STATE
MIND

Arrakis v2 core and palm

01-11-2022 - 25-11-2022

Table of contents

1. Project Brief

Title	Description
Client	Arrakis
Project name	Arrakis v2 core and palm
Timeline	01-11-2022 - 25-11-2022
Initial commit	376bfcec803f0644fdc601db3a5772d2179c13a0, 06f8439430e3b0af9cbbf887926ff93844c28a7d
Final commit	683a355de3317278f5f09dcd8aa136e1a8f80639, 0d09e21c818542d6705b8a84a3233a473ac5fff3

Short Overview

Arrakis is web3's liquidity layer, which at its core acts as a decentralized market-making platform enabling projects to create deep liquidity for their tokens on decentralized exchanges.
The core contracts allow users to:

- create an `ArrakisV2` vault instance that manages holdings of a given token pair
- dispatch and collect these holdings to/from Uniswap V3 Liquidity Positions (for the defined token pair) via a settable `manager` smart contract
- configure important vault setup parameters (manager, restrictedMint, pools) via the vault `owner` role

PALM is the first application built on top of the flexible ArrakisV2 core system, optimized for automated management of protocol owned liquidity (thus, Protocol Automated Liquidity Management).
PALM allows users to:

- Create a "private" vault that is managed by `PALMManager` who will run automated strategies on behalf of the vault creator. Only vault creators can add and remove liquidity from their private vault
- Vault creators have the ability to pick from a list of whitelisted strategy templates, and further configure the strategy with custom parameters
- Vault creators can increase or decrease liquidity deposited in the vault at any time, as well as change the strategy configuration (or delegate this strategy configuration ability to a third party)
- Finally vault creators can remove all of their liquidity and close the vault at any time

Project Scope

The audit covered the following files:

ArrakisV2Storage.sol	ArrakisV2FactoryStorage.sol	ArrakisV2Helper.sol
Position.sol	UniswapV3Amounts.sol	Underlying.sol
Pool.sol	Manager.sol	SArrakisV2Helper.sol
SArrakisV2.sol	ArrakisV2Resolver.sol	ArrakisV2.sol
FArrakisV2Factory.sol	ArrakisV2Factory.sol	ArrakisV2Beacon.sol
PALMManager.sol	PALMTermsStorage.sol	PALMManagerStorage.sol
SPALMTerms.sol	SPALMManager.sol	FPALMTerms.sol
PALMTerms.sol		

2. Finding Severity breakdown

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

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss 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 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
High	6
Medium	3
Informational	42

4. Conclusion

Commits with all fixes: `683a355de3317278f5f09dcd8aa136e1a8f80639`, `0d09e21c818542d6705b8a84a3233a473ac5fff3`
6 high and 3 medium and 42 informational severity issue was found, 5 out of 6 high, 2 out of 3 medium and 26 out of 42 informational severity issues were fixed.

Deployment

File name	Contract deployed on mainnet
ArrakisV2.sol	0xb5C3B286dD591282Fe87Dfab0613488e1b6B09Ba
ArrakisV2Beacon.sol	0x891E7E4baFfeF0ef7bc4b1E85d122bDd7363b8B3
ArrakisV2Factory.sol	0x055B6d3919042Be29C5F044A55529933e1273A88
ArrakisV2Helper.sol	0xccEe73eA4c7a42491c68FEa78B1BDDD1A35C8d9C
ArrakisV2Resolver.sol	0x4bc385b1dDf0121CC40A0715CfD3beFE52f905f5
Pool.sol	0x4cD41204AA4C7438374256bD7bE850eF9fcFaB84
Position.sol	0xF7cB77C8dCB22A1bb4435932f3515319721Faf44
Underlying.sol	0x92CB4F7e4CB623E73D5Ec84A43669ADc757C2bd2

File name	Contract deployed on mainnet
PALMManager.sol	0xF90ec87BA0BA9AC92f5374f112740Ce291B8877e
PALMTerms.sol	0x50763a665Dc24692E25eC8e2c203A79e602D2890

5. Findings report

High

Possible loss of funds by the manager — Acknowledged

Description

The manager has full access to vault rebalance parameters. E.g. In the function `_rebalance` manager can:

1. Burn all the liquidity in the uniswap pool (contract approves all its balance for transfer regardless of swap amount);
2. Asset swap via low-level call;
3. Pass his address as a recipient to the swap function or sandwich himself (or by someone else).

Worth noting that if vault uses the same uniswap pool for swaps then the sandwich attack becomes even cheaper (easier to skew the price because of the burning [1]).

Recommendation

We recommend calling swap routers via an interface (or implementing a new contract to manage swaps) to check all necessary input parameters and avoid making arbitrary low-level calls. Another way to address this issue may be adding restrains to the `SwapPayload::expectedMinReturn` parameter, but it may require an external oracle.

Client's comments

> Left unresolved. From perspective of v2-core manager is intentionally a trusted party that passes potentially sensitive/manipulable parameters. For publicly accessible vaults, specific manager contract implementations will be used to yield more trustless manager

Possible uniswap pool manipulation — Fixed at 23ac78

Description

If, when calling the `burn` function, the vault doesn't have enough of either token0 or token1 (e.g. after the vault mints positions) it will burn some of the liquidity according to user input. Arbitrary users can, intentionally or not, burn all the pool's liquidity and the vault won't receive any yield until the next rebalance.
On the other side, it will negatively affect all the uniswap users, if the vault holds a significant part of the pool's liquidity, e.g. using ArrakisV2 as a tool to aid in the sandwich attacks.

Recommendation

We recommend decreasing users' exposure to position burn functionality.

Possible fee hijacking (or DOS) by the manager — Fixed at 50706e

Description

Currently, Arrakis relies on the manager's good faith, since the vault quarries the manager's smart contract to get the `managerFeeBPS` parameter. Although the average scenario manager would be an instance of `PALMManager.sol` (`managerFeeBPS` is immutable), an owner can change the manager address. An attacker with manager access could set `managerFeeBPS = 10000 - arrakisFeeBPS` (or set `managerFeeBPS` > 10000 - arrakisFeeBPS` to break the vault) to steal all the yield (+ all the accrued manager fees).

Recommendation

We recommend storing the `managerFeeBPS` parameter value in the storage of `ArrakisV2Storage` contract with additional value restrains in setters (explicit max & min values) and limiting `managerFeeBPS <= hundredPercent` on setting.

First minter can change LP token pricing — Fixed at 23ac78

Description

At the lines ArrakisV2.sol#L83-L84:

```
amount0 = FullMath.mulDivRoundingUp(mintAmount_, current0, denominator);
amount1 = FullMath.mulDivRoundingUp(mintAmount_, current1, denominator);
```

When `totalSupply` is `0`, the variables `current0, current1, denominator` are set to `init0, init1, 1 ether` respectively. If `init0` and `init1` are less than `1e18`, then the first minter can mint `mintAmount_ = 1 wei` for `amount0 = 1 wei` and `amount1 = 1 wei` regardless of the values `init0` and `init1`.

Recommendation

It is recommended to ensure `amount0, amount1` are proportional to `init0, init1`.

Vault can renew the term for free — Fixed at 465466

Description

The vault owner mints `1 wei` when deploying a vault with `openTerm()`. Then they can renew term for free since in the function `renewTerm()` at the line PALMTerms.sol#L153, `emolumentShares` are calculated from the balance of `PALMTerms` and the function `increaseLiquidity()` doesn't mint new LP tokens.

Recommendation

It is recommended to mint new LP tokens in `increaseLiquidity()`.

Vault owner can lose tokens when increasing liquidity — Fixed at 90e5c1

Description

If the vault owner burns all LP tokens owned by `PALMTerms` with `decreaseLiquidity()`, then any future call to `increaseLiquidity()` would lose tokens in the vault's address. PALMTerms.sol#L170

Recommendation

It is recommended to mint new LP tokens in increaseLiquidity().

Medium

Checking caller — Fixed at 465466

Description

Function addVault() doesn't check `msg.sender`. Anyone can call this function and add vault if its owner is `terms`. This can lead to DOS of contract and unintended behavior.

Recommendation

It is recommended to add `msg.sender` check, if it is vault's owner or `terms`.

No check if liquidity > 0 in standardBurnParams() — Fixed at 50706e

Description

In the function `standardBurnParams()` at the lines ArrakisV2Resolver.sol#L220-L225:

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

There is no check if `burns[i].liquidity > 0`. If `burns[i].liquidity = 0`, then `burn()` in `ArrakisV2` will revert.

Recommendation

It is recommended to check if `burns[i].liquidity > 0` and not include range if `liquidity` = 0.

Anyone can call renewTerm() — Acknowledged

Description

In the function `renewTerm()` at the line PALMTerms.sol#L143, there is no check if the caller is the owner of the vault. An attacker can frontrun `closeTerm()` and call `renewTerm()`, then the vault would pay the emolument twice.

Recommendation

It is recommended to add the modifier `requireIsOwner()`.

Client's comments

> By design. RenewTerm can be called only when term management time ends. If client want to terminate the term with palm, they can call closeTerm function before end of management time. If they do not, renewal is automatic and we (or anyone) will promptly call renewTerm to extract the fee of the last epoch.

Informational

Redundant modifier — Fixed at 90e5c1

Description

PALMTerms deploys new vaults with himself as an owner via openTerm function, as well PALMManager's addVault function accepts only term-owned vaults.

PALMManager's removeVault, setVaultData, setVaultStraByName and withdrawVaultBalance methods have modifier onlyVaultOwner, which is redundant as these methods are always called from PALMTerms. addVault has the onlyPALMTermsVaults modifier (you can add random garbage to the mapping by bypassing it), which could be replaced with onlyPALMTerms modifier too, since it is always supposed to be called from PALMTerms.

Recommendation

We recommend replacing onlyVaultOwner and onlyPALMTermsVaults modifiers with onlyPALMTerms.

Unused modifier code duplication — Fixed at 90e5c1

Description

PALMManagerStorage removeVault function has the same require check as onlyManagedVaults modifier.

Recommendation

We recommend replacing the duplicated line with onlyManagedVaults modifier.

Reducing SLOAD operations — Fixed at 23ac78

Description

You can reduce SLOAD operations, hence reducing gas spending, by copying often-used storage variables to the memory.

Recommendation

We recommend copying storage variables to memory in case of multiple variable read operations. For example, ArrakisV2::_rebalance function could be optimized by copying factory, token0 and token1 into the memory before all loops.

Event indexed fields — Fixed at 23ac78

Description

In ArrakisV2Storage, events LogMint, LogBurn and LPBurned could be modified to have reciever (user) field indexed to improve parsing user balance.

Recommendation

We recommend making reciever(user) field indexed.

Typo in function naming — Fixed at 90e5c1

Description

PALMManagerStorage function setVaultStraByName should be spelled setVaultStratByName.

Recommendation

We recommend changing the naming and all its references in IPALMManager and PALMTermsStorage.

Direct token transfers — Acknowledged

Description

If `ArrakisV2` contract has pool tokens on its balance (excluding arraking and manager balances) and if its total supply is zero then anyone who calls mint() function will get more tokens than he transferred. It is possible if minting LP tokens and transferring tokens to vault take place not atomically and mint is not restricted.

Recommendation

It is recommended to avoid direct transfers to `ArrakisV2` vault if mint is not restricted.

Range existence check — Fixed at 23ac78

Description

In cycle at Line 169 `burns_[i].range` may not be present in the `ranges` array.

Same issue:

- In cycle at Line 322

Recommendation

It is recommended to add `require` statement if range exists.

Gas optimisation in range deletion — Acknowledged

Description

During the deletion of range at Line 269 all elements after it are moved left in cycle at Lines 271-273.

Recommendation

It is recommended to swap elements to delete and the last element of the array and call `pop()` function.

Zero address check — Fixed at 23ac78

Description

At Line 144 `params.owner` variable is not checked for zero address. If ownership is transferred to zero address, admin functionality will be unavailable and it can not be restored.

Same issue:

- Line 43
- Line 49
- Line 13
- Line 136
- Line 88

Recommendation

It is recommended to add `require` statement to check variables and parameters for zero address.

Bad readability — Fixed at 23ac78

Description

At Line 149 variables `init0` and `init1` are set during event emmitting.

Same issue:

- Line 161
- Line 193
- Line 197
- Line 101
- Line 113
- Line 125
- Line 135
- Line 215
- Line 293

Recommendation

It is recommended to separate variable initialization and event emmitting.

Redundant check — Acknowledged

Description

At Line 171 `_pools.contains()` is redundant because it is checked in `_pools.remove()`. Same comment with `_pool.add()`.

Same issue:

- Line 185
- Line 234
- Line 248
- Line 312
- Line 261

Recommendation

It is recommended to use `require` statement with `remove()` and `add()` EnumerableSet' methods instead of `containts()` method.

Gas optimisation in vaults() — Acknowledged

Description

Function vaults() iterates through `_vaults` enumerable set and saves values to an allocated memory array. The enumerable set has built-in function `values()` for that purpose.

Recommendation

It is recommended to use the enumerable set's function `values` to reduce code size and gas costs.

Gas optimisation in conversion from int to string — Acknowledged

Description

Function _uint2str is more gas expansive than Openzeppelin function `toString()`.

Recommendation

It is recommended to use Openzeppelin function `toString()` to convert `uint` to `str`.

Code refactoring — Acknowledged

Description

In the function standartBurnParams() helper's function `totalUnderlyingWithFees()` is called and then leftovers are calculated. `ArrakisV2Helper` contract has a function that calculates leftovers - `totalUnderlyingWithFeesAndLeftOver()`.

Recommendation

It is recommended to use `ArrakisV2Helper`'s function `totalUnderlyingWithFeesAndLeftOver()` to get all necessary data.

Zero liquidity check — Fixed at 23ac78

Description

At Line 205 liquidity of position is returned and saved to `burns` array. If liquidity is zero, it is redundant to add position info to an array.

Recommendation

It is recommended to add `require` statement to check if the position has liquidity.

Blacklisting strategy — Acknowledged

Description

In `PALMManagerStorage` contract there is a method to whitelist strategies, but there is no method to blacklist them if a strategy is irrelevant or ineffective.

Recommendation

It is recommended to add a method to blacklist strategies.

Redundant approve call — Acknowledged

Description

At Lines 104-105 vault's allowance is set to zero for both tokens, but it is already zero.

Recommendation

It is recommended to remove `safeApprove()` calls setting allowance to zero.

Unmatched to documentation — Acknowledged

Description

Function increaseLiquidity() simply transfers tokens from `msg.sender` to vaults. Based on docs, it should call `mint()` function.

Same issue:

- In function rebalance() during deposits there is no check for current and average price.

Recommendation

It is recommended to leave a comment if it is intended behaviour.

No max limit for ranges

Acknowledged

Description

In the function `rebalance()` at the line ArrakisV2.sol#L241.
The manager can add ranges, but there is no max limit for the number of ranges. If the `ranges` array is too big, it will be impossible to mint and burn LP tokens since there will not be enough gas in the block.

Recommendation

It is recommended to limit the maximum number of ranges.

Variable totalSupply shadows function

Fixed at 23ac78

Description

At the lines ArrakisV2.sol#L63, ArrakisV2.sol#L106:

```
uint256 totalSupply = totalSupply();
```

The variable `totalSupply` shadows function `totalSupply()`.

Recommendation

It is recommended to rename the variable.

Possible to use LP token as vault token

Acknowledged

Description

In the function `mint()` at the lines ArrakisV2.sol#L86-L94:

```
_mint(receiver_, mintAmount_);

// transfer amounts owed to contract
if (amount0 > 0) {
    token0.safeTransferFrom(msg.sender, me, amount0);
}
if (amount1 > 0) {
    token1.safeTransferFrom(msg.sender, me, amount1);
}
```

The LP token is minted before `token0` and `token1` are pulled to the contract, which means that it is possible to use LP token of the vault as `token0` or `token1`.

Recommendation

It is recommended to mint LP tokens after pulling `token0` and `token1`.

Possible to burn zero LP tokens

Fixed at 23ac78

Description

In the function `burn()` at the line ArrakisV2.sol#L101, the parameter `burnAmount_` is not checked to be bigger than `0`.

Recommendation

It is recommended to check if `burnAmount_ > 0`.

Incorrect event emit

Acknowledged

Description

In the function `burn()` at the line ArrakisV2.sol#L216:

`emit LogUncollectedFees(underlying.fee0, underlying.fee1);`

Some fees will be collected when burning liquidity, so this event emit `LogUncollectedFees()` is incorrect.

Recommendation

It is recommended to emit `LogUncollectedFees()` before the `return` statement at the line ArrakisV2.sol#L159.

TODO comments

Fixed at 23ac78

Description

At the lines:

- ArrakisV2.sol#L238
- ArrakisV2Resolver.sol#L111

`TODO` comments should be removed before deployment.

Recommendation

It is recommended to remove `TODO` comments.

Using a number literal

Acknowledged

Description

- In the function `_applyFees()` at the lines ArrakisV2.sol#L462-L45
- At the lines ArrakisV2Resolver.sol#L133-L134
- At the line ArrakisV2Resolver.sol#L292
- At the line PALMTermsStorage.sol#L87

The literal `10000` can be replaced with a constant variable for better readability.

Recommendation

It is recommended to use a constant variable instead of a literal.

Inadequate view functions in ArrakisV2Storage

Acknowledged

Description

At the lines ArrakisV2Storage.sol#L65-L66:

```
EnumerableSet.AddressSet internal _pools;
EnumerableSet.AddressSet internal _routers;
```

The are no view functions for pools and routers used in the vault.

Recommendation

It is recommended to add view functions for pools and routers.

Function removePools() doesn't remove ranges

Acknowledged

Description

In the function `removePools()` at the lines ArrakisV2Storage.sol#L169-L176:

```
function removePools(address[] calldata pools_) external onlyOwner {
    for (uint256 i = 0; i < pools_.length; i++) {
        require(_pools.contains(pools_[i]), "NP");

        _pools.remove(pools_[i]);
    }
    emit LogRemovePools(pools_);
}
```

The pool is only removed from `_pools`, but there might be active positions with same fee tier in `_ranges`.

Recommendation

It is recommended to remove ranges with the same fee tier when removing a pool and ensure they have no liquidity.

Functions can be declared as external

Fixed at [23ac78](23ac78)

Description

- The function `vaults()` at the line ArrakisV2Factory.sol#L57 can be `external` since it is not used internally.
- The function `getProxyAdmin()` at the line ArrakisV2FactoryStorage.sol#L86 can be `external` since it is not used internally.
- The function `getProxyImplementation()` at the line ArrakisV2FactoryStorage.sol#L96 can be `external` since it is not used internally.
- The function `getAmountsForLiquidity()` at the line ArrakisV2Resolver.sol#L266 can be `external` since it is not used internally.

Recommendation

It is recommended to change these functions to `external`.

Function vaults() may run out of gas

Fixed at [23ac78](23ac78)

Description

The function `vaults()` at the lines ArrakisV2Factory.sol#L57-L65:

```
function vaults() public view returns (address[] memory) {
    uint256 length = numVaults();
    address[] memory vs = new address[](length);
    for (uint256 i = 0; i < length; i++) {
        vs[i] = _vaults.at(i);
    }

    return vs;
}
```

If there are a lot of vaults deployed, this function will be unusable since there will not be enough gas in the block to loop over all of the vaults.

Recommendation

It is recommended to change the function to get one vault with parameter `index`.

Unnecessary external calls

Fixed at 23ac78

Description

At the lines:

- ArrakisV2Helper.sol#L36
- ArrakisV2Helper.sol#L71
- ArrakisV2Helper.sol#L88
- ArrakisV2Resolver.sol#L78
- ArrakisV2Resolver.sol#L122
- ArrakisV2Resolver.sol#L206

The external calls `vault_.factory()` are unnecessary since `factory` is already available as an immutable variable.

Recommendation

It is recommended to use variable `factory` instead of an external call.

Inefficient function ranges() in ArrakisV2Helper

Fixed at 23ac78

Description

At the line ArrakisV2Helper.sol#L171, the function `ranges()` is gas inefficient. It can be replaced with a getter function in `ArrakisV2Storage` that would return `ranges`.

Recommendation

It is recommended to add a getter function in `ArrakisV2Storage` to return `ranges`.

Array operators can be changed to EnumerableSet.AddressSet

Fixed at 90e5c1

Description

At the line PALMManagerStorage.sol#L65.
The array `operators` has costly operations when adding, removing operators and checking if an operator exists. It can be optimized if `operators` is a `EnumerableSet.AddressSet`.

Recommendation

It is recommended to change `operators` to a `EnumerableSet.AddressSet`.

gelatoFeeCollector can be set to address(0)

Fixed at 90e5c1

Description

At the lines PALMManagerStorage.sol#L138, PALMManagerStorage.sol#L217.
The variable `gelatoFeeCollector` can be set to `address(0)`.

Recommendation

It is recommended to ensure `gelatoFeeCollector` can never be `address(0)`.

Can fund vault balance with msg.value = 0 — Fixed at 90e5c1

Description

In the function `fundVaultBalance()` at the lines PALMManagerStorage.sol#L275-L283, there is no check if `msg.value > 0`.

Recommendation

It is recommended to check if `msg.value > 0`.

Irrelevant comments — Fixed at 90e5c1

Description

At the lines PALMTerms.sol#L83, PALMTerms.sol#L92, the comments are irrelevant.

Recommendation

It is recommended to move the comment at line PALMTerms.sol#L83 and remove the comment at line PALMTerms.sol#L92.

User can mint LP tokens to themselves in PALMTerms — Fixed at 90e5c1

Description

At the lines PALMTerms.sol#L93-L102, `token0` and `token1` are transferred to the contract before `vaultV2.setRestrictedMint(address(this));` is called. If `token0` or `token1` is an `ERC777` token, then the caller can mint LP tokens before the mint is restricted.

Recommendation

It is recommended to add restricted mint as a vault deployment parameter.

setRestrictedMint() can be frontrun — Acknowledged

Description

At the lines ArrakisV2Storage.sol#L196-L198, the function `setRestrictedMint()` can be frontrun to mint tokens before restricted mint is set since `restrictedMint = address(0)` by default.

Recommendation

It is recommended to add restricted mint as a vault deployment parameter.

Gas optimization in renewTerm() — Fixed at 90e5c1

Description

In the function `renewTerm()` at the line PALMTerms.sol#L149, it is possible to use memory variable `manager_` to save gas.

Recommendation

It is recommended to change to `manager_.renewTerm(address(vault_));`

User can receive less than expected when decreasing liquidity — Fixed at 90e5c1

Description

In the function `decreaseLiquidity()` at the lines PALMTerms.sol#L190-L194.

```
require(
    amount0 >= decreaseBalance_.amount0Min &&
        amount1 >= decreaseBalance_.amount1Min,
    "PALMTerms: received below minimum"
);
```

When calling `decreaseLiquidity()` the user expects to receive `decreaseBalance_.amount0Min`, but actually they receive `amount0 - emolumentAmt0` which might be smaller than `decreaseBalance_.amount0Min` since there is only a check for `amount0`.

Recommendation

It is recommended to check if `amount0 - emolumentAmt0 >= decreaseBalance_.amount0Min`.

Incorrect version — Fixed at 23ac78

Description

Incorrect version here, we think, that it is supposed to be 2.0. ArrakisV2FactoryStorage.sol#L31

Recommendation

We recommend changing the version to 2.0 or another, different from 1.0, to not be confused with the first version before deploying.

Gas optimizations in for loop — Fixed at 23ac78

Description

There're some suboptimal maths here: Position.sol#L40
Underlying.sol#L38
ArrakisV2.sol#L254 … ArrakisV2Resolver.sol#L288
PALMManagerStorage.sol#L227 … PALMManagerStorage.sol#L436

Recommendation

We recommend replacing code like:

```
for (uint i = 0; i < length; i++) {
    ...
}
```

to:

```
for (uint i; i < length; ++i) {
    ...
}
```

Cause it will save some computed units.

Code duplicating	Fixed at 90e5c1

Description

Here, we set `vaults[vault_].balance` to zero, but deleting this element makes the same. PALMManagerStorage.sol#L371.

Recommendation

We recommend deleting duplicated functionality.

7. Appendix B. Slither

Informational/High/low-level-calls

Low level call in ArrakisV2FactoryStorage.getProxyImplementation(address): - (success,returndata) = proxy.staticcall(0x5c60da1b)

Low level call in ArrakisV2FactoryStorage.getProxyAdmin(address): - (success,returndata) = proxy.staticcall(0xf851a440)

Low level call in ArrakisV2._rebalance(Rebalance): - (success) = rebalanceParams_.swap.router.call(rebalanceParams_.swap.payload)

Informational/High/naming-convention

Parameter ArrakisV2FactoryStorage.initialize(address).owner is not in mixedCase

Variable ArrakisV2Storage._pools is not in mixedCase

Constant ArrakisV2FactoryStorage.version is not in UPPER_CASE_WITH_UNDERSCORES

Constant ArrakisV2Storage.arrakisFeeBPS is not in UPPER_CASE_WITH_UNDERSCORES

Variable ArrakisV2FactoryStorage._vaults is not in mixedCase

Variable ArrakisV2Storage._routers is not in mixedCase

Informational/High/solc-version

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

solc-0.8.13 is not recommended for deployment

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Pragma version0.8.13 necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7

Informational/Medium/costly-loop

ArrakisV2.rebalance(Range[],Rebalance,Range[]) has costly operations inside a loop: - delete ranges[index]

ArrakisV2.rebalance(Range[],Rebalance,Range[]) has costly operations inside a loop: - ranges.pop()

Informational/Medium/similar-names

Variable Underlying.getUnderlyingBalances(PositionUnderlying).tokensOwed0 is too similar to Underlying.getUnderlyingBalances(PositionUnderlying).tokensOwed1

Variable ArrakisV2.uniswapV3MintCallback(uint256,uint256,bytes).amount0Owed_ is too similar to ArrakisV2.uniswapV3MintCallback(uint256,uint256,bytes).amount1Owed_

Variable IArrakisV2Resolver.getMintAmounts(IArrakisV2,uint256,uint256).amount0Max_ is too similar to ArrakisV2Resolver.getMintAmounts(IArrakisV2,uint256,uint256).amount1Max_

Variable ArrakisV2._rebalance(Rebalance).balance0After is too similar to ArrakisV2._rebalance(Rebalance).balance1After

Variable ArrakisV2Resolver.getMintAmounts(IArrakisV2,uint256,uint256).amount0Max_ is too similar to IArrakisV2Resolver.getMintAmounts(IArrakisV2,uint256,uint256).amount1Max_

Variable Underlying._getFeesEarned(GetFeesPayload).feeGrowthOutside0Lower is too similar to Underlying._getFeesEarned(GetFeesPayload).feeGrowthOutside1Lower

Variable ArrakisV2._rebalance(Rebalance).aggregator0 is too similar to ArrakisV2._rebalance(Rebalance).aggregator1

Variable Underlying._getFeesEarned(GetFeesPayload).feeGrowthOutside0Upper is too similar to Underlying._getFeesEarned(GetFeesPayload).feeGrowthOutside1Upper

Variable Underlying.computeMintAmounts(uint256,uint256,uint256,uint256,uint256).amount0Max_ is too similar to Underlying.computeMintAmounts(uint256,uint256,uint256,uint256,uint256).amount1Max_

Variable ArrakisV2Resolver.getMintAmounts(IArrakisV2,uint256,uint256).amount0Max_ is too similar to ArrakisV2Resolver.getMintAmounts(IArrakisV2,uint256,uint256).amount1Max_

Variable Underlying.computeMintAmounts(uint256,uint256,uint256,uint256,uint256).amount0Mint is too similar to Underlying.computeMintAmounts(uint256,uint256,uint256,uint256,uint256).amount1Mint

Variable Underlying.getUnderlyingBalances(PositionUnderlying).amount0Current is too similar to Underlying.getUnderlyingBalances(PositionUnderlying).amount1Current

Variable ArrakisV2Storage.managerBalance0 is too similar to ArrakisV2Storage.managerBalance1

Variable IArrakisV2Resolver.getMintAmounts(IArrakisV2,uint256,uint256).amount0Max_ is too similar to IArrakisV2Resolver.getMintAmounts(IArrakisV2,uint256,uint256).amount1Max_

Variable Underlying.getUnderlyingBalances(PositionUnderlying).feeGrowthInside0Last is too similar to Underlying.getUnderlyingBalances(PositionUnderlying).feeGrowthInside1Last

Variable ArrakisV2Storage.arrakisBalance0 is too similar to ArrakisV2Storage.arrakisBalance1

Variable ArrakisV2Storage.addPools(uint24[],address,address).token0Addr is too similar to ArrakisV2Storage.addPools(uint24[],address,address).token1Addr

Variable ArrakisV2._rebalance(Rebalance).balance0Before is too similar to ArrakisV2._rebalance(Rebalance).balance1Before

Informational/Medium/too-many-digits

Underlying._computeFeesEarned(ComputeFeesPayload) uses literals with too many digits: - fee = FullMath.mulDiv(computeFees_.liquidity,feeGrowthInside - computeFees_.feeGrowthInsideLast,0x100000000000000000000000000000000)

Low/High/variable-scope

Variable 'ArrakisV2Factory._preDeploy(InitializePayload,bool).result' in ArrakisV2Factory._preDeploy(InitializePayload,bool) potentially used before declaration: name = result

Variable 'ArrakisV2.rebalance(Range[],Rebalance,Range[]).exist' in ArrakisV2.rebalance(Range[],Rebalance,Range[]) potentially used before declaration: (exist,index) = Position.rangeExist(ranges,rangesToRemove_[i_scope_0])

Variable 'Manager.getManagerFeeBPS(IManager).feeBPS' in Manager.getManagerFeeBPS(IManager) potentially used before declaration: feeBPS

Low/Medium/calls-loop

Underlying._getFeesEarned(GetFeesPayload) has external calls inside a loop: (feeGrowthOutside0Lower,feeGrowthOutside1Lower) = feeInfo_.pool.ticks(feeInfo_.lowerTick)

ArrakisV2Resolver.standardRebalance(RangeWeight[],IArrakisV2) has external calls inside a loop: (sqrtPriceX96) = IUniswapV3Pool(vaultV2_.factory().getPool(token0Addr,token1Addr,rangeWeight.range.feeTier)).slot0()

ArrakisV2Resolver.standardBurnParams(uint256,IArrakisV2) has external calls inside a loop: (liquidity,None,None,None,None) = IUniswapV3Pool(vaultV2_.factory().getPool(address(vaultV2_.token0()),address(vaultV2_.token1()),ranges[i].feeTier)).positions(Position.getPositionId(address(vaultV2_),ranges[i].lowerTick,ranges[i].upperTick))

ArrakisV2FactoryStorage.upgradeVaults(address[]) has external calls inside a loop: ITransparentUpgradeableProxy(vaults_[i]).upgradeTo(arrakisV2Beacon.implementation())

ArrakisV2Helper._getAmountsAndFeesFromLiquidity(address,address,Range,address) has external calls inside a loop: pool = IUniswapV3Pool(factory.getPool(token0_,token1_,range_.feeTier))

Underlying.underlying(RangeData) has external calls inside a loop: (sqrtPriceX96,tick) = underlying_.pool.slot0()

ArrakisV2Helper.ranges(IArrakisV2) has external calls inside a loop: rgs[i] = vault_.ranges(i)

ArrakisV2Helper.ranges(IArrakisV2) has external calls inside a loop: vault_.ranges(index)

ArrakisV2Resolver.standardRebalance(RangeWeight[],IArrakisV2) has external calls inside a loop: (liquidity,None,None,None,None) = IUniswapV3Pool(vaultV2_.factory().getPool(token0Addr,token1Addr,ranges[i].feeTier)).positions(Position.getPositionId(address(vaultV2_),ranges[i].lowerTick,ranges[i].upperTick))

Underlying._getFeesEarned(GetFeesPayload) has external calls inside a loop: payload = ComputeFeesPayload(feeInfo_.feeGrowthInside0Last,feeGrowthOutside0Lower,feeGrowthOutside0Upper,feeInfo_.pool.feeGrowthGlobal0X128(),feeInfo_.pool,feeInfo_.liquidity,feeInfo_.tick,feeInfo_.lowerTick,feeInfo_.upperTick)

Underlying._getFeesEarned(GetFeesPayload) has external calls inside a loop: (feeGrowthOutside0Upper,feeGrowthOutside1Upper) = feeInfo_.pool.ticks(feeInfo_.upperTick)

ArrakisV2._withdraw(IUniswapV3Pool,int24,int24,uint128) has external calls inside a loop: (withdraw.burn0,withdraw.burn1) = pool_.burn(lowerTick_,upperTick_,liquidity_)

Underlying.getUnderlyingBalances(PositionUnderlying) has external calls inside a loop: (liquidity,feeGrowthInside0Last,feeGrowthInside1Last,tokensOwed0,tokensOwed1) = positionUnderlying_.pool.positions(positionUnderlying_.positionId)

ArrakisV2.rebalance(Range[],Rebalance,Range[]) has external calls inside a loop: pool = factory.getPool(address(token0),address(token1),ranges_[i].feeTier)

ArrakisV2._withdraw(IUniswapV3Pool,int24,int24,uint128) has external calls inside a loop: (collect0,collect1) = pool_.collect(address(this),lowerTick_,upperTick_,type()(uint128).max,type()(uint128).max)

ArrakisV2FactoryStorage.makeVaultsImmutable(address[]) has external calls inside a loop: ITransparentUpgradeableProxy(vaults_[i]).changeAdmin(address(1))

Underlying._getFeesEarned(GetFeesPayload) has external calls inside a loop: payload.feeGrowthGlobal = feeInfo_.pool.feeGrowthGlobal1X128()

Underlying.totalUnderlyingWithFees(UnderlyingPayload) has external calls inside a loop: pool = IUniswapV3Pool(underlyingPayload_.factory.getPool(underlyingPayload_.token0,underlyingPayload_.token1,underlyingPayload_.ranges[i].feeTier))

ArrakisV2.burn(BurnLiquidity[],uint256,address) has external calls inside a loop: withdraw = withdraw(IUniswapV3Pool(factory.getPool(address(token0),address(token1),burns[i].range.feeTier)),burns_[i].range.lowerTick,burns_[i].range.upperTick,burns_[i].liquidity)

ArrakisV2FactoryStorage.upgradeVaultsAndCall(address[],bytes[]) has external calls inside a loop: ITransparentUpgradeableProxy(vaults_[i]).upgradeToAndCall(arrakisV2Beacon.implementation(),datas_[i])

Low/Medium/missing-zero-check

ArrakisV2FactoryStorage.getProxyImplementation(address).proxy lacks a zero-check on : - (success,returndata) = proxy.staticcall(0x5c60da1b)

ArrakisV2Storage.setRestrictedMint(address).minter_ lacks a zero-check on : - LogRestrictedMint(restrictedMint = minter_)

ArrakisV2FactoryStorage.getProxyAdmin(address).proxy lacks a zero-check on : - (success,returndata) = proxy.staticcall(0xf851a440)

Low/Medium/reentrancy-events

Reentrancy in ArrakisV2.withdrawManagerBalance(): External calls: - token0.safeTransfer(address(manager),amount0) - token1.safeTransfer(address(manager),amount1) Event emitted after the call(s): - LogWithdrawManagerBalance(amount0,amount1)

Reentrancy in ArrakisV2._rebalance(Rebalance): External calls: - token0.safeApprove(address(rebalanceParams_.swap.router),0) - token1.safeApprove(address(rebalanceParams_.swap.router),0) - token0.safeApprove(address(rebalanceParams_.swap.router),balance0Before) - token1.safeApprove(address(rebalanceParams_.swap.router),balance1Before) - (success) = rebalanceParams_.swap.router.call(rebalanceParams_.swap.payload) Event emitted after the call(s): - LogRebalance(rebalanceParams_)

Reentrancy in ArrakisV2.mint(uint256,address): External calls: - token0.safeTransferFrom(msg.sender,me,amount0) - token1.safeTransferFrom(msg.sender,me,amount1) Event emitted after the call(s): - LogMint(receiver_,mintAmount_,amount0,amount1) - LogUncollectedFees(fee0,fee1)

Reentrancy in ArrakisV2Factory.deployVault(InitializePayload,bool): External calls: - vault = preDeploy(params,isBeacon_) - vault = address(new BeaconProxy(address(arrakisV2Beacon),data)) - vault = address(new TransparentUpgradeableProxy(arrakisV2Beacon.implementation(),address(this),data)) Event emitted after the call(s): - VaultCreated(msg.sender,vault)

Reentrancy in ArrakisV2.burn(BurnLiquidity[],uint256,address): External calls: - token0.safeTransfer(receiver_,amount0) - token1.safeTransfer(receiver_,amount1) Event emitted after the call(s): - LPBurned(msg.sender,total.burn0,total.burn1) - LogBurn(receiver_,burnAmount_,amount0,amount1) - LogCollectedFees(total.fee0,total.fee1) - LogUncollectedFees(underlying.fee0,underlying.fee1)

Reentrancy in ArrakisV2.withdrawArrakisBalance(): External calls: - token0.safeTransfer(arrakisTreasury,amount0) - token1.safeTransfer(arrakisTreasury,amount1) Event emitted after the call(s): - LogWithdrawArrakisBalance(amount0,amount1)

Reentrancy in ArrakisV2.burn(BurnLiquidity[],uint256,address): External calls: - token0.safeTransfer(receiver_,amount0) - token1.safeTransfer(receiver_,amount1) Event emitted after the call(s): - LogBurn(receiver_,burnAmount_,amount0,amount1)

Medium/Medium/divide-before-multiply

ArrakisV2Factory._uint2str(uint256) performs a multiplication on the result of a division: -temp = (48 + uint8(_i - (_i / 10) * 10))

Medium/Medium/uninitialized-local

ArrakisV2Factory._preDeploy(InitializePayload,bool).result is a local variable never initialized

ArrakisV2.rebalance(Range[],Rebalance,Range[]).exist_scope_1 is a local variable never initialized

ArrakisV2.burn(BurnLiquidity[],uint256,address).underlying is a local variable never initialized

ArrakisV2Resolver.standardBurnParams(uint256,IArrakisV2).underlying is a local variable never initialized

Manager.getManagerFeeBPS(IManager).feeBPS is a local variable never initialized

ArrakisV2Resolver._requireWeightUnder100(RangeWeight[]).i is a local variable never initialized

ArrakisV2Resolver.standardRebalance(RangeWeight[],IArrakisV2).numberOfPosLiq is a local variable never initialized

ArrakisV2Resolver.standardRebalance(RangeWeight[],IArrakisV2).j is a local variable never initialized

Medium/Medium/unused-return

ArrakisV2Storage.blacklistRouters(address[]) ignores return value by routers.remove(routers[i])

ArrakisV2Factory._preDeploy(InitializePayload,bool) ignores return value by this.getTokenName(token0,token1)

ArrakisV2Storage._whitelistRouters(address[]) ignores return value by routers.add(routers[i])

ArrakisV2Storage._addPools(uint24[],address,address) ignores return value by _pools.add(pool)

ArrakisV2Helper.ranges(IArrakisV2) ignores return value by vault_.ranges(index)

ArrakisV2Factory.deployVault(InitializePayload,bool) ignores return value by _vaults.add(vault)

Manager.getManagerFeeBPS(IManager) ignores return value by manager_.managerFeeBPS()

ArrakisV2Storage.removePools(address[]) ignores return value by pools.remove(pools[i])

Optimization/High/external-function

getProxyImplementation(address) should be declared external: - ArrakisV2FactoryStorage.getProxyImplementation(address)

getProxyAdmin(address) should be declared external: - ArrakisV2FactoryStorage.getProxyAdmin(address)

vaults() should be declared external: - ArrakisV2Factory.vaults()

getAmountsForLiquidity(int24,int24,int24,uint128) should be declared external: - ArrakisV2Resolver.getAmountsForLiquidity(int24,int24,int24,uint128)

requireNotActiveRange(IUniswapV3Factory,address,address,address,Range) should be declared external: - Position.requireNotActiveRange(IUniswapV3Factory,address,address,address,Range)

validateTickSpacing(address,Range) should be declared external: - Pool.validateTickSpacing(address,Range)

rangeExist(Range[],Range) should be declared external: - Position.rangeExist(Range[],Range)

getManagerFeeBPS(IManager) should be declared external: - Manager.getManagerFeeBPS(IManager)

totalUnderlyingWithFees(UnderlyingPayload) should be declared external: - Underlying.totalUnderlyingWithFees(UnderlyingPayload)

computeMintAmounts(uint256,uint256,uint256,uint256,uint256) should be declared external: - Underlying.computeMintAmounts(uint256,uint256,uint256,uint256,uint256)

8. Appendix C. Tests

v2-core

Tests result

```
45 passing (56s)
```

Tests coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts\	97.33	53.57	100	97.08	
ArrakisV2.sol	97.06	52.86	100	97.14	272,286,304,377
ArrakisV2Beacon.sol	100	100	100	100	
ArrakisV2Factory.sol	96.97	50	100	97.06	113
ArrakisV2Helper.sol	100	100	100	100	
ArrakisV2Resolver.sol	96.43	58.33	100	95.08	105,106,247
contracts\abstract\	91.3	59.09	90	91.18	
ArrakisV2FactoryStorage.sol	100	50	100	100	
ArrakisV2Storage.sol	88.68	60.53	84.62	88.46	... 185,187,189
contracts\functions\	100	50	100	100	
FArrakisV2Factory.sol	100	50	100	100	
contracts\libraries\	94.12	62.5	100	94.12	
Manager.sol	66.67	100	100	66.67	12
Pool.sol	100	100	100	100	
Position.sol	100	50	100	100	
Underlying.sol	94.64	66.67	100	94.64	233,307,317

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts\structs\	100	100	100	100	
SArrakisV2.sol	100	100	100	100	
SArrakisV2Helper.sol	100	100	100	100	
All files	95.78	56.08	97.01	95.65	

v2-palm

Tests result

```
83 passing (58s)
```

Tests coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts\	100	60	100	100	
PALMManager.sol	100	50	100	100	
PALMTerms.sol	100	61.11	100	100	
contracts\abstracts\	87.88	82.43	86.79	89.93	
PALMManagerStorage.sol	91.03	86.36	87.5	91.57	... 253,254,326
PALMTermsStorage.sol	83.33	76.67	85.71	87.5	... 229,231,255
contracts\functions\	81.82	30	85.71	81.82	
FPALMTerms.sol	81.82	30	85.71	81.82	44,48
contracts\interfaces\	100	100	100	100	
IArrakisV2.sol	100	100	100	100	
IArrakisV2Beacon.sol	100	100	100	100	
IArrakisV2Factory.sol	100	100	100	100	
IArrakisV2Resolver.sol	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
IManager.sol	100	100	100	100	
IPALMManager.sol	100	100	100	100	
IPALMTerms.sol	100	100	100	100	
contracts\structs\	100	100	100	100	
SPALMManager.sol	100	100	100	100	
SPALMTerms.sol	100	100	100	100	
All files	91.44	73.08	82.72	92.58	

STATE MIND