Security Review Report
NM-0570 Clave Clagg-Contracts

NETHERMIND SECURITY

(June 27, 2025)

Contents

1 Executive Summary 2

2 Audited Files 3

3 Summary of Issues 3

4 System Overview 4
- 4.1 Clagg Admin Components . . . 4
- 4.2 Protocol Integration Components . . . 5
- 4.3 Swapper . . . 5

5 Risk Rating Methodology 6

6 Issues 7
- 6.1 [Medium] ETH can never be rescued from `ClaggRescueAdapter` . . . 7
- 6.2 [Low] Storage collision between `ClaggRescueAdapter` and `ClaggMainAdapter` . . . 8
- 6.3 [Low] `Clagg4626BaseAdapter::getStakeLimit` always returns incorrect value . . . 9
- 6.4 [Best Practices] Initialization of `path` variable is redundant for `IUniswapV3Router::exactInputSingle` . . . 10

7 Documentation Evaluation 11

8 Test Suite Evaluation 12
- 8.1 Compilation Output . . . 12
- 8.2 Tests Output . . . 13

9 About Nethermind 14

1 Executive Summary

This document presents the results of a security review conducted by Nethermind Security for Clave Clagg-contracts. The **Clave Clagg contracts** offer a streamlined interface for users to deposit funds and earn DeFi rewards from the protocols that are integrated with the Clave protocol. Clagg smart contracts can automatically handle reward compounding by claiming accrued rewards and reinvesting.

This audit focuses on recent code refactoring and enhancements. Previously, a single entry point contract managed admin operations via **Timelock**. These responsibilities are now distributed among individual adapters, which the protocol owner can add to or update. The **Swapper** contract has replaced the **Syncswap** contract for token swapping. The Swapper contract now integrates with **UniswapV2/V3** and **AerodromeV2/V3** for swapping tokens. Additionally, a new ERC4626-based adapter has been introduced to facilitate integrations with **Morpho** and **Spark** protocols.

This is a diff audit comprising 1954 lines of the Solidity code. Unlike a full-fledged audit, the focus was on the changes between two commits for the files listed in the scope.

Along this document, we report 4 points of attention, where one is classified `Medium`, two are `Low`, and one is `Informational or Best Practice` severity. The issues are summarized in Fig. 1.

This document is organized as follows. Section 2 presents the files in the scope. Section 3 summarizes the issues. Section 4 presents the system overview. Section 5 discusses the risk rating methodology. Section 6 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 presents the test suite evaluation and automated tools used. Section 9 concludes the document.

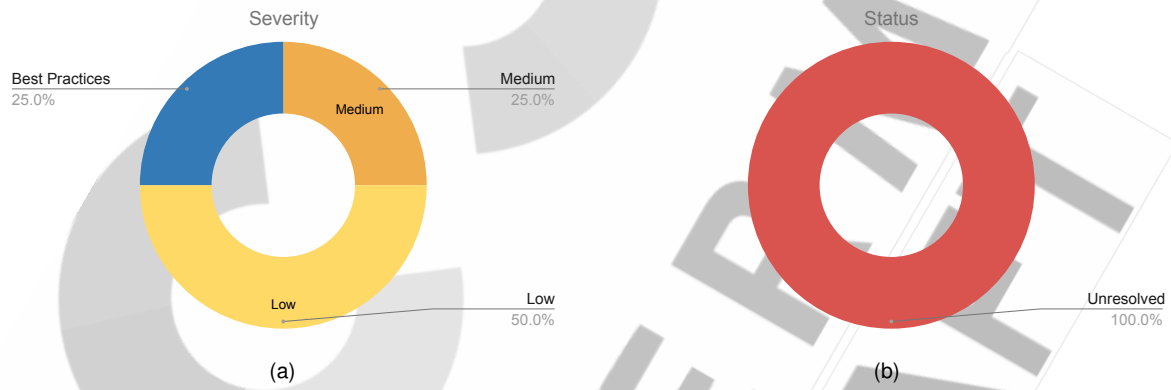

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (1), **Low** (2), **Undetermined** (0), **Informational** (0), **Best Practices** (1). **Distribution of status: Fixed** (0), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (3)

Summary of the Audit

Audit Type	Diff Audit
Initial Report	June 27, 2025
Final Report	-
Repositories	clagg-contracts
Initial Commit	a1ade32b1f4
Final Commit	-
Documentation	Code natspec
Documentation Assessment	Low
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	ClaggMain.sol	57	32	56.1%	17	106
2	IncentiveVault.sol	45	11	24.4%	15	71
3	interfaces/IIncentiveVault.sol	8	24	300.0%	5	37
4	interfaces/IClaveRegistry.sol	5	5	100.0%	2	12
5	interfaces/IAdapter.sol	29	89	306.9%	22	140
6	interfaces/aave/IRewards.sol	16	53	331.2%	5	74
7	interfaces/aave/IPool.sol	29	82	282.8%	8	119
8	interfaces/ztake/IZtake.sol	9	8	88.9%	6	23
9	interfaces/venus/IRewardsDistributor.sol	4	7	175.0%	1	12
10	interfaces/venus/IVToken.sol	8	7	87.5%	5	20
11	interfaces/venus/IComptroller.sol	4	7	175.0%	1	12
12	interfaces/merkl/IDistributor.sol	4	7	175.0%	1	12
13	adapters/ClaggMainAdapter.sol	42	27	64.3%	14	83
14	adapters/ClaggMerklAdapter.sol	11	9	81.8%	3	23
15	adapters/Clagg4626BaseAdapter.sol	54	9	16.7%	19	82
16	adapters/ClaggSparkAdapter.sol	27	9	33.3%	7	43
17	adapters/ClaggAaveAdapter.sol	103	24	23.3%	35	162
18	adapters/ClaggBaseAdapter.sol	353	105	29.7%	107	565
19	adapters/ClaggVenusAdapter.sol	73	15	20.5%	29	117
20	adapters/ClaggZtakeAdapter.sol	54	9	16.7%	19	82
21	adapters/ClaggRescueAdapter.sol	41	20	48.8%	14	75
22	adapters/ClaggMorphoAdapter.sol	27	9	33.3%	7	43
23	utils/Constants.sol	2	2	100.0%	1	5
24	utils/EnforceOwnership.sol	16	7	43.8%	4	27
25	utils/Swapper.sol	190	51	26.8%	33	274
26	utils/ETHWrapper.sol	14	11	78.6%	4	29
27	utils/TokenCallbackHandler.sol	29	6	20.7%	5	40
28	utils/interfaces/IAerodromeV2Router.sol	25	8	32.0%	5	38
29	utils/interfaces/IWETH.sol	6	5	83.3%	3	14
30	utils/interfaces/IAerodromeV3Router.sol	26	5	19.2%	5	36
31	utils/interfaces/IUniswapV2Router.sol	13	5	38.5%	2	20
32	utils/interfaces/IUniswapV3Router.sol	25	5	20.0%	5	35
33	libraries/LinkedList.sol	271	10	3.7%	38	319
34	libraries/Errors.sol	43	19	44.2%	9	71
35	libraries/WETHHelper.sol	20	16	80.0%	4	40
36	libraries/protocols/AaveAddressHelper.sol	32	26	81.2%	5	63
37	libraries/protocols/MerklAddressHelper.sol	16	15	93.8%	3	34
38	libraries/protocols/VenusAddressHelper.sol	32	27	84.4%	5	64
39	libraries/storage/MainStorageLib.sol	134	29	21.6%	41	204
40	libraries/storage/BaseStorageLib.sol	57	6	10.5%	12	75
	Total	**1954**	**821**	**42.0%**	**526**	**3301**

Note: This is a diff audit, primarily focused on comparing the changes in the files between two versions of the code base. It was not a full-fledged audit of all the files listed above.

3 Summary of Issues

	Finding	Severity	Update
1	ETH can never be rescued from `ClaggRescueAdapter`	Medium	Not Fixed
2	Storage collision between `ClaggRescueAdapter` and `ClaggMainAdapter`	Low	Not Fixed
3	`Clagg4626BaseAdapter::getStakeLimit` always returns incorrect value	Low	Not Fixed
4	Initialization of `path` variable is redundant for `IUniswapV3Router::exactInputSingle`	Best Practices	Not Fixed

4 System Overview

The `ClaggMain` contract is the primary entry point contract. The design utilizes the delegate call mechanism to enable flexible architecture.

Logic is implemented in adapter contracts. The protocol owner can add, remove, or replace adapters, hence provisions for incremental addition of functionality.

The adapter contracts can be categorized into two main classes. Admin Adapters facilitate owner-specific administrative operations, such as configuring FeeVaults, Incentive Vaults, registries, rescuing tokens or even managing adapters. The second class consists of Protocol Integration Adapters, which are specifically designed to interact with DeFi protocols. These adapters abstract the unique interaction flows of each DeFi protocol, providing a unified and generic interface for seamless integration.

The `Swapper` contract is responsible for swapping of tokens. It acts as a routing mechanism, extracting routing instructions from call data parameters to direct calls to `UniswapV2/V3` and `AerodromeV2/V3` protocols for token swapping. This implementation also incorporates chain-specific routing based on the current blockchain id. Furthermore, to support protocols like `Morpho` and `Spark`, a new base adapter extending ERC4626 has been integrated into the protocol.

The ownership tracking is now moved into the main storage. Additionally, an operator role was also added.

The following diagram provides a high-level illustration of the Clave - `Clave Protocol`.

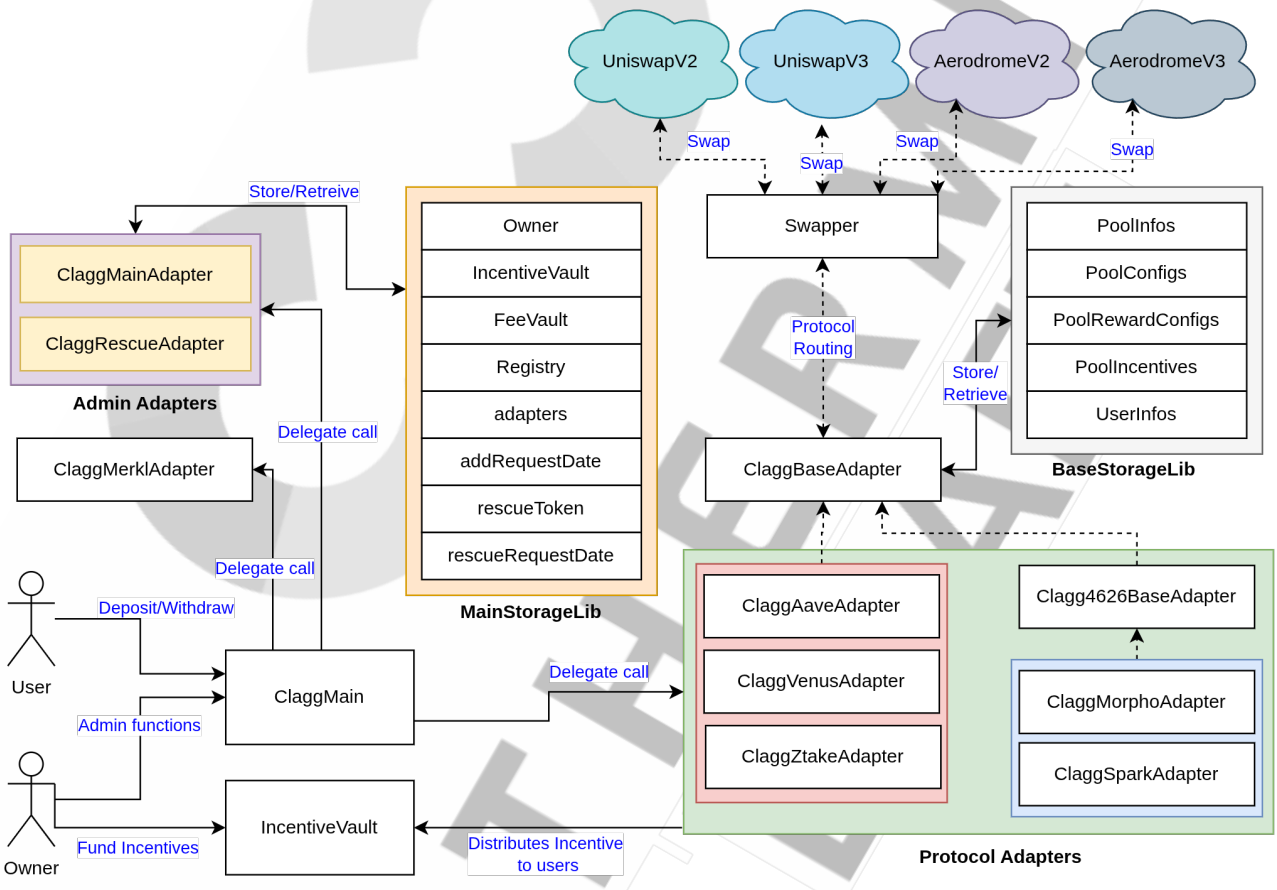

Fig. 2: Clave Clagg-Contracts

4.1 Clagg Admin Components

`ClaggMain`

The `ClaggMain` contract was refactored to separate administrative responsibilities into separate adapters. The `ClaggMain` is now an entry point contract with no knowledge about admin functions making it a generic entry point contract. As a result of the reorganization, the smart contracts in the following were revised.

- **MainStorageLib**: The variables are added to the store registry, owner, operator and rescueToken in the main storage contract. All the ownership verification was based on the owner in the **MainStorageLib** contract.

- **ClaggMainAdapter**: An earlier version of **ClaggMainAdapter** was handling the rescue token related admin operations. The refactoring was done to assign adapter management related responsibilities to **ClaggMainAdapter**.
- **ClaggRescueAdapter**: Token Rescue related responsibilities were segregated into the **ClaggRescueAdapter**. This includes admin functions such adding or removing rescue tokens, as well as performing the rescue operation.

4.2 Protocol Integration Components

`ClaggBaseAdapter`

The `ClaggBaseAdapter`is the main interfacing contract that implements the common functionality for all protocol adapters. This contract handles the swapping related functionality, and hence was revised for the new `swapper` contract.

- **Clagg4626BaseAdapter**: A new Base Adapter that extends ClaggBaseAdapter and interacts with pools based on ERC4626 vault standard.
- **ClaggMorphoAdapter**: A new adapter to integrate with *Morpho* protocol extending *Clagg4626BaseAdapter* . In the case of *Morpho* protocol, rewards cannot be claimed on chain automatically as the rewards are distributed through the Merkle tree. *Clave* protocol has an off-chain component that claims rewards once a week, which are then distributed to the depositors.
- **ClaggSparkAdapter**: A new adapter to integrate with *Spark* protocol extending *Clagg4626BaseAdapter*. *Spark* Protocol does not support reward tokens.

4.3 Swapper

Swapper contract replaces the **SyncSwaper** contract in the earlier version. The **Swapper** contract interacts with **UniswapV2/V3** and **AerodromeV2/V3**. The **Swapper** is chainid aware to retrieve the correct addresses for **UniswapV2/V3** and **AerodromeV2/V3** based on the current chain id.

5 Risk Rating Methodology

The risk rating methodology used by Nethermind Security 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 the finding is to be uncovered and exploited by an attacker. 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 Security 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 pass to the client formally;

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.

6 Issues

6.1 [Medium] ETH can never be rescued from `ClaggRescueAdapter`

File(s): `/adapters/ClaggRescueAdapter.sol`

Description: The `ClaggRescueAdapter` contract includes a rescue function designed to allow recovery of tokens from the contract after they have been whitelisted. The function aims to support rescuing both ERC20 tokens and ETH:

```
function rescue(address token, uint256 amount) external onlyOperatorOrOwner {
1==>    if (token == address(0)) revert Errors.INVALID_TOKEN();

    if (!MainStorageLib.rescueTokenExists(token)) revert Errors.INVALID_RESCUE_TOKEN();

2==>    if (token == address(0)) {
        (bool success,) = payable(msg.sender).call{value: amount}("");
        if (!success) revert Errors.ETH_TRANSFER_FAILED();
    } else {
        IERC20(token).safeTransfer(msg.sender, amount);
    }
}
```

There is a contradictory logic in the implementation:

- First (1), the function rejects a zero address token and immediately reverts with `INVALID_TOKEN` error;
- However, later (2), it attempts to handle the case where `token == address(0)` by sending ETH to the caller;

Due to the initial check, the ETH rescue code is unreachable, effectively preventing ETH from ever being rescued from the contract.

Recommendation(s): Consider removing or modifying the initial zero address check (highlighted at point 1) to allow the function to handle ETH rescue correctly

Status: Unresolved

Update from the client:

6.2 [Low] Storage collision between `ClaggRescueAdapter` and `ClaggMainAdapter`

File(s): `/adapters/ClaggRescueAdapter.sol`, `/adapters/ClaggMainAdapter.sol`

Description: The `ClaggMain` contract acts as the protocol's entry point and delegates all calls to appropriate adapter contracts via a fallback function:

```
fallback() external payable {
    uint256 dataLength = msg.data.length - 20;
    address adapter = address(bytes20(msg.data[dataLength:]));

    if (!MainStorageLib.adapterExists(adapter)) revert Errors.INVALID_ADAPTER();

    bytes calldata data = msg.data[:dataLength];

==>     (bool success, bytes memory result) = adapter.delegatecall(data);
    if (success) {
        assembly {
            return(add(result, 32), mload(result))
        }
    }

    assembly {
        revert(add(result, 32), mload(result))
    }
}
```

Because `delegatecall` is used, the adapter contracts execute in the context of the `ClaggMain` contract, meaning they share the same storage space.

Both `ClaggRescueAdapter` and `ClaggMainAdapter` declare a state variable named `timelock` at the same storage slot (slot 0):

```
contract ClaggMainAdapter is EnforceOwnership {

    event MainTimelockSet(uint96 timelock);

    uint96 public timelock;
    // --SNIP
}

contract ClaggRescueAdapter is EnforceOwnership {
    using SafeERC20 for IERC20;

    event RescueTimelockSet(uint96 timelock);

    uint96 public timelock;
    // --SNIP
}
```

Due to the shared storage layout during `delegatecall` execution, updating `timelock` in one adapter will unintentionally overwrite the timelock value in the other.

Recommendation(s): From natspec, it seems the timelock should be set once, consider making them `immutable` so they will be encoded in the bytecode directly.

Status: Unresolved

Update from the client:

6.3 [Low] `Clagg4626BaseAdapter::getStakeLimit` always returns incorrect value

File(s): `/adapters/getStakeLimit.sol`

Description: The `getStakeLimit` function is designed to return the remaining deposit capacity on the underlying ERC4626 vault (pool). The implementation calls the vault's `maxDeposit` method as follows:

```
function getStakeLimit(address pool) external view override returns (uint256) {
    PoolConfig storage poolConfig = _getPoolConfig(pool);

    address token = poolConfig.token;

    return IERC4626(pool).maxDeposit(token);
}
```

However, the `maxDeposit` function in the ERC4626 standard expects the receiver's address as its parameter, representing the account intending to deposit tokens, not the underlying asset's address. Passing the token address instead leads to incorrect return values from `maxDeposit`, causing `getStakeLimit` to report an inaccurate remaining deposit limit in all cases, depending on the pool implementation.

Recommendation(s): Since `ClaggMain` is the contract making deposits on behalf of users, the address to be passed to `maxDeposit` should be the contract's address instead.

Status: Unresolved

Update from the client:

6.4 [Best Practices] Initialization of `path` variable is redundant for `IUniswapV3Router::exactInputSingle`

File(s): `/utils/Swapper.sol`

Description: The `_swapFrom_UNISWAP_V3` function in the `Swapper` contract handles token swaps using `UniswapV3`. It decodes SwapData to get specific `UniswapV3` parameters, including an `intermediateToken`.

When a swap request does not involve an `intermediate token` (i.e., `uniswapV3SwapData.intermediateToken == address(0)`), the function proceeds with a direct swap using `exactInputSingle`. In this scenario, the `path` variable is initialized with the encoded data received. However, the `exactInputSingle` function does not utilize the `path` variable.

```
function _swapFrom_UNISWAP_V3(address _from, address _to, uint256 _amountIn, SwapData memory _swapData)
        internal
        returns (uint256 _amountOut)
    {
        UniswapV3SwapData memory uniswapV3SwapData = abi.decode(_swapData.data, (UniswapV3SwapData));

        bytes memory path;
        if (uniswapV3SwapData.intermediateToken == address(0)) {
            // @audit-issue This path variable is initialized but never used in the exactInputSingle call.
            path = abi.encodePacked(_from, uniswapV3SwapData.fee, _to);

            _amountOut = IUniswapV3Router(_swapData.router).exactInputSingle(
                IUniswapV3Router.ExactInputSingleParams({
                    tokenIn: _from,
                    tokenOut: _to,
                    fee: uniswapV3SwapData.fee,
                    recipient: address(this),
                    amountIn: _amountIn,
                    amountOutMinimum: 0,
                    sqrtPriceLimitX96: 0
                })
            );
        } else {
            // ...
        }
    }
```

There is no impact on the functioning of the `_swapFrom_UNISWAP_V3` function. Initializing the path variable in this specific condition is unnecessary as the `exactInputSingle` function does not use it. This redundancy does not impact the function's operation but represents unused code/variable.

Recommendation(s): Remove the `path` initialization.

Status: Unresolved

Update from the client:

7 Documentation Evaluation

Software documentation refers to the written or visual information that describes the functionality, architecture, design, and implementation of software. 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 Clave documentation

The `Clave` team has provided a file-diff for the changes made in the new commit.
Based on understanding the protocol from the previous audit, the Nethermind Security team conducted the audit for the changes.
The Clave team addressed the questions and concerns raised by the Nethermind Security team, providing reasoning for changes related to the project's technical aspects.

8 Test Suite Evaluation

8.1 Compilation Output

```
> npx hardhat compile

Compiled 54 Solidity files successfully (evm target: paris).
```

8.2 Tests Output

```
> npx hardhat test --network hardhat

ClaggAave
     should already be added aave adapter (10562ms)
     should set pool config correctly
     should have received USDC tokens
    Deposit - Withdraw USDC
       should deposit USDC to Clagg-Aave
       should withdraw (complete) USDC from Clagg-Aave
       should withdraw (partial) USDC from Clagg-Aave
       should compound rewards in the pool
      Incentive
         should set incentive
         should compound the incentive
         should compound the incentive via swaps (55ms)
    Deposit - Withdraw ETH
       should deposit ETH to Clagg-Aave

  ClaggMain
     should set the contract deployer as the owner
     should set the incentive vault
     should set the owner
     should add initial adapters
     should read from the adapters
     should request to add an adapter (118ms)
     should not add an adapter if the request is not in the timelock (96ms)
     should add a new adapter (86ms)
     should remove an adapter

  ClaggMorpho
     should already be added morpho adapter (795ms)
     should set pool config correctly (97ms)
     should have received USDC tokens
    Deposit - Withdraw
       should deposit USDC to Clagg-Aave (1085ms)
       should withdraw (complete) USDC from Clagg-Aave
       should withdraw (partial) USDC from Clagg-Aave
       should compound rewards in the pool

  ClaggSpark
     should already be added spark adapter (433ms)
     should set pool config correctly (91ms)
     should have received USDC tokens
    Deposit - Withdraw
       should deposit USDC to Clagg-Aave (1093ms)
       should withdraw (complete) USDC from Clagg-Aave
       should withdraw (partial) USDC from Clagg-Aave
       should compound rewards in the pool

  ClaggMain
     should already be added mock adapter (661ms)
    Uniswap v2
       should swap in path: Token A -> Token B (2083ms)
       should swap in path: Token A -> Token B -> Token C (1266ms)
    Uniswap v3
       should swap in path: Token A -> Token B (1245ms)
       should swap in path: Token A -> Token B -> Token C (1798ms)
    Aerodrome v2
       should swap in path: Token A -> Token B (3353ms)
       should swap in path: Token A -> Token B -> Token C (2081ms)
    Aerodrome v3
       should swap in path: Token A -> Token B (684ms)
       should swap in path: Token A -> Token B -> Token C (3691ms)


  43 passing (32s)
```

9 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.