SHERLOCK

# Security Review For Arrakis

| | |
|---|---|
| Collaborative Audit Prepared For: | **Arrakis** |
| Lead Security Expert(s): | **juaan** |
| | **xiaoming90** |
| Date Audited: | **February 24 - February 27, 2025** |

## Introduction

Arrakis is web3's trustless market making infrastructure protocol that enables running sophisticated algorithmic strategies on Uniswap V3.

## Scope

Repository: ArrakisFinance/arrakis-modular

Audited Commit: acd2c5dd97bd59de131f150ba44d0df54c3fff4c

Final Commit: a416e46fbbd2fcf95be581bdae60eca03181ce31

Files:

- src/constants/CArrakis.sol
- src/interfaces/IMigrationHelper.sol
- src/utils/MigrationHelper.sol
- src/utils/WithdrawHelper.sol

---

Repository: ArrakisFinance/v3-lib-0.8

Audited Commit: 756e1f70b5836b6232d74ce4b743b0c19dd9a077

Final Commit: 756e1f70b5836b6232d74ce4b743b0c19dd9a077

Files:

- contracts/FullMath.sol

## Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

## Issues Found

| High | Medium | Low/Info |
|---|---|---|
| 0 | 1 | 4 |

## Issues Not Fixed and Not Acknowledged

| High | Medium | Low/Info |
|---|---|---|
| 0 | 0 | 0 |

## Issue M-1: Migration can be DOS'd due to a non-unique private vault deployment salt

Source: https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/issues/42

### Summary

Anyone can frontrun a migration by deploying an identical arrakis private vault, causing the migration to revert.

### Vulnerability Detail

During the migration, the `MigrationHelper` deploys an arrakis modular private vault:

https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/blob/1bb3dff9e1c72ae246a6d24c924f72b6a16234fb/arrakis-modular/src/utils/MigrationHelper.sol#L216-L226

The salt used when deploying the contract is obtained via:

https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/blob/1bb3dff9e1c72ae246a6d24c924f72b6a16234fb/arrakis-modular/src/ArrakisMetaVaultFactory.sol#L225

The issue is that since this function is called directly from the `MigrationHelper`, `msg.sender` will be the `MigrationHelper` itself. This allows anyone else to frontrun a migration, by performing an arbitrary migration with the exact same vault creation payload, and the same salt.

Due to this, the user's migration will attempt to use Create3 to deploy a contract with the same salt and creationCode, which will revert since the contract already has been deployed.

### Impact

Migrations can be DOS'd due to the required private vault already being created by a malicious actor

### Tool Used

Manual Review

### Recommendation

Rather than directly calling `deployPrivateVault()` on the vault factory, consider executing this call on behalf of the safe smart account (using `execTransactionFromModule()`), as this would ensure that the salt used in the factory cannot be used by a frontrunner.

## Issue L-1: Module is not disabled after migration

Source: https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/issues/39

### Summary

The module is not disabled after migration due to the incorrect previous module being fetched.

### Vulnerability Detail

All enabled modules in a Safe account are tracked in a linked list.

To disable a module in Safe via the `disableModule()` function, the previous module in the modules linked list has to be provided (`prevModule`).

The codebase attempts to fetch the previous module in Line 371 via the `getModulesPaginated()` function. However, the issue is that this fetches the next module instead of the previous module.

As a result, attempts to disable the module will fail. Since the current codebase does not verify the returned success status of the `execTransactionFromModule()` function, the execution will fail silently.

https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/blob/d905b519004128e92e48e8012bdf4c3d8754092d/arrakis-modular/src/utils/MigrationHelper.sol#L371

```
File: MigrationHelper.sol
372:             (, address prevModule) = ISafe(params_.safe)
373:                 .getModulesPaginated(address(this), 1);
374: 
375:             state.payload = abi.encodeWithSelector(
376:                 ISafe.disableModule.selector,
377:                 prevModule,
378:                 address(this)
379:             );
380: 
381:             ISafe(params_.safe)
382:                 .execTransactionFromModule(
383:                 params_.safe, 0, state.payload, Operation.Call
384:             );
```

**Example** Assuming a user enables three modules (ModA, ModB, ModC) one after the other. Note the following:

```
SENTINEL_MODULES is used to traverse `modules`, so that:
1. `modules[SENTINEL_MODULES]` contains the first module
```

```
2. `modules[last_module]` points back to SENTINEL_MODULES
SENTINEL_MODULES = address(0x1)
```

If the user wants to remove Module_B, it must provide the previous node (Module_C).

If the `start` of the `getModulesPaginated` function is set to Module_B, the `modules[start]` in Line 211 will map to Module_A instead of ModuleC.

https://github.com/safe-global/safe-smart-account/blob/99d707583ff3e10ec2588cc0e99034353d2dbbd8/contracts/base/ModuleManager.sol#L157

```
File: ModuleManager.sol
203:     function getModulesPaginated(address start, uint256 pageSize) external
 ↪  view override returns (address[] memory array, address next) {
204:         if (start != SENTINEL_MODULES && !isModuleEnabled(start))
 ↪  revertWithError("GS105");
205:         if (pageSize == 0) revertWithError("GS106");
206:         // Init array with max page size
207:         array = new address[](pageSize);
208:
209:         // Populate return array
210:         uint256 moduleCount = 0;
211:         next = modules[start];
212:         while (next != address(0) && next != SENTINEL_MODULES && moduleCount <
 ↪  pageSize) {
213:             array[moduleCount] = next;
214:             next = modules[next];
215:             ++moduleCount;
216:         }
```

In the above case, disableModule() will revert due to the following check:

```
File: ModuleManager.sol
68:       if (modules[prevModule] != module) revertWithError("GS103");
```

## Impact

The `MigrationHelper` module will not be disabled after migration.

## Code Snippet

https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/blob/d905b519004128e92e48e8012bdf4c3d8754092d/arrakis-modular/src/utils/MigrationHelper.sol#L371

## Tool Used

Manual Review

## Recommendation

Consider updating the codebase to fetch the correct previous module.

## Issue L-2: `success` value returned by `execTransactionFromModule` was not validated

Source: https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/issues/40

### Summary

The `success` value returned by `execTransactionFromModule` function was not validated, causing failed transactions executed via Safe's `execTransactionFromModule` to go undetected.

### Vulnerability Detail

**Background** The codebase relies extensively on Safe's `execTransactionFromModule` to initiate transactions from the user's Safe wallet.

Following is the implementation of the Safe's `execTransactionFromModule` function. Within the function, it calls the `execute` function internally.

https://github.com/safe-global/safe-smart-account/blob/c460de82430a2b52ca9bb284190b09dc0df8e110/contracts/base/ModuleManager.sol#L61

```
File: ModuleManager.sol
    function execTransactionFromModule(
        address to,
        uint256 value,
        bytes memory data,
        Enum.Operation operation
    ) external override returns (bool success) {
        (address guard, bytes32 guardHash) = preModuleExecution(to, value, data, operation);
        success = execute(to, value, data, operation, type(uint256).max);
        postModuleExecution(guard, guardHash, success);
    }
```

An important point to note is that the `execute` function uses a low-level `call` opcode. A low-level `call` opcode will never revert the transaction on a failed call but set the `success` to 0 to indicate a failed call.

https://github.com/safe-global/safe-smart-account/blob/c460de82430a2b52ca9bb284190b09dc0df8e110/contracts/base/Executor.sol#L23

```
File: Executor.sol
    function execute(
..SNIP..
        } else {
```

```
..SNIP..
38:             assembly {
39:                 success := call(txGas, to, value, add(data, 0x20), mload(data),
 ↪  0, 0)
40:             }
```

The `success` result is eventually returned from the `execTransactionFromModule` function. Thus, it is important that the `success` result is verified consistently throughout the entire codebase to ensure that none of the transactions fail silently.

Following are instances of issues:

**Instance 1 - Disable Module Fail Silently** No check against the returned `success` result was performed when disabling the module, as shown in Line 381 below. As a result, any error while disabling the module will not be caught and fail silently.
https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/blob/1bb3dff9e1c72ae246a6d24c924f72b6a16234fb/arrakis-modular/src/utils/MigrationHelper.sol#L380

```
File: MigrationHelper.sol
375:            state.payload = abi.encodeWithSelector(
376:                ISafe.disableModule.selector,
377:                prevModule,
378:                address(this)
379:            );
380:
381:            ISafe(params_.safe)
382:                .execTransactionFromModule(
383:                params_.safe, 0, state.payload, Operation.Call
384:            );
```

**Instance 2** Line 122 and Line 145 of the `WithdrawHelper.withdraw` function does not check the returned `success` boolean. As a result, the transaction might fail silently. Following are some of the reasons why an ETH transfer can fail:

- Lack of sufficient ETH balance
- The recipient contract has no payable fallback or receive function
- Recipient contract reverts
- Gas limit too low
- Call depth exceeded/Out of gas

https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/blob/d905b519004128e92e48e8012bdf4c3d8754092d/arrakis-modular/src/utils/WithdrawHelper.sol#L122

```
File: WithdrawHelper.sol
118:            uint256 amountToTransfer =
119:                    amount0_ > balance0 ? balance0 : amount0_;
```

```
                balance0 -= amountToTransfer;
            if(token0 == NATIVE_COIN) {
                ISafe(safe_).execTransactionFromModule(
                    receiver_, amountToTransfer, "", Operation.Call
                );
..SNIP..
            if(token1 == NATIVE_COIN) {
                ISafe(safe_).execTransactionFromModule(
                    receiver_, amountToTransfer, "", Operation.Call
                );
```

## Impact

Transaction executed via Safe's `execTransactionFromModule` might fail silently.

## Code Snippet

https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/blob/1bb3dff9e1c72ae246a6d24c924f72b6a16234fb/arrakis-modular/src/utils/MigrationHelper.sol#L380

https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/blob/d905b519004128e92e48e8012bdf4c3d8754092d/arrakis-modular/src/utils/WithdrawHelper.sol#L122

## Tool Used

Manual Review

## Recommendation

Ensure that the `success` value returned by `execTransactionFromModule` is consistently validated across the codebase to avoid any edge case.

## Discussion

**xiaoming9090**

Fixed. The `success` boolean returned from `execTransactionFromModule` and `execTransactionFromModuleReturnData` functions within the `WithdrawHelper.sol` and `MigrationHelper.sol` contracts is validated.

## Issue L-3: `returnData` value was not verified

Source: https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/issues/41

### Summary

Some functions returned a `success` boolean to indicate a failure instead of reverting when encountering an error or reaching an invalid state. However, this is not verified, leading the transaction to proceed despite an error.

### Vulnerability Detail

The codebase relies extensively on Safe's `execTransactionFromModule` to initiate transactions from the user's Safe wallet.

Throughout the codebase, the current implementation only checks the `success` boolean returned from the `execTransactionFromModule` function to verify that a transaction is successful. The `success` boolean returned from `execTransactionFromModule` function only indicates whether or not the transaction has reverted. This is sufficient for functions that revert internally when encountering a failure or reaching an invalid state.

However, some functions do not revert but return `false` boolean when encountering an error. In this case, since the returned data of the `execTransactionFromModule` function was not verified, the codebase will continue to proceed, assuming all is good even if the functions return a `false` boolean to indicate a failure.

The following are all the methods being called by the `execTransactionFromModule` function within the codebase and whether the method returns a `success` boolean to indicate a successful execution.

| Method | Return `success` boolean? |
|---|---|
| PALMTerms.closeTerm | No |
| IArrakisMetaVaultPrivate.whitelistDepositors | No |
| IArrakisMetaVaultPrivate.deposit | No |
| IArrakisMetaVaultPrivate.withdraw | No |
| IArrakisStandardManager.rebalance | No |
| IArrakisStandardManager.updateVaultInfo | No |
| IWETH9.withdraw | No |

| Method | Return `success` boolean? |
|---|---|
| ISafe.disableModule | No |
| IERC20.approve | Yes |
| IERC20.transfer | Yes |

Following is the EIP-20 specification for `transfer` and `approve` showing that the functions are expected to return a `success` boolean value.

https://eips.ethereum.org/EIPS/eip-20#transfer and https://eips.ethereum.org/EIPS/eip-20#approve

```
function transfer(address _to, uint256 _value) public returns (bool success)
function approve(address _spender, uint256 _value) public returns (bool success)
```

## Impact

The codebase will continue to proceed, assuming all is good, even if the functions return a `false` boolean to indicate a failure. As a result, it might lead to an incorrect state.

## Code Snippet

https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/blob/d905b519004128e92e48e8012bdf4c3d8754092d/arrakis-modular/src/utils/WithdrawHelper.sol#L127

https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/blob/d905b519004128e92e48e8012bdf4c3d8754092d/arrakis-modular/src/utils/WithdrawHelper.sol#L150

https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/blob/d905b519004128e92e48e8012bdf4c3d8754092d/arrakis-modular/src/utils/WithdrawHelper.sol#L218

https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/blob/d905b519004128e92e48e8012bdf4c3d8754092d/arrakis-modular/src/utils/WithdrawHelper.sol#L237

## Tool Used

Manual Review

## Recommendation

For those functions (e.g., `ERC20.transfer` and `ERC20.approve`) that return `success` boolean, consider using `execTransactionFromModuleReturnData` instead of`execTransactionFromModule`. Decode the `returnData` from the `execTransactionFromModuleReturnData` function and verify that the `success` boolean returned from the affected functions are `true`.

## Discussion

**xiaoming9090**

Fixed. `returnData` is now decoded to check whether the operation is successful.

## Issue L-4: Migrations which include a rebalance payload can be temporarily DOS'd

Source: https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/issues/43

### Summary

Migrations can be DOS'd by pre-initializing the UniswapV4 pool with the same PoolKey as in the migration params

There are two ways by which this can occur (when a malicious actor frontruns the migration to initialize the UniV4 pool):

1. `params_.poolCreation.createPool` is set to true, the `MigrationHelper` attempts to initialize the new UniV4 pool, but this reverts in Uniswap's PoolManager https://github.com/Uniswap/v4-core/blob/65690c0719b3eade39a852e4c2e9ecc9af41f6e7/src/libraries/Pool.sol#L104
2. If the `createPool` param from above is set to false, the migration can still revert if it involves a rebalance

### Vulnerability Detail

In the migration flow, after creating and depositing into the new arrakis modular vault, a rebalance can occur optionally: https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/blob/1bb3dff9e1c72ae246a6d24c924f72b6a16234fb/arrakis-modular/src/utils/MigrationHelper.sol#L309-L324

If configured to do so, rebalance will add liquidity to UniswapV4: https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/blob/1bb3dff9e1c72ae246a6d24c924f72b6a16234fb/arrakis-modular/src/libraries/UniswapV4.sol#L357-L380

Then at the end, the amount of tokens deposited is checked against slippage parameters: https://github.com/sherlock-audit/2025-02-arrakis-safe-helpers/blob/1bb3dff9e1c72ae246a6d24c924f72b6a16234fb/arrakis-modular/src/abstracts/UniV4StandardModule.sol#L509-L510

The issue is that a malicious actor can frontrun a migration by initializing the UniV4 pool with the same `PoolKey`, and set the `sqrtPriceX96` to `MIN_TICK`.

This would cause the added liquidity to be single sided (only token1 liquidity will be added). This would cause the slippage checks to revert since `amount0Minted < minDeposit0_`

### Impact

Migrations which involve a rebalance call to add liquidity to the new UniV4 pool can be DOS'd

## Tool Used

Manual Review

## Recommendation

To mitigate the first potential DoS, consider checking the `PoolManager` for whether or not the UniV4 pool was initialized or not, rather than using the `params_.poolCreation.createPool` param that is passed in.

Regarding the second impact, it's recommended to disallow including a rebalance call during migration if the pool was already initialized with an unfavourable price. This can be done in the frontend, and has been included in this report since the team requested ideas on areas to be careful when constructing the migration calldata.

## Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users’ sole risk and is the users’ responsibility.