PUBLIC

Code Assessment

of the Kandel Strats Smart Contracts

April 04, 2023

Produced for

by

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help Mangrove Association (ADDMA) with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Kandel Strats according to Scope to support you in forming an opinion on their security risks.

Mangrove Association (ADDMA) implements a "buy low, sell high" market making strategy leveraging the Mangrove core system, while optimizing the capital efficiency by supplying the idle funds on AaveV3.

The most critical subjects covered in our audit are functional correctness, access control and precision of arithmetic operations. Security regarding all is generally good. Security regarding functional correctness is good as long as drying out the Aave pool on purpose, see Provoking an Aave Liquidity Crisis, is unprofitable based on the borrow and supply caps, and the flashloan fees.

The general subjects covered are code complexity, error handling, specification and gas inefficiency. Security regarding all the aforementioned subjects is good. However, documentation could be more explicit for makers since the provided arguments on creation should be meaningful but are not checked by code.

All the issues uncovered during the review have been either fixed or acknowledged.

In summary, we find that the codebase provides a satisfactory level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	1
• Code Corrected	1
High-Severity Findings	1
• Risk Accepted	1
Medium-Severity Findings	1
• Code Corrected	1
Low-Severity Findings	6
• Code Corrected	4
• Specification Changed	1
• Acknowledged	1

2 Assessment Overview

In this section we briefly describe the overall structure and scope of the engagement including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files for the `AaveKandelSeeder`, `KandelSeeder`, `AaveKandel`, `Kandel`, and `AavePooledRouter` contracts inside `mangrove-core` repository based on the documentation files. Namely this includes the following files:

Seeders:

```
strategies/offer_maker/market_making/kandel/abstract/AbstractKandelSeeder.sol
strategies/offer_maker/market_making/kandel/AaveKandelSeeder.sol
strategies/offer_maker/market_making/kandel/KandelSeeder.sol
```

Kandel:

```
strategies/offer_maker/abstract/Direct.sol
strategies/offer_maker/market_making/kandel/abstract/DirectWithBidsAndAsksDistributions.sol
strategies/offer_maker/market_making/kandel/abstract/HasIndexedBidsAndAsks.sol
strategies/offer_maker/market_making/kandel/abstract/AbstractKandel.sol
strategies/offer_maker/market_making/kandel/abstract/CoreKandel.sol
strategies/offer_maker/market_making/kandel/abstract/TradesBaseQuotePair.sol
strategies/offer_maker/market_making/kandel/abstract/GeometricKandel.sol
strategies/offer_maker/market_making/kandel/AaveKandel.sol
strategies/offer_maker/market_making/kandel/Kandel.sol
```

Aave routing system:

```
strategies/integrations/AaveV3Lender.sol
strategies/routers/integrations/HasAaveBalanceMemoizer.sol
strategies/routers/AbstractRouter.sol
strategies/routers/integrations/AavePooledRouter.sol
```

The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	20 February 2023	18ebd898c591fda33ff9b57361b14bad8f2d13a0	Initial Version
2	20 March 2023	3bff09efba82a6d55d19eeb807654833339785f1	After Initial Version

For the solidity smart contracts, the compiler version `0.8.17` was chosen.

2.1.1 Excluded from scope

Third-party dependencies and any other file not explicitly listed above are outside the scope of this review.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Mangrove Association (ADDMA) offers a market making strategy typed called Kandel that interacts with the Mangrove system. A base Kandel is implemented along with a Kandel depositing the idle funds into Aave v3.

2.2.1 Kandel

Kandel is a strategy type introduced by Mangrove Association (ADDMA) that offers the received amounts from a traded against offer in a new one that puts the market maker in a profitable situation (if traded against). Note that each Kandel is owned by the market makers and are not shared.

The implemented Kandel types operate on an `Ask` and `Bid` market. Hence, the strategy is to create `Ask` offers when a taker trades against a `Bid` offer and vice-versa. Given the notation of asks and bids, a Kandel strategy operates on a market with a `BASE` and a `QUOTE` asset. Since market makers define the ask price (asking for base), asks are traded on Mangrove's `(BASE, QUOTE)` market (in the notation of `(outbound_tkn, inbound_tkn)`). Similarly, bids are traded on Mangrove's `(QUOTE, BASE)` market.

The price progression for all implemented Kandels is a geometric one. Hence, when referring to Kandels we are referring to such geometric Kandels. The strategy operates on a "buy low sell high" basis so that the Kandel's strategy is to bid low prices for the base and ask higher prices (selling the base at a higher price and index).

The price progression is defined by a set of price indices (in a certain range defined by the market maker). A higher index indicates that the maker is selling the base asset for more quote than at a lower index. Similarly, the price increases for the quote in terms of the base when indices are reduced. Note that the only profitable scenario for the maker is to increase the index for the dual offer if the offer was a bid, and to decrease the index of the dual offer if the offer was an ask. More specifically, the price index of the dual offer is computed using a maker-defined `spread` that defines how many steps (in terms of indices) a dual offer makes in comparison to the offer (limited by the number of `pricePoints`). Hence, the dual offers index is computed as an increase or decrease by `spread` when the dual offer is of type `Ask` or `Bid` respectively (respecting the valid index range).

The exact price change is defined by a maker-defined value `ratio` so that the price that the maker sells it for is the price at which the taker sold it to the maker scaled by the `ratio` on each index.

The prices at which the Kandel operates are defined by the offers' `gives` and `wants`. Hence, their ratio should follow the price progression described above. Kandel strategies enforce at least the same liquidity is maintained (liquidity non-decreasing with the exception when a dual offer is at the boundaries). Hence, the minimum liquidity that a dual offer wants is what the offer gave (what the order wanted). Thus, the minimum that the Kandel gives is what the order gave divided by the price progression factor. Any liquidity beyond that is treated as compounded liquidity which can be reinvested. The amount that is used for reinvesting is determined by the `compoundRate` (compound rate for `Ask` and `Bid` offers may differ). The resulting `gives` is the sum of the existing `gives` at the index and the newly computed one for the dual offer. The `wants` at the dual index are then scaled up by new total `gives` so that the price is maintained at the index. The dual offer management is handled by the required `makerPosthook()` function called by the Mangrove core.

The `makerExecute()` hook implements the retrieval of funds (required by the Mangrove core system). Namely, it tries using the funds directly available to the Kandel and if in case these are insufficient and the Kandel has a router, it tries pulling the funds from the router.

The intended function for initialize new indices by the maker is `populate()` which allows creating offers with the according parameters and funding them with the bounty and the tokens. Further, makers can

adapt their offers using `populateChunk()`. Further, the maker can retract offers (`retractOffers()`) and optionally withdraw the funds accordingly (`retractAndWithdraw()`). Note that several parameters can be set outside of `populate()` with functions `setGasprice`, `setGasreq` and `setCompoundRates`. Note that Kandel strategies may change the router with `setRouter()`.

Further, the funds can be deposited with `depositFunds()` and withdrawn with `withdrawFunds()` while the bounty ETH can be withdrawn with `withdrawFromMangrove()`.

Deposits, withdrawals and the posthook may interact with the router if one is present.

Note that the simple Kandel implements the above logic.

2.2.2 AaveKandel & AavePooledRouter

AaveKandel implements the same strategy as Kandel, but the idle funds are lent on AaveV3 while they are not used by a trade on Mangrove. In order to optimize capital efficiency, multiple AaveKandel can share the same liquidity in an Aave pool, and they are then grouped under the same `reserveId`. To do so, AaveKandels are helped by the AavePooledRouter, which is responsible for the internal fund tracking of each `reserveId` in the form of shares, and AaveV3 interactions, like supplying a pool or redeeming the overlying (`aToken`) for the underlying asset.

When funds are deposited, either with `depositFunds()` or `populate()`, the funds will be pulled first to `AaveKandel`, then to `AavePooledRouter` where shares will be minted for the given tokens and `reserveId`, and finally the tokens will be supplied on AaveV3.

Upon the `makerExecute()` callback, `AaveKandel` will first check whether its own token balance can cover for `order.wants`, if it is not the case, then `AavePooledRouter` is queried for providing the missing amount. The router will check its token balance as well and if it cannot cover for the asked amount, the whole supply of `aToken` is redeemed from Aave and the liquidity is kept in the router. From there, depending on the pulling strategy (`strict` if the `reserveId` is a shared pool), the router will provide the calling `AaveKandel` with either only the asked amount (`strict`), or the whole `reserveId` token balance and the shares are burned accordingly. The choice of getting all the liquidity out of AaveV3 is justified by the fact that if mulitple offers created by `AaveKandel` are consumed during a trade, the funds are pulled out of AaveV3 only once to save gas. `AaveKandel` will return a special `makerData` upon the offer that triggered the liquidity withdrawal from the Aave pool: `IS_FIRST_PULLER`, that will be used later in the `makerPosthook` callback.

When `makerPosthook()` is triggered, the dual offer is created just like with `Kandel`. When the received `makerData` is `IS_FIRST_PULLER`, the `AavePooledRouter` will pull back the remaining `BASE` and `QUOTE` tokens, mint the corresponding amount of shares, and supply back Aave with the whole token balance of the router. If `makerData` has any other value, the remaining `BASE` and `QUOTE` tokens are simply pulled back by the router and the shares are minted.

2.2.3 KandelSeeder & AaveKandelSeeder

As the Kandels are deployed per maker, a special contract is in charge to deploy the strategies, setup the initial parameters and the admin of the strategy, as well as grouping the strategies sharing a pool under the same `reserveId`. This is done by the `sow()` function of the seeders. The `KandelSeeder` contract does exactly what is described above. `AaveKandelSeeder` does the same as `KandelSeeder`, but also has to take care of setting up the router and bind the deployed stategies to it: `AaveKandelSeeder` deploys an `AavePooledRouter` when deployed, and will bind each new `AaveKandel` it deploys to it when `sow()` is called.

2.2.4 Roles & Trust Model

- *Maker using a Kandel*: A maker should precisely know what effects of the functions could be and how to use them correctly. Otherwise, his offers could get sniped or the Kandel could break. More specifically, `populate()` and `populateChunk()` heavily rely on correct inputs, e.g, they expect a correct distribution of the asks and bids regarding the parameters of the geometric price distribution.

Similarly, the maker is expected to be careful with other functions (e.g. setting the router, usage of function that iterate over the offer IDs).

- *Mangrove core system*: The Mangrove core system is expected to work correctly and as expected.
- *Routers*: Routers are fully trusted to handle funds appropriately. The trust model and roles is given by the routers. For the Aave Router, see below for further information.
- *Aave Pooled Router admin*: The Aave router admin is expected to be trusted (see *Routers*). The admin is expected to be set to the seeder of `AaveKandel` so it can bind the deployed contracts and allow them to interact with the router. Hence, the admin of the Aave router implemented is trusted.
- *Aave manager*: The Aave manager is expected to be trusted not to block markets that are used or arbitrarily removing lenders' approval.
- *Others*: Untrusted.
- The Kandel strategies should not be used with tokens with callbacks, e.g., ERC777.

2.2.5 Changes In Version 2

1. The `L0W_VOLUME` status is returned when the offers new gives or new wants are 0 (populating offers / dual offers).
2. Kandels support now only their base and quote tokens. However, they can still approve arbitrary tokens.
3. `KandelSeeder` does not support the liquidity sharing anymore.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	1

- Provoking an Aave Liquidity Crisis Risk Accepted

Medium-Severity Findings	0
Low-Severity Findings	1

- Posthook Revert Due to Overflow in Dual Offer Computation Acknowledged

5.1 Provoking an Aave Liquidity Crisis

Security High Version 1 Risk Accepted

ISSUEIDPREFIX-001

`AaveKandel` stops providing liquidity if too much is borrowed. Generally, market makers should be aware of that. However, one can throw Aave temporarily into a liquidity crisis by

1. flashloaning the entire balance
2. borrowing the entire balance

That could allow an attacker to create a temporary liquidity crisis wrapping a sniping operation on Mangrove core for his profit.

However, note that the flashloaning attack is to some degree by the economic factor of Aave flashloans while the other operation can be achieved even for free by taking out free flashloan for collateral assets on other protocols. Such attacks are however limited by the borrow caps that Aave can impose. Namely, the attack can be carried out as long as the Aave's available capital after the borrow cap is reached is less than an offer's promised amount. Note that 1. and 2. can be combined to reduce the total cost of an attack with 1. so that the limit set by the borrow caps can be bypassed.

Risk accepted:

Mangrove Association (ADDMA) replied:

```
1/ deploy 80 offers from AAVE kandel (WETH,USDC) on mangrove

Note 1: 80 offers is the limit beyond which one cannot collect all failing offers because of stack overflow
Note 2: WETH has borrow cap on AAVE so the attack has to be on offers that have USDC outbound
```

```
2/ attacker supplies enough DAI on AAVE to be able to borrow the whole supply of USDC

Note 1: the script mocks up a flashloan of DAI to obtain enough collateral. There is currently no real way to
flashloan DAIs on polygon. In the overall cost of the attack we add 400K gas as an estimate of the flashloan
cost plus the cost of repaying the borrow on AAVE which is not scripted here (AAVE on polygon still does not
allow repay and borrow on the same block, although ethereum deployment does). The attack using AAVE native
flashloan has also been tested but result in a prohibitive cost for the attacker (around 1000 USD worth of fees).

Note 2: there is currently a supply cap on AAVE for DAI which is just enough to do this, but supplying a bit too
much DAI actually reverts

3/ attacker borrows USDC supply and triggers a market order

4/ all 80 aave kandel offers trigger a failure cascade and the bounty is sent to the attacker
Assuming a tx.gasprice == Mangrove's gasprice at 90 gwei, we get:

   • Attack collects 0.84691197 matics for 9 936 879 gas units
   • cost of the attack: 0.89431911 native tokens

Conclusion:

  - Under favorable gas conditions for the attacker, drying up the AAVE pool can be profitable (when tx.gasprice is
    significantly lower than Mangrove's). However the profit is quite low compared to traditional flashloan attacks
    and not iterable: failing Aave Kandel's offer do not repost themselves. Yet the attack would at least be griefing
    users as it would result in losing provision and having their offers unpublished from Mangrove.

  - We believe the best protection relies on launching AAVE Kandel Strats on markets that have a borrow cap,
    which would force the attacker of going through the costly AAVE flashloan mechanism to bypass the cap.

  - The above script is included in the test AaveKandel.t.sol (test_liquidity_borrow_marketOrder_attack)
```

5.2 Posthook Revert Due to Overflow in Dual Offer Computation

Design | Low | Version 1 | Acknowledged

ISSUEIDPREFIX-002

The maker posthook of Kandels can revert due to a potential overflow in `dualWantsGivesOfOffer()` in extreme scenarios. Consider the following example:

- Assume that the previously computed `gives` is equal to `2**96-1`.
- Assume that `r` is equal to `(2*10**5)**8 = 2**8 * 10**40`.
- Hence, the first computed `givesR` will be `(2**104-2**8) * 10**40`, which needs 237 bits.
- In the `else` branch, assume that `order.offer.wants()` is small, e.g. `1`, the updated `givesR` now needs 256 bits
- Hence, with `offerGives >= 2**19`, the computed `wants` will be `> 2**256` which leads to a reverting overflow.

Ultimately, the posthook for small offers could revert.

Acknowledged:

Mangrove Association (ADDMA) replied:

```
The scenario that yields an overflow occurs with unlikely values. The outcome of the overflow
is to make offer logic's posthook fail and hence entails no penalty for maker.
```

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	1

- Depositing ATokens Allows Stealing Them Code Corrected

High-Severity Findings	0

Medium-Severity Findings	1

- Wrong Dual Price Computation on Crossing Boundaries Code Corrected

Low-Severity Findings	5

- Balance in Router Can Be Present Specification Changed
- More Precision Can Be Used for wants Code Corrected
- Reserve Balance Does Not Include Kandel's Balance Code Corrected
- Tokens Without Return Values Code Corrected
- AaveKandelSeeder Missing Active Market Check Code Corrected

6.1 Depositing ATokens Allows Stealing Them

Security Critical Version 1 Code Corrected

ISSUEIDPREFIX-007

The Aave v3 pooled router, allows arbitrary tokens. Tokens that are not supported by Aave are kept in the router contract directly and treated equivalent to buffered tokens coming from Aave withdrawals. However, depositing aTokens leads to accounting issues, treating the aTokens received from actual supply operations as donations and, hence, buffered tokens.

Consider the following scenario:

1. 1M aDAI held by the router (DAI was supplied).
2. Attacker creates Kandel and uses `depositFunds()` to supply 1 aDAI.
3. Since not shares exist for aDAI, the `INIT_MINT` will be minted. The supply fails but the code does not revert since `noRevert` is `true`.
4. The attacker now uses `withdrawFunds()` to redeem `aDAI.balanceOf(router)` aDAI.
5. The router's withdraw function realizes that there is enough buffered amount of aDAI. Hence, `toRedeem` is 0.
6. `_burnShares()` first computes the shares to burn. Since there is no overlying for aDAI, the input amount will match the aDAI balance of the contract. Hence, the total shares for aDAI will be burned which are equal to `INIT_MINT` and thus equal to the attacker's shares. All shares are burned since the attacker is the only holder.
7. `toRedeem` is 0. Hence, withdrawing from Aave will not be tried. All the aDAI balance is sent to the attacker.

Ultimately, all funds in the Aave pooled router could be at risk.

Code corrected:

On deployment, the `AaveKandel` tries to call `UNDERLYING_ASSET_ADDRESS()` on the base and the quote token. In case staticcall does not revert, the token is classified as an aToken and the deployment will revert. In case the staticcall reverts, the error is caught and execution proceeds. Note that the staticcall could revert without the error message if the base or quote token are EOAs.

Further, only the base and quote tokens can be deposited to and withdrawn from the `AaveKandel`.

Both restrictions combined, prevent aTokens from being deposited to the router from Kandels.

6.2 Wrong Dual Price Computation on Crossing Boundaries

Correctness **Medium** **Version 1** **Code Corrected**

ISSUEIDPREFIX-013

When the dual price is computed, the function `GeometricKandel.dualWantsGivesOfOffer` is not aware of how many `steps` the price should move to stay within the defined boundaries and will always move it by `ratio**spread`. Whenever `steps <= spread` holds, this can lead to prices that are off by factors of `ratio` on the boundary of the indices, thus breaking the assumption that each index should be at a distance `ratio` from its neighbors.

Consider the following example:

1. The following setup is active:

```
pricePoints = 6
spread = 3
compoundRate = p = 10 ** PRECISION
```

2. A trade occurs against a `Bid` at `index = 4` so that the dual offer parameters are (note that since the `pricePoints` limit of 5 is exceeded, it is set to the maximum value allowed):

```
virtual_dual_index = 7
=> dual_index = 5

dualOffer = Ask
```

   Hence, we jump by only 1 index and expect a price update of `ratio / p`. Hence, the expected price is the expected price at index 5

```
expected_wants / expected_gives = (order.wants * ratio**1 / p**1) / (order.gives)
```

3. The `gives` of the dual offer are computed (assume now that at index 5 `gives` had been `0`):

```
new_gives = order.gives * compoundRate * ratio**spread / (ratio**spread * p)
<=> new_gives = order.gives
```

4. The `wants` of the dual offer are computed:

```
new_wants = order.wants * new_gives * ratio**spread / (order.gives * p**spread)
<=> new_wants = order.wants * order.gives * ratio**spread / (order.gives * p**spread)
<=> new_wants = order.wants * ratio**spread / p**spread
```

5. The price at index 5 is now:

```
new_wants / new_gives = (order.wants * ratio**spread / p**spread) / (order.gives)
```

Note that the new price at index 5 is distinct from the expected price.

Code corrected:

The `spread` applied to the ratio is now updated by `GeometricKandel.transportDestination` for the price to stay within bounds.

6.3 Balance in Router Can Be Present

Correctness | Low | Version 1 | Specification Changed

ISSUEIDPREFIX-012

Note that the documentation of `AavePooledRouter.__pull__()` specifies:

```
///@dev outside a market order (i.e if `__pull__` is not called during offer logic's execution) the `token` balance of this router should be empty.
/// This may not be the case when a "donation" occurred to this contract
/// If the donation is large enough to cover the pull request we use the donation funds
```

However, note that this is not necessarily the case since the posthook of the first puller could revert and leave the funds inside the router without pushing them out. However, after the next execution or the next deposit, the funds should be moved to Aave if everything works correctly.

Note that an attacker could force a balance into the router without donating to it by

- increasing the total supply on Aave such that the supply cap makes the first puller's flushing revert.
- trading against an attacker-strategy-owned Kandel and then trading against the attacker strategy that changes the router of the Aave Kandel so that the Aave Kandel does not push and supply on the Aave router.

Specification Changed:

The comment has been adapted.

6.4 More Precision Can Be Used for `wants`

Design | Low | Version 1 | Code Corrected

ISSUEIDPREFIX-009

When computing the `wants` amount of the dual offer, full precision is used when `uint160(givesR) == givesR || order.wants < 2 ** 18`, but the second condition can be relaxed to be `order.wants < 2 ** 19` to allow full precision more often.

Code corrected:

The second condition has been relaxed to accept values `< 2 ** 19`.

6.5 Reserve Balance Does Not Include Kandel's Balance

Correctness Low Version 1 Code Corrected

ISSUEIDPREFIX-011

The `reserveBalance()` is the available balance for a strategy of an offered token. Note that Direct strategies try to use the local balance always first, see function `__get__()`.

```
uint amount_ = IERC20(order.outbound_tkn).balanceOf(address(this));
if (amount_ >= amount) {
  return 0;
}
```

However, the AaveKandel does not account for the local balance (potentially received through donations) in `reserveBalance()`.

Code corrected:

The function `AaveKandel.reserveBalance()` has been updated to take its own balance into account.

6.6 Tokens Without Return Values

Design Low Version 1 Code Corrected

ISSUEIDPREFIX-008

The Mangrove core system and the transfer libraries used in Kandel handle tokens without return values on transfers. However, for approvals return values are always expected. Note that this is not always the case due to tokens implementing the ERC-20 standard incorrectly (e.g., USDT).

Code corrected:

The `approve()` function handles now the case where no return value exists.

6.7 `AaveKandelSeeder` Missing Active Market Check

Design Low Version 1 Code Corrected

ISSUEIDPREFIX-006

Only the `(BASE,QUOTE)` market is checked to be active on Mangrove, but the `(QUOTE,BASE)` base market should also be checked as it is used as well. If the inverse market is inactive, initial `Ask` and the `Bid` dual offers will fail to be posted.

Code corrected:

The two markets are now checked to be active on Mangrove in `AbstractKandelSeeder.sow()`.

6.8 Explicit Variable Visibility

Informational | Version 1 | Code Corrected

ISSUEIDPREFIX-005

Several variables have undeclared visibility which results in variables being `internal`. Note that clear specifying clear visibility can help maintain code.

Code corrected:

Variables have now explicit visibility.

6.9 Gas Inefficiencies

Informational | Version 1 | Code Corrected

ISSUEIDPREFIX-004

1. The internal function `DirectWithBidsAndAsksDistribution.populateChunk` initializes `i` to `0`, this is a redundant assignment.
2. When using `tokenPairOfOfferType()` with both offer types in a function, e.g., `DirectWithBidsAndAsksDistribution.populateChunk` or `DirectWithBidsAndAsksDistribution.retractOffers`, one of the function calls can be avoided by assigning the inverted token pair.
3. In the function `CoreKandel.retractAndWithdraw` the modifier `onlyAdmin` is unnecessary

Code corrected:

For 3, the modifier was not removed by choice.

6.10 NatSpec

Informational | Version 1 | Specification Changed

ISSUEIDPREFIX-010

The NatSpec documentation is extensive. However, there at several places the NatSpec is missing or incomplete.

For `_supply`, `checkAsset`, `_totalBalance` or `balanceOfReserve` the return value is undocumented. For `_sharesOfAmount`, `mintShares` or `_burnShares` an argument is undocumented. `depositFunds` or `withdrawFunds` do not have any NatSpec.

Note that the examples are an incomplete list of lacking NatSpec documentation.

Documentation changed:

The documentation has been adapted.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Magic Values

Informational **Version 1** **Acknowledged**

ISSUEIDPREFIX-003

The use of magic numbers in the codebase is not recommended, they should be replaced by variables with a self-explanatory name. Examples are:

- `"mgv/writeOffer/density/tooLow"`
- `"mgv/tradeSuccess"`

Acknowledged:

Mangrove Association (ADDMA) replied:

> since these magic values are part of mangrove’s specification, we assume they won’t change.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Maker Should Oversupply Due to Aave Being off by 1

Note Version 1

The maker should oversupply `AaveKandel` by some WEI to account for Aave internal loss of precision, that can lead the token amount to be off by 1 on redemption, as it could make the trade revert if they are the only one to use the Aave pool from a given `AavePooledRouter`.

8.2 Supplying Caps Not Considered

Note Version 1

Aave V3 has supply caps. However, these are not considered when supplying. Hence, supplying could fail so that tokens are treated as buffered.

8.3 `AaveKandelSeeder` Missing Existence Check of Pool for Asset

Note Version 1

No check is done on strategy deployment for a pool of `BASE` or `QUOTE` on AaveV3. If such pools cannot be supplied, the `AaveKandel` strategy can be deployed but there will be no yield from the deposits to the router.