256 in0, uint256 in1) = LiquidityAmounts
    .getAmountsForLiquidity(
        sqrtPriceX96,
        TickMath.getSqrtRatioAtTick(positions_[i].range.lowerTick),
        TickMath.getSqrtRatioAtTick(positions_[i].range.upperTick),
        positions_[i].liquidity // <-- consider it is small enough
    );
if (in0 > 0) init0 += in0 + 1;
if (in1 > 0) init1 += in1 + 1;
//   ^-- in0/in1 is zero cause of integer division.
//       However, it should be taken into account.
```

Thus, required **init0** and **init1** will be calculated incorrectly.

Recommendation

We recommend reconsidering the logic of **getAmountsForLiquidity** method. Instead of actual amounts return amounts with precision.

MEDIUM-04 | Initial amounts can be manipulated in Static Deployer | Fixed at 3eaf25

Description

In deployStaticVault method of **ArrakisV2StaticDeployer** contract, there is no check that **init0** and **init1** are not exceeding max amounts. Thus, it is possible to manipulate the price in uni and set the wrong **init0** and **init1**. And then cause ownership is renounced it is impossible to change **init0** and **init1** manually.

```
(uint256 init0, uint256 init1) = _getInits(
    params_.positions,
    params_.token0,
    params_.token1
);

require(
    init0 >= params_.minDeposit0 && init1 >= params_.minDeposit1,
    "slippage"
);
//    ^-- Only check of the the minimum values
```

Recommendation

We recommend adding a check that **init1** and **init0** do not exceed max amounts.

MEDIUM-05	Insufficient checks of vault parameters for setting it as static	Fixed at 3eaf25

Description

Function **ArrakisV2StaticManager.setStaticVault()** is adding vault to mapping of vaults and access to this function is granted to everyone. But vault address can't be deleted from this mapping. Checks, that **owner()** and **manager()** of this vault are correct doesn't working with malicious ones.

```
    function setStaticVault(SetStaticVault calldata params_) external {
        ...
        // only vault owner can call
        require(msg.sender == IArrakisV2(params_.vault).owner(), "NO");
        // must be manager
        require(address(this) == IArrakisV2(params_.vault).manager(), "NM");
        // set fee take rate
        ...
    }
```

In addition to this, **compoundFees** function is an external one, but **owner()** of static vault can't get fees from it.

```
    function compoundFees(IArrakisV2 vault_) external whenNotPaused {
        ...
    }

    function withdrawAndCollectFees(
        ...
    ) external onlyOwner {
        ...
    }
```

Recommendation

We recommend specifying in more detail what functionality you want to create for the static manager contract, if you want to give an external address an access. If not, we recommend setting access modifier on **setStaticVault()** function.

MEDIUM-06	Incorrect swap amount calculatins	Fixed at 3eaf25

Description

In **RouterSwapResolver.calculateSwapAmount()** there's a logic to set **zeroForOne** flag and calculate **swapAmount** based on a comparison of the quantity two tokens, but this comparison doesn't consider price and decimals at all. So, this comparison is incorrect, if after **resolver.getMintAmounts()** amount of any token wouldn't be equal to zero.

Recommendation

We recommend replacing

```
if (amount0Left > amount1Left) {
    ...
} else if (amount1Left > amount0Left) {
    ...
}
```

with

```
uint256 value0To1Left = amount0Left * factor0 * price18Decimals / 1 ether;
uint256 value1To0Left = amount1Left * factor1;

if (value0To1Left > value1To0Left) {
    ...
} else if (value0To1Left < value1To0Left) {
    ...
}
```

INFORMATIONAL-01	Check managerFeeBPS's value	Fixed at 3eaf25

Description

A **managerFeeBPS** variable doesn't have a check for its value.

Recommendation

We recommend adding a check that the **managerFeeBPS** is not more than 10000.

INFORMATIONAL-02	SafeApprove deprecated	Fixed at 3eaf25

Description

safeApprove method is deprecated.

Recommendation

We recommend replacing **safeApprove** with **safeIncreaseAllowance** and **safeDecreaseAllowance**:

- ArrakisV2StaticDeployer.sol#L94
- ArrakisV2StaticDeployer.sol#L95
- RouterSwapExecutor.sol#L41
- RouterSwapExecutor.sol#L47
- RouterSwapExecutor.sol#L59
- RouterSwapExecutor.sol#L61

INFORMATIONAL-03	Improper rounding	Fixed at 3eaf25

Description

The **ArrakisV2StaticDeployer** contract has a function **_getInits** that has an improper rounding up.

Recommendation

We recommend using **getAmountsForDelta** from Core contracts instead of **getAmountsForLiquidity**.

INFORMATIONAL-04	Conversion edge case	Fixed at 7668f7

Description

In the **ArrakisV2StaticDeployer** contract, there is a function called **_getInits**, which includes an overflow check. However, this check does not consider the edge case "**positions_[i].liquidity == type(uint128).max / 2**" to be true, even though it is.

Recommendation

We recommend modifying this check to the following:

```
positions_[i].liquidity <= type(uint128).max / 2
```

INFORMATIONAL-05	Possible redundant event logs	Fixed at 7668f7

Description

The **ArrakisV2RouterStorage** contract has the functions **whitelist** and **blacklist**, which add/remove addresses to/from EnumerableSet. However, these operations are not wrapped in **require** statements to check if they have succeeded or not. As a result, the event logs may contain redundant addresses that were not processed.

Recommendation

We recommend adding **require** statements around the **add** and **remove** operations.

INFORMATIONAL-06	Deprecated and unnecessary safeApproves	Fixed at 7668f7

Description

In the **ArrakisV2Router** contract, there is a function called **_addLiquidity** that performs approvals with zero values at the end. However, this implementation uses the deprecated **safeApprove** function, and these approvals are unnecessary due to the previous checks.

Recommendation

We recommend removing these **safeApprove** statements.

INFORMATIONAL-07	Missing of zeroes checks	Fixed at 7668f7

Description

When we deploy a contract or communicate with contract, we can pass parameters to contract. So, contract responsibility is validation input params for prevention incorrect state or unexpected behavior.
We found several missing zero address checks:

- **ArrakisV2StaticDeployer constructor** for params **uniswapFactory_**, **arrakisFactory_**, **gaugeFactory**, **staticManager**
- **RouterSwapResolver constructor** for params **helper_**, **resolver_**

- **RouterSwapExecutor constructor** for params **_router**
- **ArrakisV2GaugeFactoryStorage constructor** for params **gaugeBeacon_**
- **ArrakisV2RouterStorage constructor** for params **weth_**, **resolver_**, **permit2_**
- **ArrakisV2StaticManagerStorage constructor** for params **helper_**
- **ArrakisV2StaticManagerStorage initialize** function for params **owner_**
- **ArrakisV2RouterStorage initialize** for params **owner_**
- **ArrakisV2RouterStorage updateSwapExecutor** for params **swapper_**

Recommendation

We recommend adding zero address checks.

INFORMATIONAL-08	Incorrect of zeroes checks	Fixed at 7668f7

Description

ArrakisV2GaugeFactoryStorage has check of input addresses for **initialize**, **setDefaultReward** functions. However, if one **param != address(0)** checks will be pass, because check use ||(OR) operator

```
function initialize(
    address owner_,
    address rewardToken_,
    address ve_,
    address veBoost_
) external initializer {
    require(
        owner_ != address(0) II <-- if owner_ true check is true
            rewardToken_ != address(0) II <-- if rewardToken_ true check is true
            ve_ != address(0) II <-- if ve_ true check is true
            veBoost_ != address(0), <-- if veBoost_ true check is true
        "address zero"
    );
    ...
}
```

```
function setDefaultReward(
    address rewardToken_,
    address ve_,
    address veBoost_
) external onlyOwner {
    require(
        rewardToken_ != address(0) II <-- if rewardToken_ true check is true
            ve_ != address(0) II <-- if ve_ true check is true
            veBoost_ != address(0), <-- if veBoost_ true check is true
        "address zero"
    );
    ...
}
```

Recommendation

We recommend replacing ||(OR) to **&&**(AND) operator.

INFORMATIONAL-09	ArrakisV2GaugeFactory restricts distributor	Acknowledged

Description

ArrakisV2GaugeFactory allow deploy **Gauge** via **deployGauge** function.
However, factory set factory owner as distributor for **defaultRewardToken**. It can be problem, if deployer want to create Gauge with **defaultRewardToken** and provide rewards in this token.

```
bytes memory data = abi.encodeWithSelector(
    IGauge.initialize.selector,
    stakingToken_,
    address(this),
    defaultRewardToken,
    ve,
    veBoost,
    owner() // <-- owner is hardcoded to distributor
);
```

Recommendation

We recommend providing the ability for deployer to configure distributor for **defaultRewardToken** function at the moment of deployment.

INFORMATIONAL-10	Incorrect name of the interface	Fixed at 3eaf25

Description

In **ArrakisV2GaugeFactoryStorage.construtor() arrakisGaugeBeacon** is declared as **IArrakisV2Beacon**, but it should be declared as **IArrakisV2GaugeBeacon**.

Recommendation

We recommend replacing the name of interface with correct one.

INFORMATIONAL-11	Revert on overflow at TWAP	Acknowledged

Description

The **TWap** library has functions **getPrice0** and **getPrice1** that revert if the **getSqrtTwapX96** would return **sqrtPriceX96** >= 2^128 because of getting a square of it.

Recommendation

We recommend fixing it getting **this function** as a reference.

INFORMATIONAL-12	Remove excess contract usability	Acknowledged

Description

There are different interactions with gauge contracts, but there are no restrictions if these gauges have any connections with Arrakis' ecosystem.

Recommendation

We recommend adding a check to see if the input gauge's addresses belong to the ecosystem's ones:

- ArrakisV2GaugeFactory.sol::addGaugeReward
- ArrakisV2GaugeFactory.sol::setGaugeRewardDistributor
- ArrakisV2GaugeFactoryStorage.sol::getProxyAdmin
- ArrakisV2GaugeFactoryStorage.sol::getProxyImplementation

INFORMATIONAL-13	Indexed Event Parameters	Acknowledged

Description

Currently, none of the events has **indexed** parameters. Making some parameters **indexed** could be useful later for filtering contracts' logs.

Recommendation

We recommend making these parameters **indexed**:

- CreateStaticVault:[vault, gauge, caller]
- InitFactory:[owner]
- GaugeCreated:[deployer, gauge]
- DefaultRewardSet:[token, ve, veBoost]
- Compound:[vault, caller]

INFORMATIONAL-14	No check for permitted tokens	Acknowledged

Description

The **ArrakisV2Router** contract contains functions that utilize Permit2 transfers and other interactions with tokens, but it does not verify whether these tokens are the same. Specifically, it does not ensure that the permitted tokens match the vault's tokens. Typically, if improper tokens are provided in the permit structure, the transaction will revert. However, if the router has the required tokens, the transaction could potentially lock the user's funds if it succeeds.

Recommendation

We recommend checking if they are the same:

- L275
- L302
- L375
- L404

INFORMATIONAL-15	twapDuration's inconsistent type	Acknowledged

Description

The current solution uses uint24 type for **twapDuration** variable. However, the Uniswap's **observe** function gets a uint32[] array as an input.

Recommendation

We recommend replacing uint24 type with uint32:

- ArrakisV2StaticManager.sol#L155
- Twap.sol#L14
- Twap.sol#L31
- Twap.sol#L47
- Twap.sol#L63
- Twap.sol#L81

- SStaticManager.sol#L7

INFORMATIONAL-16	twapDeviation's inconsistent type	Acknowledged

Description

The current solution uses int24 type for **twapDeviation** variable. However, it cannot be less than zero by definition.

Recommendation

We recommend replacing int24 type with uint24:

- SStaticManager.sol#L6
- Twap.sol#L82
- ArrakisV2StaticManager.sol#L156

INFORMATIONAL-17	gauges() reverts when numGauges() is zero	Acknowledged

Description

The **ArrakisV2GaugeFactory** contract has a function **gauges** that reverts when **numGauges()** is zero with any input variables.

Recommendation

We recommend adding a check when **numGauges()** is zero and returning an empty array.

INFORMATIONAL-18	Zero value check	Acknowledged

Description

The **Twap** library has a function **getTwap** that has **twapDuration_** as an input and will revert for **twapDuration_** = 0 because of division by zero.

Recommendation

We recommend adding a check to ensure that **twapDuration_** is not zero before performing the division.

INFORMATIONAL-19	calculateSwapAmount revert if tokens have decimals > 18	Acknowledged

Description

ERC-20 standard allows set any decimals for token, also UniswapV3 work with tokens have any decimals. The only implicit constraint for many protocols is correct math with tokens which should fit to uint256, so most parts of tokens have **decimals <= 18**.

However, some tokens in current or future can have **decimals > 18** and we can't expect otherwise.

RouterSwapResolver calculateSwapAmount hardcoded 18 decimals and denominator size to bytecode and will always revert(without a message) for tokens with big decimals.

```
uint256 factor0 = 10 **
    (18 - IERC20Metadata(address(vault.token0())).decimals());
//  ^-- revert without message if token0.decimals() > 18
uint256 factor1 = 10 **
    (18 - IERC20Metadata(address(vault.token1())).decimals());
//  ^-- revert without message if token1.decimals() > 18
```

Recommendation

We recommend

- adding function for calculation **swapAmount** with custom denominator and decimals for tokens have **decimals > 18**.
- adding a revert message or custom error for **calculateSwapAmount** function if tokens have incorrect **decimals**

INFORMATIONAL-20	Redundant input data for swap function	Acknowledged

Description

swap function receives **SwapAndAddData** parameter, but used only **SwapData** and **vault** from **AddLiquidityData**. Also, **swap** function shouldn't know about **AddLiquidityData**, and when we pass less data we save some gas.

Recommendation

We recommend reducing input params like **SwapData**, **token0**, **token1** for **swap** function.

INFORMATIONAL-21	Contract size and deploy cost optimization	Acknowledged

Description

Long string in contract to increase contract bytecode and deploying cost. We found 2 ways in require statement:

- Long message string
- Short message string

It is actual for many require statements for contracts of v2-periphery.

Recommendation

We recommend replacing long message strings to short message strings or use custom error.

INFORMATIONAL-22	Gas optimization for calldata params	Acknowledged

Description

Solidity provides several data locations, like **storage**, **memory**, **calldata**. Operation reading, writing has different gas costs. Generally, all references data types must declare data location, when it uses as function parameters. Also, assignments between from **calldata** to **memory** always create a separate copy. When we declared input params with **memory** can implicitly create new copy and spent more gas than **calldata**. So, if contract doesn't modify input parameters, we prefer use **calldata**.

We found several places using **memory** for input params:

- **IArrakisV2Router addLiquidity**
- **IArrakisV2Router removeLiquidity**
- **IArrakisV2Router swapAndAddLiquidity**
- **IArrakisV2Router addLiquidityPermit2**
- **IArrakisV2Router removeLiquidityPermit2**

- **IArrakisV2Router swapAndAddLiquidityPermit2**
- **ArrakisV2Router addLiquidity**
- **ArrakisV2Router removeLiquidity**
- **ArrakisV2Router swapAndAddLiquidity**
- **ArrakisV2Router addLiquidityPermit2**
- **ArrakisV2Router removeLiquidityPermit2**
- **ArrakisV2Router swapAndAddLiquidityPermit2**
- **ArrakisV2Router _swapAndAddLiquidity**
- **ArrakisV2Router _removeLiquidity**
- **IRouterSwapExecutor swap**
- **RouterSwapExecutor swap**

Recommendation

We recommend replacing data location from **memory** to **calldata**

INFORMATIONAL-23	Unsafe call for unknown dummy Gauge	Acknowledged

Description

ArrakisV2GaugeFactory can deploy **Gauge** via **deployGauge** function and tracks deployed **Gauge** via **_gauges** variable. So, factory should interact with only known gauges. However, factory call Gauge in **setGaugeRewardDistributor**, **addGaugeReward** functions without checking in **_gauges**. If the owner call **setGaugeRewardDistributor**, **addGaugeReward** with unknown **Gauge(admin = factory)** function executed without error.

```
EnumerableSet.AddressSet internal _gauges; // <-- track all deployed gauges

function addGaugeReward(
    IGauge gauge_,
    address token_,
    address distributor_
) external onlyOwner {
    uint256 len = gauge_.reward_count(); // <-- doesn't check for known
    for (uint256 i; i < len; i++) {
        require(gauge_.reward_tokens(i) != token_, "AE");
    }
    gauge_.add_reward(token_, distributor_); // <-- doesn't check for known
}

function setGaugeRewardDistributor(
    IGauge gauge_,
    address token_,
    address distributor_
) external onlyOwner {
    gauge_.set_reward_distributor(token_, distributor_); // <-- doesn't check for known
}
```

Recommendation

We recommend adding checks on known Gauges for **setGaugeRewardDistributor**, **addGaugeReward** functions.

```
require(_gauges.contrains(address(gauge_), "UG");
```

INFORMATIONAL-24	Inconsistent logic adds reward for Gauges	Acknowledged

Description

ArrakisV2GaugeFactory allow deploy **Gauge** via **deployGauge** function.

Also, deployer can add single **rewardToken_** and **rewardDistributor_** when create Gauge, but **onlyOwner** can add new reward token. Deployer may have several rewards tokens, but can't and add reward tokens to **Gauge** without the factory owner.

Recommendation

- If deployer is a trusted party, we recommend allowing the addition of a new reward token for deployer after deploying Gauge. Also, we should allow adding list of reward tokens for **deployGauge** function.
- If deployer not trusted party, we recommend denying adding reward token for **deployGauge** function.

INFORMATIONAL-25	Lack of events	Acknowledged

Description

There is a lack of events in some contracts when the state changes:

1. The functions **ArrakisV2GaugeFactory.addGaugeReward()**, **ArrakisV2GaugeFactory.setGaugeRewardDistributor()** do not emit an event, neither does the gauge contract. It could be useful to track this data.
2. Since anyone can call the function **ArrakisV2StaticManager.setStaticVault()**, it should emit an event to track which vaults are added.
3. The function **ArrakisV2StaticManager.withdrawAndCollectFees()** doesn't emit an event with which vaults were withdrawn from and which tokens were transferred.
4. The function **ArrakisV2RouterStorage.updateSwapExecutor()** doesn't emit an event when the swap executor changes.

Recommendation

We recommend to add events where necessary.

INFORMATIONAL-26	Low-level call doesn't forward the revert reason	Acknowledged

Description

At the line RouterSwapExecutor.sol#L55:

```
(bool success, ) = swapAndAddData_.swapData.swapRouter.call(
    swapAndAddData_.swapData.swapPayload
);
require(success, "swap: low-level call failed");
```

The low-level call doesn't forward the revert reason. If the transaction reverts, it can be useful for a user to know why it reverted.

Recommendation

We recommend to forward the revert reason.

INFORMATIONAL-27	Owner can renounce ownership	Acknowledged

Description

The **OwnableUpgradeable** contract implements a **renounceOwnership()** function which will remove any functionality that is only available to the owner. The contracts **ArrakisV2GaugeFactory**, **ArrakisV2RouterStorage**, **ArrakisV2StaticManagerStorage** inherit from **OwnableUpgradeable**, but do not override the **renounceOwnership()** function. The contracts have functions that are callable only by the owner, so if **renounceOwnership()** is called by mistake, these functions will be uncallable.

Recommendation

We recommend to override **renounceOwnership()** to revert on call.

INFORMATIONAL-28	ArrakisV2RouterStorage allows direct ETH transfers	Acknowledged

Description

ArrakisV2RouterStorage has the **receive()** fallback to receive ETH sent by the **WETH** contract. Currently, any user can send ETH directly. It would be better to allow only the **WETH** contract to send ETH. It is possible to force ETH by using **selfdestruct**, but this would disallow locking ETH sent by mistake.

Recommendation

We recommend to change to:

```
receive() external payable {
    require(msg.sender == weth, 'Not WETH');
}
```

INFORMATIONAL-29	Attacker can steal tokens and allowance from RouterSwapExecutor	Acknowledged

Description

In the function **RouterSwapExecutor.swap()**, **swapRouter** and **swapPayload** are user chosen. An attacker can choose **swapRouter** as any token and transfer from **RouterSwapExecutor** balance or allowance. The **RouterSwapExecutor** should not have any tokens and allowance, but this is not an intended use of **swap()** function.

Recommendation

We recommend to use only whitelisted routers as **swapRouter**.

INFORMATIONAL-30	RouterSwapExecutor transfers all balance after swapping	Acknowledged

Description

In the function **RouterSwapExecutor.swap()**, the **amount1Diff** is calculated as **balance1**. If **RouterSwapExecutor** has non zero **token1** balance before the swap, it will be sent to the router and the leftover to the user. The user will intentionally or unintentionally receive the tokens from the contract.

Recommendation

We recommend to calculate **amount1Diff** the same way as **amount0Diff**. If the locking of tokens in **RouterSwapExecutor** is undesired, add a recover method to retrieve tokens.

INFORMATIONAL-31	Incorrect getter method in ArrakisV2GaugeFactory	Acknowledged

Description

In getProxyAdmin method of **ArrakisV2GaugeFactoryStorage** contract, there is a staticcall to **admin()** method of **ArrakisV2Beacon**. But **ArrakisV2Beacon** doesn't have a such method but has **owner()** one.

Recommendation

We either recommend calling **owner()** method instead of **admin()** or changing the function name to **getGaugeAdmin**.

INFORMATIONAL-32	Incorrect comments in ArrakisV2GaugeFactoryStorage	Acknowledged

Description

In getProxyAdmin and getProxyImplementation methods of **ArrakisV2GaugeFactoryStorage** contract, it is stated:

```
// We need to manually run the static call since the getter cannot be flagged as view
```

But still, it is possible to set the interface in a such way.

Recommendation

We recommend changing or removing the comments.

INFORMATIONAL-33	Lack of nonreentrant modifier in RouterSwapExecutor	Acknowledged

Description

Method swap of **RouterSwapExecutor** doesn't have nonreentrant modifier. So it is possible to make reentrancy calls in **swap** if an external **router** doesn't have such modifiers.

Recommendation

We recommend adding nonreentrant modifier.

INFORMATIONAL-34	Uninitialized swapper in ArrakisV2Router	Acknowledged

Description

swapper field of ArrakisV2RouterStorage is left uninitialized even after initialize method call.

Recommendation

We recommend initializing it with other fields in **initialize** method.

INFORMATIONAL-35	Incorrect access modifiers	Acknowledged

Description

Misconfiguration in **ArrakisV2Staticmanager** about ability to set vault as static was solved by adding **onlyDeployer** access modifier. But similar one problem in **ArrakisV2GaugeFactory** is still actual. It will lead to set up **owner()** of factory as main distributor and another one thing is that functionality is providen for only owner of this contract. This behaviour is incomprehensible for our team.

Recommendation

We recommend adding similar access modifier, or providing additional info about this question.

INFORMATIONAL-36	Incorrect name of variables	Acknowledged

Description

In **ArrakisV2Router** functions there's an requirement, that **amount0Max** or **amount1Max** is above zero, but there's no comparison between max and min values of some structs. Moreover, in some cases **amount0** and **amount1** can be above max values cause of roundings up in mint functions. So, due to roundings up it seems like both **amount0Max** and **amount1Max** should be over 0.

```
require(
  params_.amount0Max > 0 || params_.amount1Max > 0,
  ^-- this || check should set only on swapAndAdd functions
  "Empty max amounts"
);

...

(amount0, amount1, sharesReceived) = resolver.getMintAmounts(
  ^-- Here mulDivRoundingsUp() is used, so amount can be more than amountMax
  IArrakisV2(params_.vault),
  params_.amount0Max,
  params_.amount1Max
);

require(
  amount0 >= params_.amount0Min &&
    amount1 >= params_.amount1Min &&
    sharesReceived >= params_.amountSharesMin,
  "below min amounts"
);
^-- Only check that result amounts doesn't less than min
...
```

Recommendation

We recommend renaming these variables or provide additional logic and requirements for them, making code more clear.

INFORMATIONAL-37	Inconsistent logic of withdrawing fees	Acknowledged

Description

In **ArrakisV2StaticManager.withdrawAndCollectFees()** there's a check, that length of vaults array should be more than 0, but as we see below we can set custom tokens to withdraw, which could not correspond to vault's ones.

Recommendation

We recommend removing this line

```
require(vault_.length > 0, "ZV");
```

7. Appendix B. Slither

Informational/High/low-level-calls

Low level call in RouterSwapExecutor.swap(SwapAndAddData): - (success) = swapAndAddData_.swapData.swapRouter.call(swapAndAddData_.swapData.swapPayload)

Low level call in ArrakisV2GaugeFactoryStorage.getProxyAdmin(address): - (success,returndata) = proxy.staticcall(0xf851a440)

Low level call in ArrakisV2GaugeFactoryStorage.getProxyImplementation(address): - (success,returndata) = proxy.staticcall(0x5c60da1b)

Informational/High/missing-inheritance

ArrakisV2GaugeFactory should inherit from IArrakisV2GaugeFactory

ArrakisV2Router should inherit from IArrakisV2Router

Informational/High/naming-convention

Function IGauge.claim_rewards(address) is not in mixedCase

Function IGauge.commit_transfer_ownership(address) is not in mixedCase

Function IGauge.accept_transfer_ownership() is not in mixedCase

Function IGauge.set_reward_distributor(address,address) is not in mixedCase

Function IGauge.reward_data(address) is not in mixedCase

Function IGauge.deposit_reward_token(address,uint256) is not in mixedCase

Function IGauge.claimable_reward(address,address) is not in mixedCase

Function IGauge.user_checkpoint(address) is not in mixedCase

Function IGauge.reward_count() is not in mixedCase

Function IGauge.staking_token() is not in mixedCase

Variable ArrakisV2GaugeFactoryStorage._gauges is not in mixedCase

Function IGauge.future_admin() is not in mixedCase

Function IGauge.reward_tokens(uint256) is not in mixedCase

Function IGauge.add_reward(address,address) is not in mixedCase

Function IGauge.claimed_reward(address,address) is not in mixedCase

Informational/High/solc-version

Pragma version0.8.13 allows old versions

Pragma version0.8.13 allows old versions

Pragma version0.8.13 allows old versions

Pragma version>=0.8.0 allows old versions

Pragma version>=0.8.0 allows old versions

Pragma version0.8.13 allows old versions

Pragma version>=0.8.0 allows old versions

Pragma version>=0.8.0 allows old versions

Pragma version>=0.8.0 allows old versions

Pragma version>=0.8.0 allows old versions

Pragma version0.8.13 allows old versions

Pragma version>=0.8.0 allows old versions

Pragma version0.8.13 allows old versions

Pragma version0.8.13 allows old versions

Pragma version>=0.8.0 allows old versions

Pragma version>=0.8.0 allows old versions

Pragma version>=0.8.0 allows old versions

Pragma version>=0.8.0 allows old versions

Pragma version>=0.8.0 allows old versions

solc-0.8.13 is not recommended for deployment

Pragma version0.8.13 allows old versions

Pragma version0.8.13 allows old versions

Pragma version0.8.13 allows old versions

Pragma version>=0.8.0 allows old versions

Pragma version>=0.8.0 allows old versions

Informational/Medium/dead-code

Twap.getPrice0(IUniswapV3Pool,uint24) is never used and should be removed

Twap.getPrice1(IUniswapV3Pool,uint24) is never used and should be removed

Twap.getSqrtTwapX96(IUniswapV3Pool,uint24) is never used and should be removed

Informational/Medium/similar-names

Variable ArrakisV2Router.swapAndAddLiquidity(SwapAndAddData).amount0Diff is too similar to ArrakisV2Router._swapAndAddLiquidity(SwapAndAddData).amount1Diff

Variable RouterSwapResolver.calculateSwapAmount(IArrakisV2,uint256,uint256,uint256).amount0Left is too similar to RouterSwapResolver.calculateSwapAmount(IArrakisV2,uint256,uint256,uint256).amount1Left

Variable ArrakisV2Router._swapAndAddLiquidity(SwapAndAddData).amount0Diff is too similar to ArrakisV2Router.swapAndAddLiquidity(SwapAndAddData).amount1Diff

Variable RouterSwapExecutor.swap(SwapAndAddData).amount0Diff is too similar to RouterSwapExecutor.swap(SwapAndAddData).amount1Diff

Variable ArrakisV2Router.swapAndAddLiquidityPermit2(SwapAndAddPermit2Data).amount0Diff is too similar to ArrakisV2Router.swapAndAddLiquidity(SwapAndAddData).amount1Diff

Variable ArrakisV2Router.swapAndAddLiquidityPermit2(SwapAndAddPermit2Data).amount0Diff is too similar to ArrakisV2Router._swapAndAddLiquidity(SwapAndAddData).amount1Diff

Variable IArrakisV2Router.swapAndAddLiquidityPermit2(SwapAndAddPermit2Data).amount0Diff is too similar to IArrakisV2Router.swapAndAddLiquidityPermit2(SwapAndAddPermit2Data).amount1Diff

Variable ArrakisV2Router.swapAndAddLiquidity(SwapAndAddData).amount0Diff is too similar to ArrakisV2Router.swapAndAddLiquidityPermit2(SwapAndAddPermit2Data).amount1Diff

Variable ArrakisV2Router._swapAndAddLiquidity(SwapAndAddData).amount0Diff is too similar to ArrakisV2Router._swapAndAddLiquidity(SwapAndAddData).amount1Diff

Variable IArrakisV2Router.swapAndAddLiquidityPermit2(SwapAndAddPermit2Data).amount0Diff is too similar to IArrakisV2Router.swapAndAddLiquidity(SwapAndAddData).amount1Diff

Variable ArrakisV2Router.swapAndAddLiquidity(SwapAndAddData).amount0Diff is too similar to ArrakisV2Router.swapAndAddLiquidity(SwapAndAddData).amount1Diff

Variable IArrakisV2Router.swapAndAddLiquidity(SwapAndAddData).amount0Diff is too similar to IArrakisV2Router.swapAndAddLiquidity(SwapAndAddData).amount1Diff

Variable ArrakisV2StaticManager.compoundFees(IArrakisV2).proportion0 is too similar to ArrakisV2StaticManager.compoundFees(IArrakisV2).proportion1

Variable IRouterSwapExecutor.swap(SwapAndAddData).amount0Diff is too similar to RouterSwapExecutor.swap(SwapAndAddData).amount1Diff

Variable ArrakisV2Router._swapAndAddLiquidity(SwapAndAddData).amount0Diff is too similar to ArrakisV2Router.swapAndAddLiquidityPermit2(SwapAndAddPermit2Data).amount1Diff

Variable ArrakisV2Router.swapAndAddLiquidityPermit2(SwapAndAddPermit2Data).amount0Diff is too similar to ArrakisV2Router.swapAndAddLiquidityPermit2(SwapAndAddPermit2Data).amount1Diff

Variable RouterSwapExecutor.swap(SwapAndAddData).amount0Diff is too similar to IRouterSwapExecutor.swap(SwapAndAddData).amount1Diff

Variable IRouterSwapExecutor.swap(SwapAndAddData).amount0Diff is too similar to IRouterSwapExecutor.swap(SwapAndAddData).amount1Diff

Variable IArrakisV2Router.swapAndAddLiquidity(SwapAndAddData).amount0Diff is too similar to IArrakisV2Router.swapAndAddLiquidityPermit2(SwapAndAddPermit2Data).amount1Diff

Low/High/shadowing-local

IGauge.initialize(address,address,address,address,address,address).admin shadows: - IGauge.admin() (function)

Low/Medium/calls-loop

ArrakisV2GaugeFactory.addGaugeReward(IGauge,address,address) has external calls inside a loop: require(bool,string)(gauge_.reward_tokens(i) != token_,AE)

ArrakisV2StaticManager.withdrawAndCollectFees(IArrakisV2[],IERC20[],address) has external calls inside a loop: require(bool,string)(vaults_[i].manager() == address(this),NM)

ArrakisV2StaticManager.withdrawAndCollectFees(IArrakisV2[],IERC20[],address) has external calls inside a loop: vaults_[i].withdrawManagerBalance()

ArrakisV2StaticManager.withdrawAndCollectFees(IArrakisV2[],IERC20[],address) has external calls inside a loop: balance = IERC20(tokens_[i_scope_0]).balanceOf(address(this))

Low/Medium/missing-zero-check

ArrakisV2GaugeFactoryStorage.getProxyAdmin(address).proxy lacks a zero-check on : - (success,returndata) = proxy.staticcall(0xf851a440)

ArrakisV2GaugeFactoryStorage.getProxyImplementation(address).proxy lacks a zero-check on : – (success,returndata) = proxy.staticcall(0x5c60da1b)

RouterSwapExecutor.constructor(address)._router lacks a zero-check on : – router = _router

Low/Medium/reentrancy-benign

Reentrancy in ArrakisV2StaticManager.setStaticVault(SetStaticVault): External calls: – IArrakisV2(params_.vault).setManagerFeeBPS(managerFeeBPS) State variables written after the call(s): – vaults[params_.vault] = params_.vaultInfo

Low/Medium/reentrancy-events

Reentrancy in ArrakisV2GaugeFactory.deployGauge(address,address,address): External calls: – gauge = deploy(stakingToken) – gauge = address(new BeaconProxy(address(arrakisGaugeBeacon),data)) – IGauge(gauge).add_reward(rewardToken_,rewardDistributor_) Event emitted after the call(s): – GaugeCreated(msg.sender,gauge)

Reentrancy in ArrakisV2StaticDeployer.deployStaticVault(InitializeStatic): External calls: – [vault = arrakisFactory.deployVault(InitializePayload(params_.feeTiers,params_.token0,params_.token1,address(this),init0,init1,address(this),new address),true)](https://github.com/ArrakisFinance/v2-periphery/blob/cab630396506aad825d838f98d60d287ed49c0b9/contracts/ArrakisV2StaticDeployer.sol#L80-L92) – IERC20(params_.token0).safeApprove(vault,init0) – IERC20(params_.token1).safeApprove(vault,init1) – IERC20(params_.token0).safeTransferFrom(msg.sender,address(this),init0) – IERC20(params_.token1).safeTransferFrom(msg.sender,address(this),init1) – IArrakisV2(vault).mint(1000000000000000000,params_.receiver) – IArrakisV2(vault).rebalance(rebalance) – IArrakisV2(vault).setManager(address(staticManager)) – staticManager.setStaticVault(SetStaticVault(vault,params_.vaultInfo)) – gauge = gaugeFactory.deployGauge(vault,params_.rewardToken,params_.rewardDistributor) – IArrakisV2(vault).renounceOwnership() Event emitted after the call(s): – CreateStaticVault(vault,gauge,msg.sender,init0,init1)

Reentrancy in ArrakisV2StaticManager.compoundFees(IArrakisV2): External calls: – vault_.rebalance(rebalance) Event emitted after the call(s): – Compound(address(vault_),msg.sender,growthFactor)

Medium/Medium/uninitialized-local

ArrakisV2Router._swapAndAddLiquidity(SwapAndAddData).isToken0Weth is a local variable never initialized

ArrakisV2Router.addLiquidityPermit2(AddLiquidityPermit2Data).isToken0Weth is a local variable never initialized

ArrakisV2Router.addLiquidity(AddLiquidityData).isToken0Weth is a local variable never initialized

ArrakisV2StaticDeployer.deployStaticVault(InitializeStatic).rebalance is a local variable never initialized

ArrakisV2StaticDeployer._getInits(PositionLiquidity[],address,address).i is a local variable never initialized

ArrakisV2StaticManager._checkTWAPs(IArrakisV2,uint24,int24).i is a local variable never initialized

ArrakisV2Router.swapAndAddLiquidity(SwapAndAddData).isToken0Weth is a local variable never initialized

ArrakisV2GaugeFactory.addGaugeReward(IGauge,address,address).i is a local variable never initialized

ArrakisV2StaticManager.withdrawAndCollectFees(IArrakisV2[],IERC20[],address).i is a local variable never initialized

ArrakisV2StaticManager.compoundFees(IArrakisV2).rebalance is a local variable never initialized

ArrakisV2StaticManager.compoundFees(IArrakisV2).i is a local variable never initialized

ArrakisV2StaticManager.withdrawAndCollectFees(IArrakisV2[],IERC20[],address).i_scope_0 is a local variable never initialized

Medium/Medium/unused-return

ArrakisV2Router._addLiquidity(address,uint256,uint256,uint256,address,address) ignores return value by IArrakisV2(vault_).mint(mintAmount_,address(this))

ArrakisV2StaticDeployer.deployStaticVault(InitializeStatic) ignores return value by IArrakisV2(vault).mint(1000000000000000000,params_.receiver)

ArrakisV2GaugeFactory.deployGauge(address,address,address) ignores return value by _gauges.add(gauge)

ArrakisV2Router._addLiquidity(address,uint256,uint256,uint256,address,address) ignores return value by IArrakisV2(vault_).mint(mintAmount_,receiver_)

8. Appendix C. Tests

Tests result

```
67 passing (3m)
```

Tests coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts\	89.02	63.14	91.89	87.41	
ArrakisV2GaugeBeacon.sol	100	100	100	100	
ArrakisV2GaugeFactory.sol	100	75	100	100	
ArrakisV2Router.sol	93.13	64.84	100	93.29	... 673,692,696
ArrakisV2StaticDeployer.sol	95.65	91.67	75	96.67	137
ArrakisV2StaticManager.sol	100	55.56	100	100	
RouterSwapExecutor.sol	100	75	100	100	
RouterSwapResolver.sol	0	0	33.33	7.14	... 100,101,102
contracts\abstract\	42.86	16.67	50	55.88	
ArrakisV2GaugeFactoryStorage.sol	33.33	13.64	40	42.11	... 102,105,106
ArrakisV2RouterStorage.sol	60	25	60	77.78	57,61
ArrakisV2StaticManagerStorage.sol	50	16.67	50	66.67	37,41
contracts\libraries\	58.33	33.33	40	50	
Twap.sol	58.33	33.33	40	50	... 56,68,70,72
contracts\structs\	100	100	100	100	
SArrakisV2Router.sol	100	100	100	100	
SArrakisV2StaticDeployer.sol	100	100	100	100	
SPermit2.sol	100	100	100	100	

SStaticManager.sol	100	100	100	100
All files	84.23	57.28	76.79	82.37

STATE
MIND