PUBLIC

Code Assessment

of the Mangrove Strategies Smart Contracts

November 16, 2023

Produced for

by

Contents

1 Executive Summary

Dear all,

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

Mangrove Association updated the existing strategies Mangrove Order, implementing Good-till-cancelled and Fill-or-kill orders, and Kandel, a "buy low, sell high" market-making strategy that leverages the Mangrove core system, while optimizing the capital efficiency by supplying the idle funds on AaveV3. The code was mainly adapted for compatibility with the changes made in the core. Additionally, the changes include some simplifications.

The most critical subjects covered in our audit are functional correctness, access control, absence of reentrancy possibilities, handling of funds 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, unit testing, documentation, specification, gas efficiency, trustworthiness and error handling. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a good 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	0
High-Severity Findings	1
• Risk Accepted	1
Medium-Severity Findings	0
Low-Severity Findings	3
• Code Corrected	1
• 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 inside the Mangrove Strategies repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	30 October 2023	220dd1b6c95451278dd095a32a71825394a9530d	Initial Version
2	15 November 2023	6d79993939f1e585fcecafd3aadd898fd9de03e3	After fixes

For the solidity smart contracts, the compiler version `0.8.20` was chosen.

The following files are in scope:

```
./strategies/routers/abstract/AbstractRouter.sol
./strategies/routers/integrations/HasAaveBalanceMemoizer.sol
./strategies/routers/integrations/AavePooledRouter.sol
./strategies/routers/SimpleRouter.sol
./strategies/MangroveOffer.sol
./strategies/MangroveOrder.sol
./strategies/utils/AccessControlled.sol
./strategies/integrations/AaveV3Lender.sol
./strategies/offer_forwarder/abstract/Forwarder.sol
./strategies/offer_maker/abstract/Direct.sol
./strategies/offer_maker/market_making/kandel/abstract/TradesBaseQuotePair.sol
./strategies/offer_maker/market_making/kandel/abstract/CoreKandel.sol
./strategies/offer_maker/market_making/kandel/abstract/GeometricKandel.sol
./strategies/offer_maker/market_making/kandel/abstract/DirectWithBidsAndAsksDistribution.sol
./strategies/offer_maker/market_making/kandel/abstract/KandelLib.sol
./strategies/offer_maker/market_making/kandel/abstract/HasIndexedBidsAndAsks.sol
./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
./strategies/offer_maker/market_making/kandel/Kandel.sol
./strategies/offer_maker/market_making/kandel/AaveKandel.sol
./strategies/vendor/aave/v3/IPoolAddressesProvider.sol
./strategies/vendor/aave/v3/IRewardsControllerIsh.sol
./strategies/vendor/aave/v3/IPool.sol
./strategies/vendor/aave/v3/DataTypes.sol
./strategies/interfaces/IForwarder.sol
./strategies/interfaces/IOfferLogic.sol
./strategies/interfaces/IOrderLogic.sol
./strategies/interfaces/ILiquidityProvider.sol
```

2.1.1 Excluded from scope

All other files are not in scope. The core system is not in scope. External protocols are not in scope. For Kandel, it is expected that users are aware of what the functions will do. A bad setup could be created. However, given the trust model, we put such bad setups out of scope (e.g. using the `populate()` function can create arbitrary bad distributions).

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.

For a detailed system overview of the Mangrove Order and Kandel strategies, refer to the respective previous Mangrove Order audit report and the previous Kandel audit report covering the both strategies.

The following overview will focus on the parts that changed in the updated codebase. The most significant changes are a consequence of the changes made to the core after the above reports were published. Additionally, there are some simplifications and refactorings in the Kandel strategies.

Mangrove Association offers peripheral market maker contracts, called strategies, for interaction with Mangrove core system. The contracts stand as an intermediate component between the end-users and the Mangrove core system. More specifically, the strategies implement the market maker interface defined in the core along with some additional functionality for managing offers made. Both implemented strategies follow roughly the same processes. For example, if their offers had been filled only partially, they will repost the residual offers on the Mangrove exchange. Yet, the strategies differ significantly. Note that strategies may have routers which are also described in the above audit reports.

The `MangroveOrder` strategy is a shared strategy that allows users to take offers on the Mangrove exchange, taking a role as a taker in the context of Mangrove core. The order must be either fully filled or a resting offer is posted on the opposite market (subject to parameterization). Note that the resting offer may be posted with an expiry time so that a good-till-cancelled mode is implemented. For further details, please consult the audit report mentioned above. Besides the changes in relation to the core, `MangroveOrder` did not undergo any significant changes.

Kandel strategies are market maker contracts owned by one end-user. They create bid-ask spreads on two opposite semi-books with a geometric price progression so that the market makers follow a "buy low sell high" strategy. The simple `Kandel` keeps the funds idle while the `AaveKandel` invests the funds into Aave in a batched manner through the corresponding Aave router.

Besides the changes regarding the modifications in the core, the Kandel strategies have been simplified to remove the compounding factor introduced. Now, they will reinvest the full amounts instead of (optionally only) a portion. Additionally, Kandels received the functionality to create the offer distribution directly on-chain so that the created bid-ask spread can be instantiated on the Mangrove core system. Further, note that the Kandel strategies did undergo some additional refactoring.

2.2.1 Trust Model

The trust model is the same as in the previous reports. Please use them for reference. However, note that the trust model in relation to the core system is according to the core reports.

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

- Unsafe Casts for Geometric Kandels Acknowledged

5.1 Provoking an Aave Liquidity Crisis

Security High Version 1 Risk Accepted

CS-MGVSTRATS-004

`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 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 Unsafe Casts for Geometric Kandels

Correctness **Low** **Version 1** **Acknowledged**

CS-MGVSTRATS-003

`KandelLib.createGeometricDistribution()` performs several unsafe casts. Namely, there are casts from `uint` to `int` for `_baseQuoteTickOffset` and indices.

The unsafe casts for `_baseQuoteTickOffset` may create problems and wrong results for the usage of `GeometricKandel.createDistribution()`. Additionally, the parameters specified are not validated (e.g. price points could be less than or equal to the step size or they could be too high). However, the other functions defined ensure that the reasonable restrictions for the geometric parameters are enforced.

The casting of indexes, however, may even create problems for the populating functions. Consider, the following.

```
int tick = -(Tick.unwrap(baseQuoteTickIndex0) + int(_baseQuoteTickOffset) * int(index));
```

Assume that `index == 2**256-1-x`, then the cast will return `-1-x`. Ultimately, a positive tick will be turned into a negative one, only to turn it back into a positive one. That positive tick will be used for bids. The following will become the tick for the bid in such a scenario.

```
int tick = Tick.unwrap(baseQuoteTickIndex0) + int(_baseQuoteTickOffset) * (1+x);
```

As a consequence, the bid prices increase with increasing indices (worse for market makers) instead of decreasing. Hence, the bids would bid a too high price.

Note that indices can reach that size (e.g. `from` is high).

Ultimately, the behaviour is undefined and may create bad setups.

Acknowledged:

Mangrove Association acknowledged the behaviour and indicates that these parameters should be validated off-chain similar to other ones:

```
In general parameters are considered validated off-chain, e.g., through simulation
before passing to the contracts, so it is up to the caller to ensure correct usage.
```

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	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	2

- Lack of Router Sanity Check Code Corrected
- Specification Mismatch Specification Changed

Informational Findings	2

- Gas Optimisations Code Corrected
- Lack of Events Code Corrected

6.1 Lack of Router Sanity Check

Design Low Version 1 Code Corrected

CS-MGVSTRATS-001

While the `MangroveOffer` library does not enforce a router to be set, more derived contracts such as `Forwarder` strictly require a router. However, `setRouter()` is not adapted to enforce such a check.

Code corrected:

A check has been added to `Forwarder` and to `AaveKandel` which are the strategies that require a router to be set.

6.2 Specification Mismatch

Correctness Low Version 1 Specification Changed

CS-MGVSTRATS-002

The specification of `RESERVE_ID` states that

```
///@dev RESERVE_ID==address(0) will pass address(this) to the router for the id field.
```

However, `RESERVE_ID` will never be `address(0)` as it is set to `address(this)` if the corresponding constructor argument is `address(0)`.

Additionally, the documentation of the geometric price of `baseQuoteTickIndex0` describes the following

```
///@param baseQuoteTickIndex0 the tick of base per quote for the price point at index 0 [...]
```

However, that may be misinterpreted. Namely, considering the two markets for asks and bids, the ask's price will be denominated in quote per base (ratio of inbound and outbound) while the bids' prices will be denominated in base per quote. The documentation, thus, suggests that the tick is defined for the market of the bids. However, that contrasts the implementation where the tick is used for the ask markets.

Specification changed:

The NatSpec has been adjusted.

6.3 Gas Optimisations

Informational **Version 1** **Code Corrected**

CS-MGVSTRATS-005

Below is an incomplete list of potential gas optimisations:

1. In `Forwarder.sol`, `deriveGasprice()` evaluates `1e6 * (offerGasbase + gasreq)` to compute a value that is already in `num`.
2. In `CoreKandel.sol`, `setParams` guards against truncation when casting `newParams.pricePoints` to `uint32`. This is unnecessary since `newParams.pricePoints` is already an `uint32`.
3. `setExpiry()` and `retractOffer()` in `MangroveOrder` have the `mgvOrOwner` modifier. However, the core will never call these functions.

Code corrected:

The code has been adjusted to include the above gas optimisations.

6.4 Lack of Events

Informational **Version 1** **Code Corrected**

CS-MGVSTRATS-006

While the codebase emits events on many occasions, a `SetAdmin` is missing in the constructor of the `AccessControlled` contract.

Code corrected:

The event is now emitted.

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**

CS-MGVSTRATS-007

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 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 Hypothetical Bad Setups for Kandels

Note Version 1

While we exclude "weird" setups for the Kandel from scope, we would like to give an incomplete list of examples of such breaking setups:

1. `populate()` does not enforce the geometric price progression over indices. The distribution could be arbitrary.
2. `populate()` does not enforce that for every index a bid and ask exist. Hence, one could create setups where the updating of dual offers would always fail due to a lack of offer ID.
3. The geometric population functions have assumptions such as the tick spacing of the market being respected by the arguments.

8.2 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, which 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`.

8.3 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.

8.4 Updating Approvals on Order Update

Note Version 1

A user can update their orders by using `Forwarder.updateOffer`. Users need to remember that, in case the `makerExecute` hook to their order fails, they will have to reimburse the taker. A reason for an order to fail is that there is not enough allowance given to the router to transfer funds from the maker's reserve to the `MangroveOrder` contract. This is highly likely to happen after a user updates their offer by having it give more funds to the taker.

8.5 `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.