Security Review Report
NM-0162 Mangrove

NETHERMIND
SECURITY

(Feb 14, 2024)

Contents

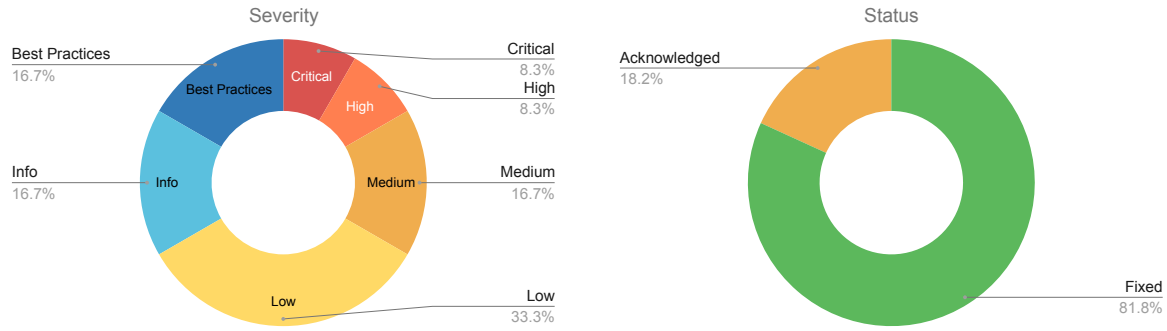
1 Executive Summary

This document outlines the security review conducted by Nethermind Security for part of the strategy-lib developed by Mangrove. Mangrove is an on-chain order book DEX that allows liquidity providers to post arbitrary smart contracts as offers. To make the execution of common strategies on the platform simpler, the Mangrove team has created a set of smart contracts that can be directly used by users or can be taken as building blocks for more complex use cases. The engagement covered two strategies: **MangroveAmplifier** and **MangroveOrder**. The former implements "liquidity amplification", which allows the offer owner to post offers in multiple markets using the same collateral. The latter will allow users to execute *"Good Til Canceled"* orders, *"Good Til Canceled Enforced"* orders, *"Post Only" orders*, *"Immediate Or Cancel"* orders, and *"Fill Or Kill"* orders.

The audited code comprises 1175 lines of code. The Mangrove team has provided comprehensive documentation explaining the behavior of the core protocol and the reviewed strategies. Aside from the provided documentation, the Mangrove and Nethermind Security teams have actively communicated to clarify any remaining questions about the expected behavior of the protocol.

The audit was performed using: (a) manual analysis of the codebase, (b) automated analysis tools, and (c) simulation of the smart contracts. **Along this document, we report** twelve points of attention, where one is classified as Critical, one is classified as High, two are classified as Medium, four are classified as Low, and four are classified as Informational or Best Practice. The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 presents the files in the scope of this audit. Section 3 summarizes the issues. Section 4 presents the system overview. Section 5 analyses some of the risks associated with the use of the amplified liquidity strategy. Section 6 discusses the risk rating methodology adopted for this audit. Section 7 details the issues. Section 8 discusses the documentation provided by the client for this audit. Section 9 presents the compilation, tests, and automated tests. Section 10 concludes the document.



(a) distribution of issues according to the severity

(b) distribution of issues according to the status

Fig 1: (a) Distribution of issues: Critical (1), **High** (1), **Medium** (2), **Low** (4), **Undetermined** (0), **Informational** (2), **Best Practices** (2). **(b) Distribution of status: Fixed** (9), **Acknowledged** (3), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Jan 26, 2024
Final Report	Feb 14, 2024
Methods	Manual Review, Automated Analysis
Repository	mangrove-strats
Commit Hash	5462064d6df604aaae297a34e44599405de13fa7
Final Commit Hash	784758afa4761a9a4e6e2323d56be56b67ce54f3
Documentation	Developers documentation
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/strategies/MangroveOffer.sol	123	105	85.4%	24	252
2	src/strategies/MangroveAmplifier.sol	217	117	53.9%	32	366
3	src/strategies/MangroveOrder.sol	157	135	86.0%	22	314
4	src/strategies/TakerOrderLib.sol	29	38	131.0%	9	76
5	src/strategies/routers/RouterProxyFactory.sol	37	31	83.8%	6	74
6	src/strategies/routers/SmartRouter.sol	70	27	38.6%	10	107
7	src/strategies/routers/SimpleRouter.sol	26	9	34.6%	4	39
8	src/strategies/routers/RouterProxy.sol	24	22	91.7%	8	54
9	src/strategies/routers/SmartRouterStorage.sol	24	12	50.0%	5	41
10	src/strategies/routers/abstract/RoutingOrderLib.sol	26	15	57.7%	7	48
11	src/strategies/routers/abstract/AbstractRouter.sol	60	48	80.0%	17	125
12	src/strategies/utils/AccessControlled.sol	31	40	129.0%	9	80
13	src/strategies/offer_forwarder/RenegingForwarder.sol	91	60	65.9%	14	165
14	src/strategies/offer_forwarder/abstract/Forwarder.sol	182	129	70.9%	33	344
15	src/strategies/interfaces/IForwarder.sol	6	13	216.7%	3	22
16	src/strategies/interfaces/IOfferLogic.sol	23	42	182.6%	10	75
17	src/strategies/interfaces/IOrderLogic.sol	40	35	87.5%	6	81
18	src/strategies/interfaces/ILiquidityProvider.sol	9	21	233.3%	5	35
	Total	**1175**	**899**	**76.5%**	**224**	**2298**

3 Summary of Issues

	Finding	Severity	Update
1	Unauthorized user can update and retract offers within a bundle	Critical	Fixed
2	Wrong computation of residual values	High	Fixed
3	Total withdrawn provision incorrectly computed in `_retractBundle(...)`	Medium	Fixed
4	`_retractOffer(...)` returns an unwithdrawn `freeWei` amount, resulting in the depletion of the contract funds	Medium	Fixed
5	Strategy admin can seize users unlocked wei balances	Low	Acknowledged
6	The `__lastLook__(...)` function might revert because of the strict inequality check	Low	Fixed
7	Unnecessary call to `_retractBundle(...)` when the maker fails to get the outbound tokens	Low	Fixed
8	Updated offers after retraction can use an outdated volume condition	Low	Fixed
9	Offers from the same `offerList` in a bundle are not updated when `_updateBundle(...)` is called.	Info	Acknowledged
10	`outboundVolume` of bundles can be unintentionally modified	Info	Acknowledged
11	Incorrect NatSpec comments	Best Practices	Fixed
12	Unused imports	Best Practices	Fixed

4 System Overview

`Mangrove` offers unparalleled flexibility for its users through the use of strategies. Users can incorporate defensive code, post unprovisioned offers, and redisplay liquidity after their offers are taken. The protocol provides multiple pre-developed strategies that users can use for common use cases without needing to write their own logic. Moreover, these strategies serve as building blocks for crafting more complex processes. This section provides an overall description of two strategies: **MangroveAmplifier** and **MangroveOrder**. Additionally, it delves into other concepts relevant to the development of strategies. The next figure presents an architectural overview of these two strategies. For each contract, its relationships with other contracts and its most relevant functions are presented.

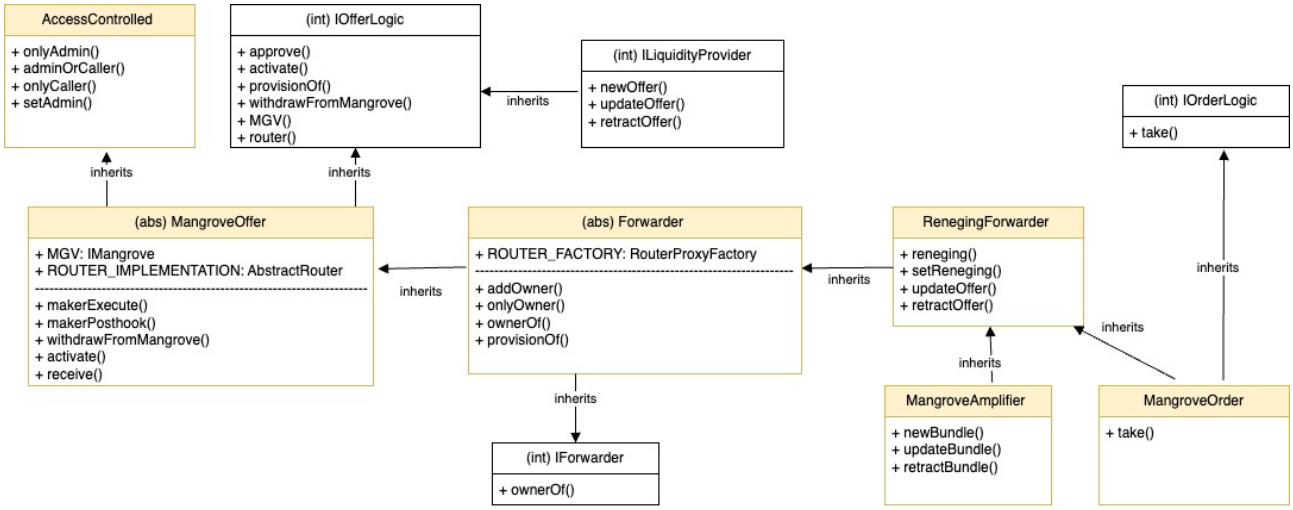


Fig. 2 : Mangrove system scheme.

4.1 MangroveAmplifier

The **MangroveAmplifier** is a custom maker contract that implements liquidity amplification. It allows the offer owner to post offers on multiple markets with the same collateral. These offers are coupled together under a single abstraction called a bundle. Users will specify the maximum amount of outbound tokens that they are willing to offer and the bundle cannot give more than this amount even if it is approved to spend more or has access to more funds.

The **MangroveAmplifier** contract contains three endpoints for the users to interact with:

- `newBundle(...)`: To post a bundle of offers on Mangrove, amplifying a certain volume of outbound tokens.
- `updateBundle(...)`: To update common parameters of a bundle of offers (i.e outbound volume and expiry date).
- `retractBundle(...)`: To retract a bundle of offers.

When a bundle is created, the offers are posted to Mangrove on behalf of the user. If one of the offers is taken, the remaining offers in the bundle are retroactively updated to either use the residual amount or retracted. Users can update their bundle if they want to change the outbound volume and/or expiry date, or retract their bundle if they no longer want to offer the outbound tokens. Additionally, they can update specific offers in the bundle to set more specific expiry times and/or volume or retract them.

4.2 MangroveOrder

The **MangroveOrder** is a custom maker contract that implements the orders used in traditional finance. It enables a taker to simultaneously place an order on Mangrove and submit an offer with the remaining order values. The following types of orders are supported by the contract:

- **Good Til Canceled Orders:** These are limit orders that remain active until it is filled or canceled by the trader. If the GTC order is partially filled, a resting order for the remaining portion is posted. The transaction will not revert if the resting order fails to be posted.
- **Good Til Canceled Enforced Orders:** These are limit orders that remain active until it is filled or canceled by the trader. If the GTC order is partially filled, a resting order for the remaining portion is posted. The transaction will revert if the resting order fails to be posted.
- **Post Only Orders:** These are limit orders that are posted to the market but not executed immediately.
- **Immediate Or Cancel Orders:** These are limit orders that must be filled immediately at the limit price or better. If the order is partially filled, the remaining portion is canceled.

- **Fill Or Kill Orders:** These are limit orders that are either completely filled or canceled. There is no partial fill, and no resting order is posted.

The primary entry point for the contract is the `take(...)` function, where takers utilize it to post a market order on Mangrove by providing the necessary order details as input.

- The created order is executed on Mangrove through the `executeMarketOrder(...)` function. In this step, inbound tokens are extracted from the user's reserves and directed to the `MangroveOffer` contract using the specified routing logic. If the market order succeeds, the acquired outbound tokens are transferred back to the taker's reserves using the appropriate router.
- In cases where the original order fails to be filled or is only partially filled, a resting order is placed using the `postRestingOrder(...)` function. This involves posting an offer on the opposite market, achieved by flipping the outbound and inbound tokens and including the residual amounts from the initial order. Any bounty accrued during order execution, along with the `wei` sent by the caller to the function, is utilized for provisioning the new offer. The remaining funds are promptly returned to the caller.

The function manages the execution flow based on the Order type, reverting in scenarios where the order type restricts a particular situation. For instance, if a Fill Or Kill order is partially filled, the function reverts before posting the residual offer.

4.3 Routers

Routers play an important role within the mangrove ecosystem as they are responsible for moving liquidity to and from user reserves. The router schema is shown in Fig. 3.

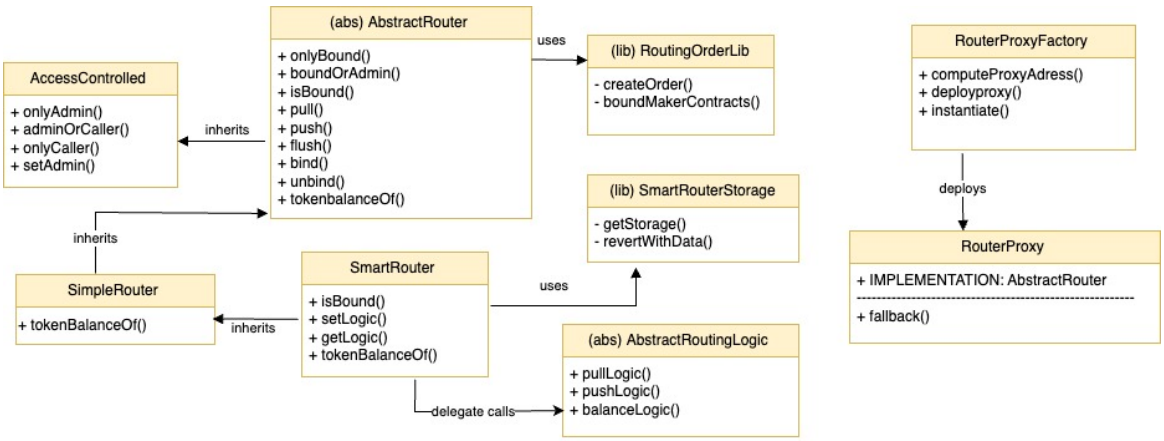


Fig. 3 : Mangrove routers scheme.

4.3.1 `RouterProxyFactory` and `RouterProxy`

The `RouterProxyFactory` and `RouterProxy` contracts are used to manage the deployment of different routers within the strategies.

The `RouterProxyFactory` is a factory contract that allows the creation of `RouterProxy` instances for different users using deterministic addresses. The deployed router address is highly dependent on its default owner and the implementation address. This contract has the following main functions:

- `deployProxy()`: serves as the main entry point to deploy a proxy, it takes the owner address and the implementation addresses as an input.
- `instantiate()`: this function encapsulates a call to the `deployProxy()` by first checking if the proxy was not already deployed; if it is the case, its address is returned.
- `computeProxyAddress()`: a utility function to precompute the deployed address, it is used within the `instantiate()` function to determine if the proxy was previously deployed.

The `RouterProxy` contract is a simple implementation of a proxy, pointing to an immutable router implementation. Similar to any proxy contract, calls to this contract are delegated to its implementation.

4.3.2 AbstractRouter

An abstract contract that represents the base layer for implementing routing mechanisms such as in `SimpleRouter` and `SmartRouter`. The `AbstractRouter` contract inherits from `AccessControlled` and utilizes the `RoutingOrderLib` library, which contains functions for managing access controls and creating routing order structures respectively. The router can be bound to one or more maker addresses, allowing those maker addresses to pull or push the liquidity from it.

The main functions in `AbstractRouter` includes:

- `pull()`: An external function callable only by `bounded` addresses, it pulls liquidity from the user reserves and sends it to the caller using a custom logic defined in child contracts within the internal `__pull__()` function.
- `push()`: Similar to `pull()`, it's an external function used to push liquidity from the caller to the reserve using a custom logic defined in child contracts within the internal `__push__()` function.
- `flush()`: Allows pushing balances of multiple tokens to the reserve. It operates by iterating through the routing orders and calling the `__push__()` internal function on them.
- `bind()`: Function used to bind a new maker contract to the router.
- `unbind()`: Function used to remove addresses from the mapping of bounded maker contracts.

4.3.3 SimpleRouter

The `SimpleRouter` contract inherits from `AbstractRouter` and implements the `__pull__()`, `__push__()` and `tokenBalanceOf()` functions by overriding them. This contract implements a simple routing logic where the funds are pulled and pushed from the owner's account using a direct ERC20 transfer.

4.3.4 SmartRouter

The `SmartRouter` inherits the functionalities of a `SimpleRouter` and implements a more complex routing logic. It delegates routing requests to arbitrary contracts that implement the `AbstractRoutingLogic` interface, referred to as routing logic. It introduces new functions to manage the routing logic associated with each offer and token:

- `setLogic()`: sets the logic contract that will be used for routing requests relative to a specific token, offer list, and offerId.
- `getLogic()`: get the address of the associated logic for the provided inputs.
- `__push__()` and `__pull__()`: These internal functions are redefined to reflect the `SmartRouter` behavior. If a routing logic is set for the specified inputs, the request is forwarded to that logic through a delegatecall. Otherwise, the default implementation defined within the `SimpleRouter` is used.

Additionally, the `SmartRouter` contract has an immutable state variable `forcedBinding`, used to bind a default address to a router. This is used to allow users to post an order and bind the maker contract is the same transaction.

4.3.5 Routers usage within the Amplifier strategy

The amplifier strategy uses a single `RouterProxyFactory` contract in addition to an immutable router implementation shared across all deployed proxy routers, directing them to a `SmartRouter` contract.

When a user initiates the creation of a new bundle, the contract interacts with the `RouterProxyFactory` to deploy a `RouterProxy` contract for the user. The deployment is done through a call to the `instantiate()` function and uses the strategy's default implementation. At this step, users have the flexibility to specify the routing logic address for each offer in the bundle, which will be specific to the inbound tokens. Additionally, a single routing logic can be designated for the shared outbound token of the bundle.

When liquidity needs to be routed to or from the reserve, such as during the offer execution, the strategy initiates a call to the `push()` or `pull()` functions on the `RouterProxy` contract. This process involves fetching the associated routing logic based on the user's request and subsequently delegating the push or pull call to it. In cases where no specific logic is provided, the default implementation is used, ensuring funds are transferred to/from the owner's wallet.

5 Analysis of amplified offers implementation

The amplifier strategy introduces the concept of Bundles, aiming to enable liquidity amplification by allowing offer owners to post offers on multiple markets with the same collateral. The bundle structure consists of a list of offers that share common parameters, including the promised asset `outbound_tkn`, the promised volume per offer `outVolume`, the outbound token's routing logic `outboundLogic`, and the expiration date of each offer in the bundle `expiryDate`.

```
struct FixedBundleParams {
    IERC20 outbound_tkn;
    uint outVolume;
    AbstractRoutingLogic outboundLogic;
    uint expiryDate;
}
```

Bundles are designed with the intention to prevent users from overspending. The bundle owner can spend up to the promised volume of the bundle, even if he approves the strategy to spend more. However, there are inherent risks associated with this concept that need careful consideration:

5.1 Offer failure within a Bundle

In scenarios where an offer within a bundle fails to deliver the promised tokens, the remaining offers in the same bundle are also at risk of reverting. This is because of the shared routing logic for the outbound token among offers within the same bundle.

One of the current system limitations is the lack of the immediate ability to retract remaining offers in the bundle during execution. This action is necessary to prevent multiple penalties for the maker. When the maker fails to deliver the promised tokens to Mangrove, the transaction will revert, making it impossible to execute any logic at that time.

The existing solution utilizes the `__posthookFallback__` function to retract the bundle, executed after the trade, which mitigates the risk but does not completely eliminate it. Malicious actors could exploit this situation within the same transaction of the offer execution by targeting remaining offers for cleaning and collecting the bounty for the entire bundle.

5.2 Discrepancy between the traded volume and `offer.gives()`

The protocol utilizes a volume condition `cond.volume` to update the traded volume of offers within a bundle that failed to be updated directly on Mangrove. A potential issue arises from the discrepancy between the offer's `offer.gives()` amount and the actual traded volume stored in `cond.volume`. Malicious actors can exploit this by targeting these bundles, causing offers to fail and bundles to be retracted.

For instance, if the bundle is updated with a smaller outbound volume, and an offer (e.g., Offer A) fails to be immediately updated on Mangrove, that offer will have `cond.volume < offer.gives()`. When a market order is matched with offer A, Mangrove uses the stored `offer.gives()` value as the trade volume. However, since the actual volume is lower, the offer execution can potentially fail, resulting in penalties imposed on the maker. Malicious actors can target offers in a similar situation, leading to unfair bundle retractions and penalties for the maker, who initially provided the promised amount reflected in `cond.volume`.

Conclusion: impact and considerations

In conclusion, the identified issues have dual impacts on the protocol. Firstly, they can disrupt the smooth functioning by causing offers to fail and triggering the retraction of entire bundles. Secondly, the financial implication is mostly constrained by the provisions paid by the maker, with the total paid amount capped at the provided provision during bundle creation.

Different factors are to be taken into consideration when evaluating the likelihood of such attacks and the motivation for bad actors. For instance, the chain where the protocol is deployed plays a significant role. When the gas price is low, attackers may be less motivated to trigger bundle retractions due to the potential higher attack cost than gains. Furthermore, the size of the bundle impacts the potential gains, with smaller bundles resulting in smaller bounties for each targeted bundle.

Understanding the impact and motivation factors for these attacks is important to design preventive measures and enhance the overall resilience of the protocol against malicious activities.

6 Risk Rating Methodology

The risk rating methodology used by Nethermind follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely an attacker will uncover and exploit the finding. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to Motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding, other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to formally pass to the client;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

7 Issues

7.1 [Critical] Unauthorized user can update and retract offers within a bundle

File(s): MangroveAmplifier.sol

Description: In the `updateBundle(...)` and `retractBundle(...)` functions, the bundle to be updated or deleted is identified using a `bundleId` and the associated outbound token `outbound_tkn`. However, the value of the outbound token is set by the caller and can be manipulated. It is used within the `_extractOwnerOf(...)` function to authorize the user to proceed with the bundle changes if he is the bundle owner.

```
function updateBundle(
// ...
bool updateExpiry,
// ...
) external {
BundledOffer[] memory bundle = __bundles[bundleId];
// @audit `outbound_tkn` can be different from the actual outbound token of the bundle.
require(_extractOwnerOf(bundle, outbound_tkn) == msg.sender, "MgvAmplifier/unauthorized");
// ...
}
```

Passing a manipulated outbound token address to the `_extractOwnerOf(...)` function results in the computation of an `olKey` pointing to a different offer list. This can lead to unauthorized access, enabling the caller to alter a bundle they do not own.

```
function _extractOwnerOf(BundledOffer[] memory bundle, IERC20 outbound_tkn) internal view returns (address) {
    // @audit  a manipulated `inbound_tkn` results in verifying the offerID in a different offer list.
    OLKey memory olKey = OLKey({
      outbound_tkn: address(outbound_tkn),
      inbound_tkn: address(bundle[0].inbound_tkn),
      tickSpacing: bundle[0].tickSpacing
    });
    return (ownerOf(olKey.hash(), bundle[0].offerId));
  }
```

A malicious user can pass an outbound token address so that the computed `olKey` inside this function has an offer with the same `bundle[0].offerId`, and he owns that offer. That way, the caller bypasses the authentication check and can alter or retract offers he doesn't own.

Recommendation(s): Consider validating the `outbound_token` input within the `updateBundle(...)` and `retractBundle(...)` functions to ensure it accurately corresponds to the specified bundle's outbound token.

Status: Fixed.

Update from the client: Offer bundles are now mapped uniquely to their *bundleId* but requires the outbound token as well.

```
///@notice maps a bundle Id and its outbound token to a set of bundled offers
mapping(uint bundleId => mapping(IERC20 outbound_tkn => BundledOffer[])) private __bundles;
```

Fixed in commit ea1b1f7ca8ef52f3cee9bf14de0e54afa56b5008.

7.2 [High] Wrong computation of residual values

File(s): MangroveAmplifier.sol

Description: The execution of `makerExecute(...)` in the `MangroveAplifier` is split into three different hooks:

- `__lastLook__(...)`: This hook checks if the offer can be executed in the current context. For an offer to be valid in the current context, it must not have expired, and the `order.takerWants` must be lower or equal to the maximum volume the offer accepts. The maximum volume that the offer accepts is `offer.gives()` or `cond.volume` when `cond.volume` is different from zero. The `cond.volume` value limits the number of outbound tokens the offer can give when the offer cannot be updated during the execution of a different offer from the same bundle;
- `__put__(...)`: This hook contains the logic for storing the received tokens;
- `__get__(...)`: This hook contains the logic for pulling the tokens that will be given from the reserve. In this strategy, this hook also updates the rest of the offers in the bundle. The rest of the bundle should be retracted when the offer was completely consumed or updated with the correct residual amount in case it was partially consumed;

The current implementation of the `__get__(...)` hook in the `MangroveAmplifier.sol` contract incorrectly updates the rest of the bundle when the offer is partially filled under certain circumstances.

```
function __get__(uint amount, MgvLib.SingleOrder calldata order) internal override returns (uint) {
  ...
  // @audit - This value is incorrect when `order.offer.gives()` is not up to date.
  uint newOutVolume = order.offer.gives() - order.takerWants;
  if (missing == 0 && newOutVolume > 0) {
    _updateBundle(bundle, IERC20(order.olKey.outbound_tkn), olKeyHash, newOutVolume);
  } else {
    ...
  }
  return missing;
}
```

When the `order.offer.gives()` value is not up to date, the `newOutVolume` value, which represents the residual tokens, is incorrect. This value may not be updated if an offer of the bundle was executed before and failed while trying to update this offer; in this case, the `code.volume` is used to set the maximum volume of the order. The `newOutVolume` value will be greater than what it should be. This will cause all the updated offers to overpromise on the number of outbound tokens they offer. These overpromises may lead to the creator of the bundle giving more outbound tokens than desired.

Similarly, the residual tokens may also be incorrect during the reposting of an offer in the `__posthookSuccess__(...)` hook execution. The `__residualValues__(...)` function implemented in the `RenegingForwarder.sol` contract accounts for `cond.volume` values when they are different from zero.

```
function __residualValues__(MgvLib.SingleOrder calldata order)
    internal
    virtual
    override
    returns (uint newGives, Tick newTick)
  {
    Condition memory cond = __renegeMap[order.olKey.hash()][order.offerId];
    if (cond.volume == 0) {
      return super.__residualValues__(order);
    } else {
      // new volume to be offered
      // if __residualValues__ is called in __posthookFallback__ the offer reneged and cond.volume could be higher than
      ↪  takerWants
      newGives = order.takerWants < cond.volume ? cond.volume - order.takerWants : 0;
      // same price
      newTick = order.offer.tick();
    }
  }
```

However, during the execution of `__residualValues__(...)`, the `cond.volume` will always be equal to zero because it will be set to zero in the `__lastLook__(...)` hook.

```
function __lastLook__(MgvLib.SingleOrder calldata order) internal virtual override returns (bytes32) {
  ...
  if (cond.volume != 0) {
    // if a max volume is set and we are not reneging, we set it back to 0 so that we don't check again and save some gas.
    // note this is OK since reposting the offer in MangroveOffer.__posthookSuccess__ will update the volume according to `__residualValues__`
    _setReneging(olKeyHash, order.offerId, cond.date, 0);
  }
  return super.__lastLook__(order);
}
```

Recommendation(s): Consider resetting the volume condition to zero within the `_updateOffer(...)` function by overriding it in the `RenegingForwarder` contract. Additionally, modify the computation of the `newOutVolume` to use the `__residualValues__(...)` function so it accounts for `cond.volume` values.

Status: Fixed.

Update from the client: We addressed this vulnerability in two steps. First we make sure that the computation of the new outbound volume always uses `__residualValues__` function which takes `cond.volume` into consideration.

```
(uint newOutVolume,) = __residualValues__(order);
```

Second, we make sure that `cond.volume` is reset to 0 in the override of `_updateOffer` in `RenegingForwarder`.

```
function _updateOffer(OfferArgs memory args, uint offerId) internal override returns (bytes32 reason) {
  Condition memory cond = reneging(args.olKey.hash(), offerId);
  if (cond.volume > 0) {
    // resetting reneging volume to 0 when updating offer
    // When reposting on partial fill, residual values are computed based on the reneging volume.
    // So new we can safely set back the max volume to 0.
    _setReneging(args.olKey.hash(), offerId, cond.date, 0);
  }
  return super._updateOffer(args, offerId);
}
```

During *makerExecute* we have the following flow:

1. In *__lastLook__* *order.wants* is compared to *cond.volume* if any. Offer reverts if the taker wants more than the maximum volume.
2. If the offer proceeds, and is partially filled, *__residualValues__* will take *cond.volume* as virtual offered volume (and not *offer.gives*) in order to compute residual gives for reposting.
3. In `__posthookSuccess__` offer is reposted via the *_updateOffer* function, which then sets *cond.volume* back to 0.

Fixed in commit 33877b6cdadac156efc9b885e9fa572a70a8b190.

7.3 [Medium] Total withdrawn provision incorrectly computed in `_retractBundle(...)`

File(s): MangroveAmplifier.sol

Description: The `_retractBundle(...)` function returns the total freed wei when retracting the offers inside the specified bundle. However, this value is reset inside the loop within every iteration instead of being accumulated.

```
function _retractBundle(BundledOffer[] memory bundle, IERC20 outbound_tkn, bytes32 skipOlKeyHash, bool deprovision)
    internal
    returns (uint freeWei)
  {
    for (uint i; i < bundle.length; i++) {
      // ...
      // @audit `freeWei` is reset within every iteration
      (freeWei, status) = _retractOffer(olKey_i, bundle[i].offerId, true, deprovision);
      // ...
    }
  }
```

The function will only return the withdrawn wei of the last removed offer. The remaining provisions will stay in the contract balance, and the bundle owner will lose access to them.

Recommendation(s): Adjust the logic to accumulate the `freeWei` value across the different iterations instead of resetting it within each iteration.

Status: Fixed.

Update from the client: To address this issue, we added a temp `_freeWei` variable as recommended.

```
function _retractBundle(BundledOffer[] memory bundle, IERC20 outbound_tkn, bytes32 skipOlKeyHash, bool deprovision)
  internal
  returns (uint freeWei)
{
  bytes32 status;
  uint _freeWei;
  bytes32 olKeyHash_i;
  for (uint i; i < bundle.length; i++) {
    OLKey memory olKey_i = OLKey({
      outbound_tkn: address(outbound_tkn),
      inbound_tkn: address(bundle[i].inbound_tkn),
      tickSpacing: bundle[i].tickSpacing
    });
    olKeyHash_i = olKey_i.hash();
    if (skipOlKeyHash != olKeyHash_i) {
      (_freeWei, status) = _retractOffer(olKey_i, bundle[i].offerId, true, deprovision);
      freeWei += _freeWei;
      if (status != bytes32(0)) {
        // this only happens if offer `i` of the bundle is in a locked offer list --see `_updateBundle`
        _setReneging(olKeyHash_i, bundle[i].offerId, block.timestamp, 0);
      }
    }
  }
}
```

Fixed in commit 16f67e03ce0b5d6279da11bad7659d593cb0ff98.

7.4 [Medium] `_retractOffer(...)` returns an unwithdrawn `freeWei` amount, resulting in the depletion of the contract funds

File(s): `Forwarder.sol`

Description: The `_retractOffer(...)` function removes an offer from the offer list and returns a `freeWei` amount, indicating the ETH withdrawn from Mangrove. This value is then used to transfer funds to the respective owner. However, an issue arises when the call to Mangrove's `retractOffer(...)` function fails, and `noRevert` is set to `true`. In such cases, no funds are withdrawn from Mangrove, but the `freeWei` can still hold a non-zero value.

```
function _retractOffer(...) internal returns (uint freeWei, bytes32 status) {
  OwnerData storage od = ownerData[olKey.hash()][offerId];
  // @audit If the offer deletion fails, `freeWei` can hold the amount of `od.weiBalance`
  freeWei = deprovision ? od.weiBalance : 0;

  try MGV.retractOffer(olKey, offerId, deprovision) returns (uint weis) {
    // ...
  } catch Error(string memory reason) {
    // @audit on failing calls, no wei is actually withdrawn from Mangrove
    require(noRevert, reason);
    status = bytes32(bytes(reason));
  }
}
```

This issue is possible within the `_retractBundle(...)` function of the `MangroveAmplifier` contract as it does not consider the returned `status` of `_retractOffer(...)` while using the `freeWei` amount. Consequently, in case of a failure, the `freeWei` amount is not withdrawn from Mangrove, yet it remains accounted for in the owner's balance `od.weiBalance`. Subsequently, the `retractBundle(...)` function attempts to transfer this unwithdrawn amount to the owner from its contract balance.

```
function _retractBundle(...) internal returns (uint freeWei) {
  for (uint i; i < bundle.length; i++) {
    // ...
    if (skipOlKeyHash != olKeyHash_i) {
      bytes32 status;
      // @audit  when status indicates a failure, the amount in `freeWei` is not withdrawn from Mangrove.
      (freeWei, status) = _retractOffer(olKey_i, bundle[i].offerId, true, deprovision);
      if (status != bytes32(0)) {
        _setReneging(olKeyHash_i, bundle[i].offerId, block.timestamp, 0);
      }
    }
  }
}
```

A malicious user can take advantage of this vulnerability by repeatedly invoking the `retractBundle(...)` function on a bundle where offer deletion consistently fails. This could lead to the depletion of contract funds as the caller accumulates `od.weiBalance` with each call.

Recommendation(s): Consider modifying the `_retractOffer(...)` or the `_retractBundle(...)` functions to either return 0 when the status indicates a failure or ignore the returned `freeWei` amount in case of failure, preventing the transfer of unwithdrawn funds to the owner.

Status: Fixed.

Update from the client: To solve this issue, we decoupled currently available weis assigned to `owner` (in `ownerData.weiBalance`) from the additional wei freed by a successful call to `retractOffer`. If `retractOffer` fails while the `noRevert` flag is positioned we can still return the weiBalance of the owner. This balance is augmented by freed weis if the call to `retractOffer` did not fail.

```
OwnerData storage od = ownerData[olKey.hash()][offerId];
freeWei = deprovision ? od.weiBalance : 0;

try MGV.retractOffer(olKey, offerId, deprovision) returns (uint weis) {
  // We add `weis` credited to the contract to `freeWei` amount
  // They were the locked native token provision of the offer that has been retracted
  freeWei += weis;
} catch Error(string memory reason) {
  require(noRevert, reason);
  status = bytes32(bytes(reason));
}
// if `deprovision` is set to false, `freeWei` is 0 because `retractOffer` will return 0
if (freeWei > 0) {
  // pulling free wei from Mangrove to `this`
  require(MGV.withdraw(freeWei), "Forwarder/withdrawFail");
  // resetting pending returned provision
  // calling code must now assign freeWei to owner
  od.weiBalance = 0;
}
```

Fixed in commit 5c0fde0716363ca21187340a1fef0ce0e2ad9855.

7.5 [Low] Strategy admin can seize users unlocked wei balances

File(s): MangroveOffer.sol

Description: The MangroveOffer contract provides the withdrawFromMangrove(...) function that allows the contract admin to retrieve the unlocked funds from Mangrove. The objective behind this function is to prevent any residual wei, mainly arising from approximations done within the __handleResidualProvision__ function, from being locked within Mangrove.

However, this function enables the contract admin to either maliciously or mistakenly withdraw unlocked funds allocated to a specific user in the Forwarder strategy. The admin can then seize a user's ownerData.weiBalance. Essentially, any amount reflected in the user's ownerData.weiBalance as unlocked in Mangrove becomes susceptible to withdrawal by the strategy admin.

Furthermore, the withdrawn funds are still accounted for in the ownerData.weiBalance users struct, resulting in an inconsistency between the user's balances in the strategy accounting and the actual funds in Mangrove. This might cause some transactions to revert, particularly when attempting operations such as retracting an offer or bundle, as they may try to withdraw an amount exceeding the available wei in Mangrove.

Recommendation(s): Consider updating the function to consider the user's balances by retracting them from the total amount to be withdrawn from Mangrove. As a different option, consider giving access to this functionality only to multisig contracts and carefully reviewing every transaction done from it to reduce the possibility of errors.

Status: Acknowledged.

Update from the client: This issue will stay unresolved. Admin cannot seize inbound or outbound token which are not assigned to the contract balance between transactions. The only thing admin could seize is free weis from Mangrove unlocked wei balance assigned to the given forwarder, but the reputational value of such operations is not worth the bounty. Moreover, admin control of the balance is a way to recover unreachable funds that would have been transferred by mistake.

7.6 [Low] The __lastLook__(...) function might revert because of the strict inequality check

File(s): RenegingForwarder.sol

Description: The __lastLook__(...) function checks if the order is expired or oversized before executing the trade. These cases are called reneging conditions. In a scenario where the offer was locked on Mangrove, but it is still necessary to update how much outbound volume it should offer, its cond.volume is set to avoid over-promising. The __lastLook(...)__ function compares the token amount that the taker wants with the cond.volume - the number of tokens the offer can sell. The problem present in this function is that it uses a strict inequality sign to determine whether the transaction should revert.

```
require(
  cond.volume == 0 || order.takerWants < uint(cond.volume),
  "RenegingForwarder/overSized"
);
```

For example, if cond.volume was set to 500 tokens, any taker should still be able to take the remaining 500 tokens promised by the offer, but this is not the case. Because of the strict inequality sign, it is only possible to execute this offer by requesting 499 tokens.

Recommendation(s): Consider using a weak inequality sign.

Status: Fixed.

Update from the client: This issue is resolved by changing the strict inequality check to a non-strict one.

```
require(cond.volume == 0 || order.takerWants <= uint(cond.volume), "RenegingForwarder/overSized");
```

Fixed in commit 4f5ce2e22184b08d0d3b0cc59a76f9acbc24ac76.

7.7 [Low] Unnecessary call to `_retractBundle(...)` when the maker fails to get the outbound tokens

File(s): MangroveAmplifier.sol

Description: The `__get__(...)` function redefined in `MangroveAmplifier` is called within the `makerExecute(...)` function, which will always revert when the returned value `missing` is different than 0:

```
require(__get__(order.takerWants, order) == 0, "mgvOffer/abort/getFailed");
```

However, the function's logic attempts to retract the bundle when `missing` value is different than 0, as the code shows below:

```
function __get__(uint amount, MgvLib.SingleOrder calldata order) internal override returns (uint) {
    uint missing = super.__get__(amount, order);

    if (missing == 0 && newOutVolume > 0) {
      _updateBundle(bundle, IERC20(order.olKey.outbound_tkn), olKeyHash, newOutVolume);
    } else {
      // not deprovisionning to save execution gas
      _retractBundle(bundle, IERC20(order.olKey.outbound_tkn), olKeyHash, false);
    }
    return missing;
  }
```

The issue lies in the fact that, due to the revert condition in `makerExecute(...)`, the actions within the else block of the `__get__(...)` function. Specifically, attempting to retract the bundle will never be executed when `missing != 0`. Consequently, a failed offer within the bundle will not trigger the expected bundle retraction. Additionally, this results in unnecessary gas consumption to retract the bundle, which will increase the bounty amount paid by the maker.

Recommendation(s): Consider moving the `_retractBundle(...)` call to the `__posthookFallback__(...)` function for the case where `missing != 0`.

Status: Fixed.

Update from the client: `_retractOffer` is now only called if missing is 0 and residual volume is 0.

```
(uint newOutVolume,) = __residualValues__(order);
if (missing == 0) {
  if (newOutVolume > 0) {
    _updateBundle(bundle, IERC20(order.olKey.outbound_tkn), olKeyHash, newOutVolume);
  } else {
    // not deprovisionning to save gas
    _retractBundle(bundle, IERC20(order.olKey.outbound_tkn), olKeyHash, false);
  }
}
return missing;
  }
```

If missing was non zero, we now retract bundle in posthookFallback

```
/// @inheritdoc MangroveOffer
/// @dev Because the offer execution failed, we will retract the rest of the bundle.
function __posthookFallback__(MgvLib.SingleOrder calldata order, MgvLib.OrderResult calldata result)
  internal
  virtual
  override
  returns (bytes32 data)
{
  bytes32 olKeyHash = order.olKey.hash();
  uint bundleId = __bundleIdOfOfferId[olKeyHash][order.offerId];
  BundledOffer[] memory bundle = __bundles[bundleId][IERC20(order.olKey.outbound_tkn)];
  // not deprovisionning to save gas
  _retractBundle(bundle, IERC20(order.olKey.outbound_tkn), olKeyHash, false);
  return super.__posthookFallback__(order, result);
}
```

Fixed in commits d7149c5c421b6a8b069467fe9abebd9f039f13b3 and 20dee5b4db4780397a6c365970129f91fcff377d.

7.8 [Low] Updated offers after retraction can use an outdated volume condition

File(s): `RenegingForwarder.sol`

Description: The `RenegingForwarder` contract introduces a volume condition on offers that couldn't be successfully updated. This condition is used instead of the `offer.gives()`, indicating the actual traded volume of outbound tokens. However, when an offer is retracted, the volume condition of that offer (the `cond.volume`) is not reset to zero. This poses a potential problem as Mangrove permits the reuse of an old `offerId` to post a new offer with updated fields.

Consider the scenario where an offer with `offerId A` is retracted, and its `cond.volume` differs from zero. If the offer owner reuses this `offerId A` to post new offer data, there will be a discrepancy between the user expectations and the actual behavior.

Despite the user anticipating the offer to promise only `offer.gives()`, the offer will utilize the outdated `cond.volume` value as its real promised amount. Moreover, if the posted offer is part of a bundle, all offers within the same bundle will erroneously be updated using the outdated `cond.volume` amount.

Recommendation(s): Consider resetting the volume condition for retracted offers before reusing them. This adjustment can be implemented within the `_updateOffer(...)` function by overriding it in the `RenegingForwarder` contract.

Status: Fixed.

Update from the client: As we decided to move the `_setReneging` execution inside `_updateOffer`, if you want to reuse an offer, you have to call `updateOffer`, thus resetting `cond.volume`.

Fixed in commit 33877b6cdadac156efc9b885e9fa572a70a8b190.

7.9 [Info] Offers from the same `offerList` in a bundle are not updated when `_updateBundle(...)` is called.

File(s): `MangroveAmplifier.sol`

Description: When the amplified offer is partially executed the remaining offers in the bundle have to be updated. The `_updateBundle(...)` function updates the `newOutVolume` for the remaining offers. Since we want to avoid updating the current offer, this function skips the current `olKeyHash`. The skipped offer will be updated in the `__posthookSuccess__(...)` function instead.

The problem arises when a bundle is created with multiple offers from the same `offerList`. These offers will all be skipped because they share the same `olKeyHash` as the offer that is currently being taken. As a result, when these unmodified offers are taken, the maker may trade more outbound tokens than intended or have their offers fail and lose the provision in progress.

Recommendation(s): Consider skipping only the offer being taken or documenting this behavior so users know the strategy does not support this use case.

Status: Acknowledged.

Update from the client: This has not been fixed as the expected behavior is not to use the amplifier twice on the same market. We do not want to programmatically prevent this in order to save some execution gas.

7.10 [Info] `outboundVolume` of bundles can be unintentionally modified

File(s): `MangroveAmplifier.sol`

Description: Offers in a bundle are highly coupled. This means that if one offer is taken, the other offers in the bundle will be retroactively updated to use the remaining `outboundVolume`. The standard way of adjusting the `outboundVolume` is through the `updateBundle(...)` function. However, there also exists an `updateOffer(...)` function in the inherited `RenegingForwarder` contract that allows the user to modify the `gives` of a single offer.

Modifying the `gives` of a single offer (that is part of a bundle) does not break the bundle's functionality. However, when the modified offer is taken, the other offers in the bundle will be retroactively adjusted to use the new `outboundVolume`. This value may potentially be higher than the original value. As a result, the offer may give more than intended.

Recommendation(s): Consider restricting the possibility of updating single offers from a bundle.

Status: Acknowledged.

Update from the client: Gives of single offers are not to be modified independently, even though it's not programmatically enforced. The expected behavior is to create bundles and modify them as a whole. We do not want to programmatically prevent this in order to keep the abstraction from the `Forwarder` more readable.

7.11 [Best Practices] Incorrect NatSpec comments

File(s): `*.sol`

Description: The codebase presents some wrong or outdated comments, below is a non-exhaustive list:

- The comments in `TakerOrderLib` explaining GTC and GTCE orders, have the names flipped;
- In the `IOfferLogic` interface, the NatSpec documentation of the `OfferArgs` structure has an outdated param and dev comment;
- The comments in `RouterProxyFactory` state that the Access Controlled Admin address is immutable but this is not the case, as it can be updated with the `setAdmin(...)` function;

Recommendation(s): Consider updating the comments in the code.

Status: Fixed.

Update from the client: Fixed in commits da50594d4fe7d65b5c921edc71f7183b62b5e0cd, df5d3ad0583d2cdd2434b9c3f63eb89e02c974df, and ab8f911e4a235075f467fca52640734061faf6e7.

7.12 [Best Practices] Unused imports

File(s): `*.sol`

Description: The codebase contains imported files that are not being used:

```
RouterProxyFactory.sol -> IERC20
SimpleRouter.sol -> IERC20
SmartRouter.sol -> TransferLib
AbstractRouter.sol -> IERC20
MangroveAmplifier -> TickLib
```

Recommendation(s): To keep the codebase clean and readable, consider removing unused imports.

Status: Fixed.

Update from the client: Fixed in commit 949464ee314be93f9bc6e14763ab10603a48fefe.

8 Documentation Evaluation

Software documentation refers to the written or visual information describing software's functionality, architecture, design, and implementation. It provides a comprehensive overview of the software system and helps users, developers, and stakeholders understand how the software works, how to use it, and how to maintain it. Software documentation can take different forms, such as user manuals, system manuals, technical specifications, requirements documents, design documents, and code comments. Software documentation is critical in software development, enabling effective communication between developers, testers, users, and other stakeholders. It helps to ensure that everyone involved in the development process has a shared understanding of the software system and its functionality. Moreover, software documentation can improve software maintenance by providing a clear and complete understanding of the software system, making it easier for developers to maintain, modify, and update the software over time. Smart contracts can use various types of software documentation. Some of the most common types include:

- Technical whitepaper: A technical whitepaper is a comprehensive document describing the smart contract's design and technical details. It includes information about the purpose of the contract, its architecture, its components, and how they interact with each other;
- User manual: A user manual is a document that provides information about how to use the smart contract. It includes step-by-step instructions on how to perform various tasks and explains the different features and functionalities of the contract;
- Code documentation: Code documentation is a document that provides details about the code of the smart contract. It includes information about the functions, variables, and classes used in the code, as well as explanations of how they work;
- API documentation: API documentation is a document that provides information about the API (Application Programming Interface) of the smart contract. It includes details about the methods, parameters, and responses that can be used to interact with the contract;
- Testing documentation: Testing documentation is a document that provides information about how the smart contract was tested. It includes details about the test cases that were used, the results of the tests, and any issues that were identified during testing;
- Audit documentation: Audit documentation includes reports, notes, and other materials related to the security audit of the smart contract. This type of documentation is critical in ensuring that the smart contract is secure and free from vulnerabilities.

These types of documentation are essential for smart contract development and maintenance. They help ensure that the contract is properly designed, implemented, and tested, and they provide a reference for developers who need to modify or maintain the contract in the future.

Remarks about Mangrove documentation

The `Mangrove` team has provided documentation about the core mechanism of their protocol and the reviewed strategies. The documentation is public as part of the developers documentation. The reviewed code also contained numerous comments giving more insights about the implementation and design choices. Moreover, the Mangrove team was available to address any inquiries or concerns raised by Nethermind Security.

9 Test Suite Evaluation

9.1 Compilation Output

```
> yarn run build
copyDeploymentAddresses.js:
  Querying mangrove-deployments for core deployments of version ^2.0.0, released or unreleased...
  ...found the following deployments of Mangrove:
    80001: 2.0.1 at 0x80cd6Ef14c23dD3957FD5629141a9d7028557c39
    11155111: 2.0.1 at 0x5B2F8058Df0A0b7744FDb4fD0885FbCD2394194C

  Querying mangrove-deployments for matching strat deployments of version ^2.0.0-b1.0, released or unreleased...
  ...found the following deployments of strats:
    80001, Mangrove v2.0.1 at 0x80cd6Ef14c23dD3957FD5629141a9d7028557c39:
      aaveKandelSeeder: not deployed
      aavePooledRouter: not deployed
      kandelLib: not deployed
      kandelSeeder: not deployed
      mangroveOrderRouter: not deployed
      mangroveOrder: not deployed
    11155111, Mangrove v2.0.1 at 0x5B2F8058Df0A0b7744FDb4fD0885FbCD2394194C:
      aaveKandelSeeder: not deployed
      aavePooledRouter: not deployed
      kandelLib: not deployed
      kandelSeeder: not deployed
      mangroveOrderRouter: not deployed
      mangroveOrder: not deployed

  Copying deployment addresses...
  ...done copying deployment addresses
copyContextAddresses.js: Copying context addresses...
copyContextAddresses.js: ...Done copying context addresses
[] Compiling...
[] Compiling 213 files with 0.8.20
[] Solc 0.8.20 finished in 75.79s
Compiler run successful!
copyArtifacts.js: Copying distribution assets...
copyArtifacts.js: ...Done copying distribution assets
Wrote index.js
```

9.2 Tests Output

```
> forge test
[] Compiling...
No files changed, compilation skipped

Running 2 tests for test/strategies/integration/AccessControl.t.sol:AccessControlTest
[PASS] testCannot_setAdmin() (gas: 18896)
[PASS] test_admin_can_set_admin() (gas: 26679)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 5.85ms

Running 13 tests for test/strategies/integration/MangroveOffer.t.sol:MangroveOfferTest
[PASS] testCannot_call_makerExecute_if_not_Mangrove() (gas: 26304)
[PASS] testCannot_call_makerPosthook_if_not_Mangrove() (gas: 15863)
[PASS] test_Admin_is_deployer() (gas: 9909)
[PASS] test_admin_can_WithdrawAllFromMangrove() (gas: 58703)
[PASS] test_admin_can_withdrawFromMangrove() (gas: 85214)
[PASS] test_approves_reverts_when_erc20_approve_fails() (gas: 17836)
[PASS] test_approves_token() (gas: 55184)
[PASS] test_failed_to_repost_is_logged() (gas: 50825)
[PASS] test_failed_trade_is_logged() (gas: 16586)
[PASS] test_get_fail_reverts() (gas: 163622)
[PASS] test_lastLook_returned_value_is_passed() (gas: 23580)
[PASS] test_setAdmin_logs_SetAdmin() (gas: 15871)
[PASS] test_withdrawFromMangrove_reverts_with_good_reason_if_caller_cannot_receive() (gas: 58541)
Test result: ok. 13 passed; 0 failed; 0 skipped; finished in 14.68ms

Running 7 tests for test/strategies/CoreOfferGasbase.gasreq.t.sol:OfferGasBaseTest_Generic_A_B
[PASS] test_gasbase_to_empty_bin_base_quote_failure() (gas: 219422)
[PASS] test_gasbase_to_empty_bin_base_quote_success() (gas: 230340)
[PASS] test_gasbase_to_empty_bin_quote_base_failure() (gas: 227421)
[PASS] test_gasbase_to_empty_bin_quote_base_success() (gas: 238339)
[PASS] test_gasbase_transfers_estimate() (gas: 409614)
[PASS] test_posthook_fail_delta_deep_order_base_quote() (gas: 2061250)
[PASS] test_posthook_fail_delta_deep_order_quote_base() (gas: 2061273)
Test result: ok. 7 passed; 0 failed; 0 skipped; finished in 27.28ms

Running 1 test for test/script/strategies/mangroveOrder/deployers/MangroveOrderDeployer.t.sol:MangroveOrderDeployerTest
[PASS] test_normal_deploy() (gas: 5184880)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 15.03ms

Running 18 tests for test/strategies/MgvAmplifier.gasreq.t.sol:MangroveAmplifierGasreqBaseWithPreviousLockTest
[PASS] test_gasreq_amplified_offer_all_other_market_locked_10() (gas: 514668)
[PASS] test_gasreq_amplified_offer_all_other_market_locked_3() (gas: 316111)
[PASS] test_gasreq_amplified_offer_all_other_market_locked_30() (gas: 1168899)
[PASS] test_gasreq_amplified_offer_all_other_market_locked_5() (gas: 371539)
[PASS] test_gasreq_amplified_offer_all_other_market_locked_50() (gas: 1819893)
[PASS] test_gasreq_amplified_offer_all_other_market_locked_8() (gas: 450413)
[PASS] test_gasreq_amplified_offer_no_market_lock_10() (gas: 479887)
[PASS] test_gasreq_amplified_offer_no_market_lock_3() (gas: 286343)
[PASS] test_gasreq_amplified_offer_no_market_lock_30() (gas: 1162886)
[PASS] test_gasreq_amplified_offer_no_market_lock_5() (gas: 340269)
[PASS] test_gasreq_amplified_offer_no_market_lock_50() (gas: 1963966)
[PASS] test_gasreq_amplified_offer_no_market_lock_8() (gas: 421585)
```

```
[PASS] test_gasreq_amplified_offer_no_market_lock_partial_fill_10() (gas: 763343)
[PASS] test_gasreq_amplified_offer_no_market_lock_partial_fill_3() (gas: 363439)
[PASS] test_gasreq_amplified_offer_no_market_lock_partial_fill_30() (gas: 1783831)
[PASS] test_gasreq_amplified_offer_no_market_lock_partial_fill_5() (gas: 472058)
[PASS] test_gasreq_amplified_offer_no_market_lock_partial_fill_50() (gas: 2626800)
[PASS] test_gasreq_amplified_offer_no_market_lock_partial_fill_8() (gas: 637149)
Test result: ok. 18 passed; 0 failed; 0 skipped; finished in 72.69ms

Running 18 tests for test/strategies/MgvAmplifier.gasreq.t.sol:MangroveAmplifierGasreqBaseTest
[PASS] test_gasreq_amplified_offer_all_other_market_locked_10() (gas: 780709)
[PASS] test_gasreq_amplified_offer_all_other_market_locked_3() (gas: 433271)
[PASS] test_gasreq_amplified_offer_all_other_market_locked_30() (gas: 1776631)
[PASS] test_gasreq_amplified_offer_all_other_market_locked_5() (gas: 530909)
[PASS] test_gasreq_amplified_offer_all_other_market_locked_50() (gas: 2769315)
[PASS] test_gasreq_amplified_offer_all_other_market_locked_8() (gas: 680755)
[PASS] test_gasreq_amplified_offer_no_market_lock_10() (gas: 579070)
[PASS] test_gasreq_amplified_offer_no_market_lock_3() (gas: 365684)
[PASS] test_gasreq_amplified_offer_no_market_lock_30() (gas: 1262089)
[PASS] test_gasreq_amplified_offer_no_market_lock_5() (gas: 419612)
[PASS] test_gasreq_amplified_offer_no_market_lock_50() (gas: 2063187)
[PASS] test_gasreq_amplified_offer_no_market_lock_8() (gas: 512028)
[PASS] test_gasreq_amplified_offer_no_market_lock_partial_fill_10() (gas: 869366)
[PASS] test_gasreq_amplified_offer_no_market_lock_partial_fill_3() (gas: 444480)
[PASS] test_gasreq_amplified_offer_no_market_lock_partial_fill_30() (gas: 1875787)
[PASS] test_gasreq_amplified_offer_no_market_lock_partial_fill_5() (gas: 566248)
[PASS] test_gasreq_amplified_offer_no_market_lock_partial_fill_50() (gas: 2704493)
[PASS] test_gasreq_amplified_offer_no_market_lock_partial_fill_8() (gas: 744569)
Test result: ok. 18 passed; 0 failed; 0 skipped; finished in 54.86ms

Running 20 tests for test/strategies/integration/Direct.t.sol:DirectTest
[PASS] test_admin_can_unbind() (gas: 19564)
[PASS] test_deprovisionOffer_throws_if_wei_transfer_fails() (gas: 196162)
[PASS] test_deprovision_twice_returns_no_fund() (gas: 197845)
[PASS] test_failed_offer_credits_maker(uint256) (runs: 256, : 412052, ~: 412052)
[PASS] test_maker_can_deprovision_Offer() (gas: 196985)
[PASS] test_maker_can_post_newOffer() (gas: 151573)
[PASS] test_maker_can_unbind() (gas: 17514)
[PASS] test_maker_can_updateOffer() (gas: 179757)
[PASS] test_mangrove_can_deprovision_offer() (gas: 198826)
[PASS] test_newOffer_fails_when_provision_is_zero() (gas: 65790)
[PASS] test_only_maker_can_updateOffer() (gas: 167185)
[PASS] test_only_owner_can_post_offers() (gas: 40739)
[PASS] test_owner_balance_is_updated_when_trade_succeeds() (gas: 416312)
[PASS] test_posting_new_offer_with_too_high_gasreq_reverts() (gas: 72127)
[PASS] test_provisionOf_returns_zero_if_offer_does_not_exist() (gas: 23971)
[PASS] test_put_fail_reverts_with_expected_reason() (gas: 202189)
[PASS] test_reposting_fails_with_expected_reason_when_inactive() (gas: 218759)
[PASS] test_reposting_fails_with_expected_reason_when_under_provisioned() (gas: 229434)
[PASS] test_updateOffer_fails_when_provision_is_too_low() (gas: 180000)
[PASS] test_updateOffer_with_funds_increases_owner_balance_on_mangrove() (gas: 222743)
Test result: ok. 20 passed; 0 failed; 0 skipped; finished in 370.08ms

Running 4 tests for test/strategies/MgvOrder.gasreq.t.sol:MangroveOrderGasreqTest_Generic_A_B
[PASS] test_gasreq_repost_on_now_empty_offer_list_with_expiry_base_quote_failure() (gas: 353991)
[PASS] test_gasreq_repost_on_now_empty_offer_list_with_expiry_base_quote_success() (gas: 354094)
[PASS] test_gasreq_repost_on_now_empty_offer_list_with_expiry_quote_base_failure() (gas: 361942)
[PASS] test_gasreq_repost_on_now_empty_offer_list_with_expiry_quote_base_success() (gas: 360359)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 6.91ms

Running 27 tests for test/strategies/integration/Forwarder.t.sol:ForwarderTest
[PASS] test_NewOwnedOffer_logging() (gas: 188891)
[PASS] test_admin_can_unbind() (gas: 28104)
[PASS] test_deprovisionOffer_throws_if_wei_transfer_fails() (gas: 205050)
[PASS] test_deprovision_twice_returns_no_fund() (gas: 208319)
[PASS] test_derived_gasprice_is_accurate_enough(uint256) (runs: 256, : 202980, ~: 204457)
[PASS] test_different_maker_can_post_offers() (gas: 298485)
[PASS] test_failed_offer_credits_maker(uint256) (runs: 256, : 500374, ~: 500374)
[PASS] test_failed_offer_handles_residual_provision(uint96) (runs: 256, : 202914, ~: 205013)
[PASS] test_maker_can_deprovision_Offer() (gas: 203855)
[PASS] test_maker_can_post_newOffer() (gas: 179179)
[PASS] test_maker_can_unbind() (gas: 25821)
[PASS] test_maker_can_updateOffer() (gas: 213077)
[PASS] test_maker_ownership() (gas: 182707)
[PASS] test_mangrove_can_deprovision_offer() (gas: 205591)
```

```
[PASS] test_newOffer_fails_when_provision_is_zero() (gas: 36738)
[PASS] test_only_maker_can_updateOffer() (gas: 195055)
[PASS] test_owner_balance_is_updated_when_trade_succeeds() (gas: 456196)
[PASS] test_posting_new_offer_with_too_high_gasreq_reverts() (gas: 75350)
[PASS] test_provisionOf_returns_zero_if_offer_does_not_exist() (gas: 26385)
[PASS] test_provision_too_high_reverts() (gas: 44007)
[PASS] test_put_fail_reverts_with_expected_reason() (gas: 262498)
[PASS] test_reposting_fails_with_expected_reason_when_inactive() (gas: 231047)
[PASS] test_reposting_fails_with_expected_reason_when_under_provisioned() (gas: 232882)
[PASS] test_updateOffer_fails_when_provision_is_too_low() (gas: 202143)
[PASS] test_updateOffer_with_funds_increases_gasprice() (gas: 258240)
[PASS] test_updateOffer_with_funds_updates_gasprice() (gas: 238110)
[PASS] test_updateOffer_with_no_funds_preserves_gasprice() (gas: 240200)
Test result: ok. 27 passed; 0 failed; 0 skipped; finished in 868.45ms

Running 3 tests for test/strategies/integration/RouterProxyFactory.t.sol:RouterProxyFactoryTest
[PASS] test_computeProxyAddress() (gas: 124700)
[PASS] test_deploysSetsAdmin() (gas: 122283)
[PASS] test_instantiate() (gas: 138078)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 1.76ms

Running 9 tests for test/strategies/kandel/Kandel.gasreq.t.sol:AaveKandelGasreqBaseTest_Generic_A_B
[PASS] test_gasreq_delivery_failure_due_to_allowance_base_quote() (gas: 398101)
[PASS] test_gasreq_delivery_failure_due_to_allowance_quote_base() (gas: 403935)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_base_quote_repost_both() (gas: 664268)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_base_quote_repost_both_fails_due_to_gasprice()
↪  (gas: 670920)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_base_quote_repost_both_setAllowances() (gas:
↪  675183)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_base_quote_repost_both_setGasprice() (gas:
↪  673709)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_quote_base_repost_both() (gas: 668736)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_quote_base_repost_both_fails_due_to_gasprice()
↪  (gas: 670604)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_quote_base_repost_both_setAllowances() (gas:
↪  679684)
Test result: ok. 9 passed; 0 failed; 0 skipped; finished in 15.40ms

Running 4 tests for test/strategies/MgvOrder.gasreq.t.sol:MangroveOrderGasreqTest_Polygon_WETH_DAI
[PASS] test_gasreq_repost_on_now_empty_offer_list_with_expiry_base_quote_failure() (gas: 361876)
[PASS] test_gasreq_repost_on_now_empty_offer_list_with_expiry_base_quote_success() (gas: 333596)
[PASS] test_gasreq_repost_on_now_empty_offer_list_with_expiry_quote_base_failure() (gas: 367244)
[PASS] test_gasreq_repost_on_now_empty_offer_list_with_expiry_quote_base_success() (gas: 339172)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 610.08ms

Running 9 tests for test/strategies/kandel/Kandel.gasreq.t.sol:NoRouterKandelGasreqBaseTest_Polygon_WETH_DAI
[PASS] test_gasreq_delivery_failure_due_to_allowance_base_quote() (gas: 256043)
[PASS] test_gasreq_delivery_failure_due_to_allowance_quote_base() (gas: 262756)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_base_quote_repost_both() (gas: 293575)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_base_quote_repost_both_fails_due_to_gasprice()
↪  (gas: 310279)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_base_quote_repost_both_setAllowances() (gas:
↪  299935)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_base_quote_repost_both_setGasprice() (gas:
↪  305366)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_quote_base_repost_both() (gas: 317070)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_quote_base_repost_both_fails_due_to_gasprice()
↪  (gas: 327793)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_quote_base_repost_both_setAllowances() (gas:
↪  324332)
Test result: ok. 9 passed; 0 failed; 0 skipped; finished in 1.01s

Running 5 tests for test/toy_strategies/Amplifier.t.sol:AmplifierTest
[PASS] test_deprovisionDeadOffers() (gas: 4815755)
[PASS] test_fallback() (gas: 4645375)
[PASS] test_offerAlreadyLive() (gas: 4670008)
[PASS] test_success_fill() (gas: 4812988)
[PASS] test_success_partialFill() (gas: 4860091)
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 981.35ms

Running 7 tests for test/strategies/kandel/KandelSeeder.t.sol:KandelSeederTest
[PASS] test_aave_manager_is_attributed() (gas: 7809)
[PASS] test_logs_new_aaveKandel() (gas: 4733424)
[PASS] test_logs_new_kandel() (gas: 4272568)
```

```
[PASS] test_maker_deploys_kandel() (gas: 4271393)
[PASS] test_maker_deploys_private_aaveKandel() (gas: 5577378)
[PASS] test_maker_deploys_shared_aaveKandel() (gas: 5578200)
[PASS] test_sow_fails_if_market_not_fully_active() (gas: 93816)
Test result: ok. 7 passed; 0 failed; 0 skipped; finished in 1.02s

Running 5 tests for test/toy_strategies/AmplifierForwarder.t.sol:AmplifierForwarderTest
[PASS] test_deprovisionDeadOffers() (gas: 6438954)
[PASS] test_fallback() (gas: 6268494)
[PASS] test_offerAlreadyActive() (gas: 6108777)
[PASS] test_success() (gas: 6398168)
[PASS] test_success_partialFill() (gas: 6415793)
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 541.47ms

Running 7 tests for test/strategies/CoreOfferGasbase.gasreq.t.sol:OfferGasBaseTest_Generic_WETH_DAI
[PASS] test_gasbase_to_empty_bin_base_quote_failure() (gas: 236124)
[PASS] test_gasbase_to_empty_bin_base_quote_success() (gas: 212855)
[PASS] test_gasbase_to_empty_bin_quote_base_failure() (gas: 243332)
[PASS] test_gasbase_to_empty_bin_quote_base_success() (gas: 220854)
[PASS] test_gasbase_transfers_estimate() (gas: 416064)
[PASS] test_posthook_fail_delta_deep_order_base_quote() (gas: 1874246)
[PASS] test_posthook_fail_delta_deep_order_quote_base() (gas: 1858424)
Test result: ok. 7 passed; 0 failed; 0 skipped; finished in 1.02s

Running 77 tests for test/strategies/kandel/Kandel.t.sol:NoRouterKandelTest
[PASS] testFail_ask_partial_fill_noDual_noIncident() (gas: 904186)
[PASS] test_allExternalFunctions_differentCallers_correctAuth() (gas: 4331545)
[PASS] test_all_bids_all_asks_and_back() (gas: 3911942)
[PASS] test_ask_complete_fill() (gas: 974914)
[PASS] test_ask_partial_fill() (gas: 944579)
[PASS] test_ask_partial_fill_existingDual() (gas: 1652358)
[PASS] test_ask_partial_fill_noDual() (gas: 1379010)
[PASS] test_bid_complete_fill() (gas: 1004669)
[PASS] test_bid_partial_fill() (gas: 969195)
[PASS] test_bid_partial_fill_existingDual() (gas: 1641353)
[PASS] test_bid_partial_fill_noDual() (gas: 1364611)
[PASS] test_createDistributionSimple_constantAskBidGives_fuzz(uint256) (runs: 256, : 368814, ~: 334956)
[PASS] test_createDistribution_bothVariable() (gas: 14980)
[PASS] test_createDistribution_constantAskGives() (gas: 1432508)
[PASS] test_createDistribution_constantBidGives() (gas: 813616)
[PASS] test_createDistribution_constantGives() (gas: 206884)
[PASS] test_createDistribution_constantGives_0() (gas: 206544)
[PASS] test_deposit0Funds() (gas: 26532)
[PASS] test_depositFunds(uint96,uint96) (runs: 256, : 381286, ~: 383941)
[PASS] test_extremes_MIN_TICK_ask() (gas: 633622)
[PASS] test_extremes_MIN_TICK_bid() (gas: 633761)
[PASS] test_extremes_MIN_TICK_max_offset_ask() (gas: 638530)
[PASS] test_extremes_MIN_TICK_max_offset_bid() (gas: 640729)
[PASS] test_extremes_many_price_points() (gas: 270043364)
[PASS] test_extremes_max_log_price_ask() (gas: 635736)
[PASS] test_extremes_max_log_price_bid() (gas: 635874)
[PASS] test_extremes_max_offset_ask() (gas: 897659)
[PASS] test_extremes_max_offset_bid() (gas: 910075)
[PASS] test_extremes_outside_range_ask() (gas: 579623)
[PASS] test_extremes_outside_range_bid() (gas: 564789)
[PASS] test_fail_to_update_dual_offer_logs_incident() (gas: 77177)
[PASS] test_heal_1FailedAsk_reposts() (gas: 2185852)
[PASS] test_heal_1FailedBid_reposts() (gas: 1671331)
[PASS] test_heal_3FailedAsk_reposts() (gas: 1870305)
[PASS] test_heal_3FailedBid_reposts() (gas: 1885625)
[PASS] test_init() (gas: 177241)
[PASS] test_marketOrder_dualOfferNew_expectedGasreq() (gas: 109053)
[PASS] test_marketOrder_dualOfferUpdate_expectedGasreq() (gas: 109041)
[PASS] test_offeredVolume_withOffers_returnsSumOfGives() (gas: 979169)
[PASS] test_pending_withOffers_disregardsOfferedVolume() (gas: 1087127)
[PASS] test_pending_withoutOffers_returnsReserveBalance() (gas: 898279)
[PASS] test_populate_allAsks_successful() (gas: 1043816)
[PASS] test_populate_allBids_successful() (gas: 964868)
[PASS] test_populate_can_get_set_params_keeps_offers() (gas: 823411)
[PASS] test_populate_can_repopulate_decreased_size_and_other_params_via_createDistribution() (gas: 2593108)
[PASS] test_populate_can_repopulate_decreased_size_and_other_params_via_populateFromOffset() (gas: 2583448)
[PASS] test_populate_density_too_low_reverted() (gas: 88946)
[PASS] test_populate_existing_offer_is_updated() (gas: 131908)
[PASS] test_populate_retracts_at_zero() (gas: 179916)
```

```
[PASS] test_populate_throws_on_invalid_pricePoints_high() (gas: 17907)
[PASS] test_populate_throws_on_invalid_pricePoints_low() (gas: 17939)
[PASS] test_populate_throws_on_invalid_stepSize_high() (gas: 13231)
[PASS] test_populate_throws_on_invalid_stepSize_low() (gas: 18244)
[PASS] test_populate_throws_on_invalid_stepSize_wrt_pricePoints() (gas: 18355)
[PASS] test_populates_emits() (gas: 164136)
[PASS] test_populates_order_book_correctly() (gas: 2498260)
[PASS] test_posthook_density_too_low_still_posts_to_dual() (gas: 275337)
[PASS] test_posthook_dual_density_too_low_not_posted_via_updateOffer() (gas: 348305)
[PASS] test_reserveBalance_withOffers_returnsFundAmount() (gas: 953207)
[PASS] test_reserveBalance_withoutOffers_returnsFundAmount() (gas: 764745)
[PASS] test_retractAndWithdraw() (gas: 674921)
[PASS] test_retractOffers() (gas: 661527)
[PASS] test_reverseTick() (gas: 2046461)
[PASS] test_setGasprice_invalid_reverts() (gas: 13260)
[PASS] test_setGasprice_valid_setsAndEmits() (gas: 21943)
[PASS] test_setGasreq_invalid_reverts() (gas: 13194)
[PASS] test_setGasreq_valid_setsAndEmits() (gas: 21980)
[PASS] test_step_higher_than_kandel_size_jumps_to_last() (gas: 271538)
[PASS] test_take_full_bid_and_ask_10_times() (gas: 33834232)
[PASS] test_take_new_offer() (gas: 2202732)
[PASS] test_tickSpacing100_aligned_offset_price0() (gas: 22245185)
[PASS] test_tickSpacing100_misaligned_offset_price0() (gas: 22246818)
[PASS] test_transport_below_min_price_accumulates_at_index_0() (gas: 663148)
[PASS] test_withdrawAll() (gas: 432682)
[PASS] test_withdrawAllWithLocal() (gas: 445819)
[PASS] test_withdrawFunds(uint96,uint96) (runs: 256, : 422853, ~: 427836)
[PASS] test_withdrawFundsWithLocal(uint96,uint96) (runs: 256, : 691039, ~: 695462)
Test result: ok. 77 passed; 0 failed; 0 skipped; finished in 1.03s

Running 9 tests for test/strategies/kandel/Kandel.gasreq.t.sol:AaveKandelGasreqBaseTest_Polygon_WETH_DAI
[PASS] test_gasreq_delivery_failure_due_to_allowance_base_quote() (gas: 501891)
[PASS] test_gasreq_delivery_failure_due_to_allowance_quote_base() (gas: 517160)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_base_quote_repost_both() (gas: 715810)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_base_quote_repost_both_fails_due_to_gasprice()
↪  (gas: 732511)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_base_quote_repost_both_setAllowances() (gas:
↪  722166)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_base_quote_repost_both_setGasprice() (gas:
↪  727597)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_quote_base_repost_both() (gas: 750071)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_quote_base_repost_both_fails_due_to_gasprice()
↪  (gas: 760791)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_quote_base_repost_both_setAllowances() (gas:
↪  757329)
Test result: ok. 9 passed; 0 failed; 0 skipped; finished in 107.81ms

Running 24 tests for test/strategies/MgvAmplifier.t.sol:MgvAmplifierTest
[PASS] test_bundle_expiry_before_offer_expiry() (gas: 697168)
[PASS] test_complete_fill_of_one_offer_retracts_bundle() (gas: 775760)
[PASS] test_deployment() (gas: 13955)
[PASS] test_external_bundle_retract() (gas: 568542)
[PASS] test_external_bundle_update_expiry() (gas: 534726)
[PASS] test_external_bundle_update_volume() (gas: 593324)
[PASS] test_external_bundle_update_volume_and_expiry() (gas: 615886)
[PASS] test_external_bundle_update_volume_and_expiry_noop() (gas: 524318)
[PASS] test_griefing_attack_poc() (gas: 932602)
[PASS] test_lock_market() (gas: 33286)
[PASS] test_max_volume_triggered_and_untriggered() (gas: 942666)
[PASS] test_newBundle_inbound_logic_logs() (gas: 594549)
[PASS] test_newBundle_inbound_outbound_logic_logs() (gas: 671077)
[PASS] test_newBundle_no_logic_logs() (gas: 529324)
[PASS] test_newBundle_outbound_logic_logs() (gas: 594149)
[PASS] test_newBundle_with_expiry() (gas: 532017)
[PASS] test_newBundle_with_insufficient_provision_reverts_with_expected_message() (gas: 422696)
[PASS] test_offer_cannot_take_more_than_max_volume() (gas: 672156)
[PASS] test_offer_expiry_before_bundle_expiry() (gas: 695803)
[PASS] test_partial_fill_of_one_offer_updates_bundle_aave_logic() (gas: 1126935)
[PASS] test_partial_fill_of_one_offer_updates_bundle_simple_logic() (gas: 849942)
[PASS] test_retracting_an_offer_in_a_locked_offer_list_makes_it_expire() (gas: 1363654)
[PASS] test_updating_an_offer_in_a_locked_offer_list_makes_it_expire() (gas: 1387243)
[PASS] test_volume_limit_is_reset_after_partial_fill() (gas: 843404)
Test result: ok. 24 passed; 0 failed; 0 skipped; finished in 945.65ms
```

```
Running 9 tests for test/strategies/kandel/Kandel.gasreq.t.sol:NoRouterKandelGasreqBaseTest_Generic_A_B
[PASS] test_gasreq_delivery_failure_due_to_allowance_base_quote() (gas: 266401)
[PASS] test_gasreq_delivery_failure_due_to_allowance_quote_base() (gas: 272235)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_base_quote_repost_both() (gas: 303239)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_base_quote_repost_both_fails_due_to_gasprice()
↪  (gas: 319943)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_base_quote_repost_both_setAllowances() (gas:
↪  316883)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_base_quote_repost_both_setGasprice() (gas:
↪  315025)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_quote_base_repost_both() (gas: 308824)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_quote_base_repost_both_fails_due_to_gasprice()
↪  (gas: 319547)
[PASS] test_gasreq_repost_and_post_dual_first_offer_now_empty_offer_list_quote_base_repost_both_setAllowances() (gas:
↪  322510)
Test result: ok. 9 passed; 0 failed; 0 skipped; finished in 8.76ms

Running 22 tests for test/strategies/integration/SmartRouter.t.sol:SmartRouterTest
[PASS] test_admin_can_unbind() (gas: 28127)
[PASS] test_deprovisionOffer_throws_if_wei_transfer_fails() (gas: 205050)
[PASS] test_deprovision_twice_returns_no_fund() (gas: 208341)
[PASS] test_failed_offer_credits_maker(uint256) (runs: 256, : 458879, ~: 458879)
[PASS] test_maker_can_deprovision_Offer() (gas: 203855)
[PASS] test_maker_can_post_newOffer() (gas: 179245)
[PASS] test_maker_can_unbind() (gas: 25987)
[PASS] test_maker_can_updateOffer() (gas: 213049)
[PASS] test_mangrove_can_deprovision_offer() (gas: 205547)
[PASS] test_newOffer_fails_when_provision_is_zero() (gas: 36717)
[PASS] test_only_maker_can_updateOffer() (gas: 195049)
[PASS] test_owner_balance_is_default_when_no_logic() (gas: 28431)
[PASS] test_owner_balance_is_updated_when_trade_succeeds() (gas: 742337)
[PASS] test_posting_new_offer_with_too_high_gasreq_reverts() (gas: 75350)
[PASS] test_provisionOf_returns_zero_if_offer_does_not_exist() (gas: 26407)
[PASS] test_pull_reverts_are_correctly_handled() (gas: 698237)
[PASS] test_push_correctly_to_aave() (gas: 700183)
[PASS] test_push_reverts_are_correctly_handled() (gas: 607376)
[PASS] test_reposting_fails_with_expected_reason_when_inactive() (gas: 236214)
[PASS] test_reposting_fails_with_expected_reason_when_under_provisioned() (gas: 238071)
[PASS] test_setLogic_logs() (gas: 47117)
[PASS] test_updateOffer_fails_when_provision_is_too_low() (gas: 202094)
Test result: ok. 22 passed; 0 failed; 0 skipped; finished in 769.45ms

Running 41 tests for test/strategies/MgvOrder.t.sol:MgvOrder_Test
[PASS] test_admin() (gas: 18492)
[PASS] test_empirical_offer_gas_cost() (gas: 277334)
[PASS] test_empty_fill_buy_with_resting_order_is_correctly_posted() (gas: 748332)
[PASS] test_empty_fill_sell_with_resting_order_is_correctly_posted() (gas: 738412)
[PASS] test_failing_resting_offer_releases_uncollected_provision() (gas: 336698)
[PASS] test_filled_resting_buy_order_ignores_resting_option_and_returns_value() (gas: 833160)
[PASS] test_filled_resting_buy_order_with_FoK_succeeds_and_returns_provision() (gas: 833077)
[PASS] test_mockup_offerLogic_gas_cost() (gas: 246036)
[PASS] test_mockup_routing_gas_cost() (gas: 118502)
[PASS] test_offer_owner_can_set_expiry() (gas: 31990)
[PASS] test_offer_owner_can_update_offer() (gas: 153799)
[PASS] test_offer_reneges_when_time_is_expired() (gas: 187675)
[PASS] test_offer_succeeds_when_time_is_not_expired() (gas: 276691)
[PASS] test_only_offer_owner_can_set_expiry() (gas: 28398)
[PASS] test_only_offer_owner_can_update_offer() (gas: 28677)
[PASS] test_order_consumption_with_routing_logic_from_aave_and_to_aave() (gas: 1430190)
[PASS] test_order_consumption_with_routing_logic_from_aave_and_to_wallet() (gas: 1247841)
[PASS] test_order_consumption_with_routing_logic_from_wallet_and_to_aave() (gas: 1084630)
[PASS] test_order_reverts_when_expiry_date_is_reached() (gas: 505412)
[PASS] test_partial_fill_buy_with_resting_order_is_correctly_posted() (gas: 994730)
[PASS] test_partial_fill_sell_with_resting_order_below_density() (gas: 890003)
[PASS] test_partial_fill_sell_with_resting_order_is_correctly_posted() (gas: 987287)
[PASS] test_partial_filled_buy_order_from_aave_is_transferred_to_taker() (gas: 1300137)
[PASS] test_partial_filled_buy_order_from_aave_is_transferred_to_taker_on_aave() (gas: 1471194)
[PASS] test_partial_filled_buy_order_is_transferred_to_taker() (gas: 845783)
[PASS] test_partial_filled_buy_order_is_transferred_to_taker_on_aave() (gas: 1069138)
[PASS] test_partial_filled_buy_order_reverts_when_FoK_enabled() (gas: 749770)
[PASS] test_partial_filled_order_returns_bounty() (gas: 1216792)
[PASS] test_partial_filled_returns_value_and_remaining_inbound() (gas: 866491)
[PASS] test_post_only_order_should_be_posted_and_not_make_market_order() (gas: 621712)
[PASS] test_restingOrder_that_fail_to_post_revert_if_no_partialFill() (gas: 872018)
```

```
[PASS] test_resting_buy_offer_can_be_fully_consumed_at_minimum_approval() (gas: 462254)
[PASS] test_resting_buy_offer_can_be_partially_filled() (gas: 284978)
[PASS] test_resting_buy_order_failing_to_post_returns_tokens_and_provision() (gas: 877501)
[PASS] test_resting_buy_order_for_blacklisted_reserve_for_inbound_reverts() (gas: 808038)
[PASS] test_resting_order_with_expiry_date_is_correctly_posted() (gas: 966928)
[PASS] test_taken_resting_order_not_reused_if_live() (gas: 154398)
[PASS] test_taken_resting_order_not_reused_if_not_owned() (gas: 958304)
[PASS] test_taken_resting_order_reused() (gas: 741922)
[PASS] test_taker_unable_to_receive_eth_makes_tx_throw_if_resting_order_could_not_be_posted() (gas: 775647)
[PASS] test_user_can_retract_resting_offer() (gas: 77845)
Test result: ok. 41 passed; 0 failed; 0 skipped; finished in 1.03s

Running 86 tests for test/strategies/kandel/AaveKandel.t.sol:AaveKandelTest
[PASS] testFail_ask_partial_fill_noDual_noIncident() (gas: 1442718)
[PASS] test_allExternalFunctions_differentCallers_correctAuth() (gas: 8347164)
[PASS] test_all_bids_all_asks_and_back() (gas: 5120940)
[PASS] test_ask_complete_fill() (gas: 1420324)
[PASS] test_ask_partial_fill() (gas: 1360858)
[PASS] test_ask_partial_fill_existingDual() (gas: 2287314)
[PASS] test_ask_partial_fill_noDual() (gas: 1795515)
[PASS] test_bid_complete_fill() (gas: 1452012)
[PASS] test_bid_partial_fill() (gas: 1390038)
[PASS] test_bid_partial_fill_existingDual() (gas: 2113480)
[PASS] test_bid_partial_fill_noDual() (gas: 1469476)
[PASS] test_cannot_create_aaveKandel_with_aToken_for_base() (gas: 4164921)
[PASS] test_cannot_create_aaveKandel_with_aToken_for_quote() (gas: 4167981)
[PASS] test_deposit0Funds() (gas: 196262)
[PASS] test_depositFunds(uint96,uint96) (runs: 256, : 775522, ~: 785640)
[PASS] test_extremes_MIN_TICK_ask() (gas: 678261)
[PASS] test_extremes_MIN_TICK_bid() (gas: 678290)
[PASS] test_extremes_MIN_TICK_max_offset_ask() (gas: 683104)
[PASS] test_extremes_MIN_TICK_max_offset_bid() (gas: 685281)
[PASS] test_extremes_many_price_points() (gas: 270086024)
[PASS] test_extremes_max_log_price_ask() (gas: 680310)
[PASS] test_extremes_max_log_price_bid() (gas: 680447)
[PASS] test_extremes_max_offset_ask() (gas: 1096272)
[PASS] test_extremes_max_offset_bid() (gas: 1290667)
[PASS] test_extremes_outside_range_ask() (gas: 624253)
[PASS] test_extremes_outside_range_bid() (gas: 609353)
[PASS] test_fail_to_update_dual_offer_logs_incident() (gas: 102434)
[PASS] test_first_offer_sends_first_puller_to_posthook() (gas: 255329)
[PASS] test_first_puller_posthook_calls_pushAndSupply() (gas: 746268)
[PASS] test_heal_1FailedAsk_reposts() (gas: 2738905)
[PASS] test_heal_1FailedBid_reposts() (gas: 2243300)
[PASS] test_heal_3FailedAsk_reposts() (gas: 2977527)
[PASS] test_heal_3FailedBid_reposts() (gas: 3011274)
[PASS] test_init() (gas: 290606)
[PASS] test_initialize() (gas: 201758)
[PASS] test_liquidity_borrow_clean_attack() (gas: 5040460)
[PASS] test_liquidity_borrow_marketOrder_attack() (gas: 61632323)
[PASS] test_liquidity_flashloan_attack() (gas: 5545453)
[PASS] test_marketOrder_dualOfferNew_expectedGasreq() (gas: 393337)
[PASS] test_marketOrder_dualOfferUpdate_expectedGasreq() (gas: 393391)
[PASS] test_not_first_offer_sends_first_puller_to_posthook_when_buffer_is_small() (gas: 339078)
[PASS] test_not_first_offer_sends_proceed_to_posthook() (gas: 238850)
[PASS] test_offerLogic_sharingLiquidityBetweenStratsBothOffersDonated_offersSucceedAndNoFundsPushedToAave() (gas:
↪  5538185)
[PASS] test_offerLogic_sharingLiquidityBetweenStratsFirstOfferDonated_offersSucceedAndPushesBaseToAave() (gas: 5792076)
[PASS] test_offerLogic_sharingLiquidityBetweenStratsNoDonation_offersSucceedAndFundsPushedToAave() (gas: 5821435)
[PASS] test_offeredVolume_withOffers_returnsSumOfGives() (gas: 1347505)
[PASS] test_pending_withOffers_disregardsOfferedVolume() (gas: 1640680)
[PASS] test_pending_withoutOffers_returnsReserveBalance() (gas: 1437374)
[PASS] test_populate_allAsks_successful() (gas: 1389574)
[PASS] test_populate_allBids_successful() (gas: 1310649)
[PASS] test_populate_can_get_set_params_keeps_offers() (gas: 868102)
[PASS] test_populate_can_repopulate_decreased_size_and_other_params_via_createDistribution() (gas: 3851916)
[PASS] test_populate_can_repopulate_decreased_size_and_other_params_via_populateFromOffset() (gas: 3842273)
[PASS] test_populate_density_too_low_reverted() (gas: 133553)
[PASS] test_populate_existing_offer_is_updated() (gas: 176537)
[PASS] test_populate_retracts_at_zero() (gas: 224524)
[PASS] test_populate_throws_on_invalid_pricePoints_high() (gas: 17885)
[PASS] test_populate_throws_on_invalid_pricePoints_low() (gas: 17939)
[PASS] test_populate_throws_on_invalid_stepSize_high() (gas: 13275)
[PASS] test_populate_throws_on_invalid_stepSize_low() (gas: 18223)
```

```
[PASS] test_populate_throws_on_invalid_stepSize_wrt_pricePoints() (gas: 18421)
[PASS] test_populates_emits() (gas: 208721)
[PASS] test_populates_order_book_correctly() (gas: 2622386)
[PASS] test_posthook_density_too_low_still_posts_to_dual() (gas: 695927)
[PASS] test_posthook_dual_density_too_low_not_posted_via_updateOffer() (gas: 823181)
[PASS] test_reserveBalance_withOffers_returnsFundAmount() (gas: 1506687)
[PASS] test_reserveBalance_withoutOffers_returnsFundAmount() (gas: 1303924)
[PASS] test_retractAndWithdraw() (gas: 1021211)
[PASS] test_retractOffers() (gas: 661549)
[PASS] test_reverseTick() (gas: 2091153)
[PASS] test_setGasprice_invalid_reverts() (gas: 13237)
[PASS] test_setGasprice_valid_setsAndEmits() (gas: 21987)
[PASS] test_setGasreq_invalid_reverts() (gas: 13193)
[PASS] test_setGasreq_valid_setsAndEmits() (gas: 21959)
[PASS] test_sharing_liquidity_between_strats(uint16,uint16) (runs: 256, : 5245116, ~: 5264309)
[PASS] test_step_higher_than_kandel_size_jumps_to_last() (gas: 327305)
[PASS] test_strats_wih_same_admin_but_different_id_do_not_share_liquidity(uint16,uint16) (runs: 256, : 5152160, ~:
↪  5171262)
[PASS] test_take_full_bid_and_ask_10_times() (gas: 39032795)
[PASS] test_take_new_offer() (gas: 3044025)
[PASS] test_tickSpacing100_aligned_offset_price0() (gas: 24707434)
[PASS] test_tickSpacing100_misaligned_offset_price0() (gas: 24709046)
[PASS] test_transport_below_min_price_accumulates_at_index_0() (gas: 733313)
[PASS] test_withdrawAll() (gas: 1028570)
[PASS] test_withdrawAllWithLocal() (gas: 1040174)
[PASS] test_withdrawFunds(uint96,uint96) (runs: 256, : 902573, ~: 910436)
[PASS] test_withdrawFundsWithLocal(uint96,uint96) (runs: 256, : 1160428, ~: 1220476)
Test result: ok. 86 passed; 0 failed; 0 skipped; finished in 2.52s

Running 6 tests for test/toy_strategies/AaveMaker.t.sol:AaveMakerTest
[PASS] test_dry_pool_dai_stable() (gas: 1278099)
[PASS] test_dry_pool_dai_variable() (gas: 1407460)
[PASS] test_dry_pool_usdc_stable() (gas: 1238103)
[PASS] test_dry_pool_usdc_variable() (gas: 1369553)
[PASS] test_dry_pool_weth_stable() (gas: 1201021)
[PASS] test_dry_pool_weth_variable() (gas: 1326671)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 2.61s

Running 44 tests for test/strategies/integration/AavePooledRouter.t.sol:AavePooledRouterTest
[PASS] test_aave_manager_can_exit_market() (gas: 166935)
[PASS] test_aave_manager_can_reenter_market() (gas: 183714)
[PASS] test_admin_can_unbind() (gas: 19587)
[PASS] test_allExternalFunctions_differentCallers_correctAuth() (gas: 11492756)
[PASS] test_claim_rewards() (gas: 218978)
[PASS] test_deposit_on_aave_maintains_reserve_and_total_balance() (gas: 432132)
[PASS] test_deprovisionOffer_throws_if_wei_transfer_fails() (gas: 196162)
[PASS] test_deprovision_twice_returns_no_fund() (gas: 197867)
[PASS] test_donation_in_underlying_increases_user_shares(uint96) (runs: 256, : 581075, ~: 581083)
[PASS] test_failed_offer_credits_maker(uint256) (runs: 256, : 611844, ~: 611844)
[PASS] test_initial_aave_manager_is_deployer() (gas: 10004)
[PASS] test_makerContract_has_initially_zero_shares() (gas: 12333)
[PASS] test_maker_can_deprovision_Offer() (gas: 196986)
[PASS] test_maker_can_post_newOffer() (gas: 151617)
[PASS] test_maker_can_unbind() (gas: 17415)
[PASS] test_maker_can_updateOffer() (gas: 179729)
[PASS] test_mangrove_can_deprovision_offer() (gas: 198871)
[PASS] test_mockup_marketOrder_gas_cost() (gas: 986695)
[PASS] test_newOffer_fails_when_provision_is_zero() (gas: 65791)
[PASS] test_non_strict_pull_transfers_whole_balance() (gas: 450316)
[PASS] test_non_strict_pull_with_insufficient_funds_throws_as_expected() (gas: 307914)
[PASS] test_non_strict_pull_with_large_buffer_does_not_triggers_aave_withdraw() (gas: 540146)
[PASS] test_non_strict_pull_with_small_buffer_triggers_aave_withdraw() (gas: 587254)
[PASS] test_only_maker_can_updateOffer() (gas: 167244)
[PASS] test_overflow_shares(uint96) (runs: 256, : 513495, ~: 514788)
[PASS] test_owner_balance_is_updated_when_trade_succeeds() (gas: 689211)
[PASS] test_posting_new_offer_with_too_high_gasreq_reverts() (gas: 72083)
[PASS] test_provisionOf_returns_zero_if_offer_does_not_exist() (gas: 23993)
[PASS] test_pull0() (gas: 19113)
[PASS] test_pull_token_decreases_last_minter_shares_to_zero() (gas: 308632)
[PASS] test_pull_token_decreases_user_shares() (gas: 506661)
[PASS] test_push0() (gas: 19009)
[PASS] test_push_token_increases_first_minter_shares() (gas: 244416)
[PASS] test_push_token_increases_user_shares() (gas: 473158)
[PASS] test_reposting_fails_with_expected_reason_when_inactive() (gas: 224493)
```

```
[PASS] test_reposting_fails_with_expected_reason_when_under_provisioned() (gas: 235145)
[PASS] test_strict_pull_transfers_only_amount_and_pulls_all_from_aave() (gas: 468476)
[PASS] test_strict_pull_with_insufficient_funds_throws_as_expected() (gas: 80995)
[PASS] test_strict_pull_with_large_buffer_does_not_triggers_aave_withdraw() (gas: 540146)
[PASS] test_strict_pull_with_small_buffer_triggers_aave_withdraw() (gas: 579848)
[PASS] test_supply_error_is_logged() (gas: 2247108)
[PASS] test_underflow_shares_18dec(uint96,uint96) (runs: 256, : 632870, ~: 632873)
[PASS] test_underflow_shares_6dec(uint96,uint96) (runs: 256, : 692012, ~: 692012)
[PASS] test_updateOffer_fails_when_provision_is_too_low() (gas: 179995)
Test result: ok. 44 passed; 0 failed; 0 skipped; finished in 2.54s

Ran 26 test suites: 477 tests passed, 0 failed, 0 skipped (477 total tests)
```

10 About Nethermind

Nethermind is a Blockchain Research and Software Engineering company. Our work touches every part of the web3 ecosystem - from layer 1 and layer 2 engineering, cryptography research, and security to application-layer protocol development. We offer strategic support to our institutional and enterprise partners across the blockchain, digital assets, and DeFi sectors, guiding them through all stages of the research and development process, from initial concepts to successful implementation.

We offer security audits of projects built on EVM-compatible chains and Starknet. We are active builders of the Starknet ecosystem, delivering a node implementation, a block explorer, a Solidity-to-Cairo transpiler, and formal verification tooling. Nethermind also provides strategic support to our institutional and enterprise partners in blockchain, digital assets, and decentralized finance (DeFi). In the next paragraphs, we introduce the company in more detail.

Blockchain Security: At Nethermind, we believe security is vital to the health and longevity of the entire Web3 ecosystem. We provide security services related to Smart Contract Audits, Formal Verification, and Real-Time Monitoring. Our Security Team comprises blockchain security experts in each field, often collaborating to produce comprehensive and robust security solutions. The team has a strong academic background, can apply state-of-the-art techniques, and is experienced in analyzing cutting-edge Solidity and Cairo smart contracts, such as ArgentX and StarkGate (the bridge connecting Ethereum and StarkNet). Most team members hold a Ph.D. degree and actively participate in the research community, accounting for 240+ articles published and 1,450+ citations in Google Scholar. The security team adopts customer-oriented and interactive processes where clients are involved in all stages of the work.

Blockchain Core Development: Our core engineering team, consisting of over 20 developers, maintains, improves, and upgrades our flagship product - the Nethermind Ethereum Execution Client. The client has been successfully operating for several years, supporting both the Ethereum Mainnet and its testnets, and now accounts for nearly a quarter of all synced Mainnet nodes. Our unwavering commitment to Ethereum's growth and stability extends to sidechains and layer 2 solutions. Notably, we were the sole execution layer client to facilitate Gnosis Chain's Merge, transitioning from Aura to Proof of Stake (PoS), and we are actively developing a full-node client to bolster Starknet's decentralization efforts. Our core team equips partners with tools for seamless node set-up, using generated docker-compose scripts tailored to their chosen execution client and preferred configurations for various network types.

DevOps and Infrastructure Management: Our infrastructure team ensures our partners' systems operate securely, reliably, and efficiently. We provide infrastructure design, deployment, monitoring, maintenance, and troubleshooting support, allowing you to focus on your core business operations. Boasting extensive expertise in Blockchain as a Service, private blockchain implementations, and node management, our infrastructure and DevOps engineers are proficient with major cloud solution providers and can host applications in-house or on clients' premises. Our global in-house SRE teams offer 24/7 monitoring and alerts for both infrastructure and application levels. We manage over 5,000 public and private validators and maintain nodes on major public blockchains such as Polygon, Gnosis, Solana, Cosmos, Near, Avalanche, Polkadot, Aptos, and StarkWare L2. Sedge is an open-source tool developed by our infrastructure experts, designed to simplify the complex process of setting up a proof-of-stake (PoS) network or chain validator. Sedge generates docker-compose scripts for the entire validator set-up based on the chosen client, making the process easier and quicker while following best practices to avoid downtime and being slashed.

Cryptography Research: At Nethermind, our Cryptography Research team is dedicated to continuous internal research while fostering close collaboration with external partners. The team has expertise across a wide range of domains, including cryptography protocols, consensus design, decentralized identity, verifiable credentials, Sybil resistance, oracles, and credentials, distributed validator technology (DVT), and Zero-knowledge proofs. This diverse skill set, combined with strong collaboration between our engineering teams, enables us to deliver cutting-edge solutions to our partners and clients.

Smart Contract Development & DeFi Research: Our smart contract development and DeFi research team comprises 40+ world-class engineers who collaborate closely with partners to identify needs and work on value-adding projects. The team specializes in Solidity and Cairo development, architecture design, and DeFi solutions, including DEXs, AMMs, structured products, derivatives, and money market protocols, as well as ERC20, 721, and 1155 token design. Our research and data analytics focuses on three key areas: technical due diligence, market research, and DeFi research. Utilizing a data-driven approach, we offer in-depth insights and outlooks on various industry themes.

Our suite of L2 tooling: Warp is Starknet's approach to EVM compatibility. It allows developers to take their Solidity smart contracts and transpile them to Cairo, Starknet's smart contract language. In the short time since its inception, the project has accomplished many achievements, including successfully transpiling Uniswap v3 onto Starknet using Warp.

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;
- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;
- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

Learn more about us at nethermind.io.

General Advisory to Clients

As auditors, we recommend that any changes or updates made to the audited codebase undergo a re-audit or security review to address potential vulnerabilities or risks introduced by the modifications. By conducting a re-audit or security review of the modified codebase, you can significantly enhance the overall security of your system and reduce the likelihood of exploitation. However, we do not possess the authority or right to impose obligations or restrictions on our clients regarding codebase updates, modifications, or subsequent audits. Accordingly, the decision to seek a re-audit or security review lies solely with you.

Disclaimer

This report is based on the scope of materials and documentation provided by you to Nethermind in order that Nethermind could conduct the security review outlined in **1. Executive Summary** and **2. Audited Files**. The results set out in this report may not be complete nor inclusive of all vulnerabilities. Nethermind has provided the review and this report on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. This report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on this report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.