Security Review Report
NM-0599 ETH Strat - Vault

(August 19, 2025)

Contents

1 Executive Summary

This document presents the results of the security review conducted by Nethermind Security for ETH Strat's **ERC4626 Vault** contracts. The system is a yield-generation strategy designed to offer stablecoin-denominated returns. The architecture is centered around two interconnected `ERC4626` vaults and a trusted, off-chain `manager`.

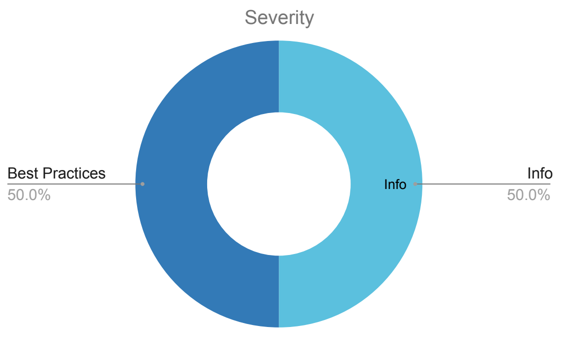
The primary vault, `StratPerpetualBond`, accepts stablecoin deposits from users. These funds are then controlled by the `manager`, who manually executes a discretionary options strategy to generate yield. The second vault, `StakedStratPerpetualBondLP`, is designed to bootstrap liquidity for the primary vault's share token (`SPB`) by rewarding liquidity providers on a decentralized exchange.

A key architectural characteristic is that direct redemptions from the primary vault are disabled. The sole exit mechanism for users is to sell their `SPB` tokens on the secondary market. The system's viability is therefore dependent on the market's confidence in the `manager` and the DEX liquidity incentivized by the second vault.

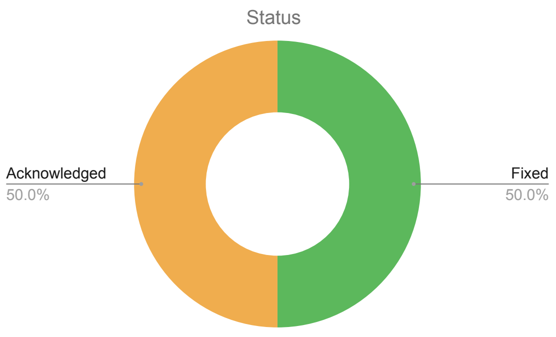
The audit comprises 105 lines of Solidity code. **The audit was performed using** (a) manual analysis of the codebase, and (b) automated analysis tools.

Along this document, we report 4 points of attention, where all four are classified as `Informational` or `Best Practices` 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.



(a)



(b)

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

Summary of the Audit

Audit Type	Security Review
Initial Report	August 13, 2025
Final Report	August 19, 2025
Initial SHA-256 Hash	3cfd632660c4e11d013bc5de7f6a6189e7ef35cf
Final SHA-256 Hash	5266e902feb4bafe2b9fa310c2a4a6cd8830e7e8
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/StratPerpetualBond.sol	96	70	72.9%	26	192
2	src/StratPerpetualBondLPVault.sol	9	9	100.0%	2	20
	Total	**105**	**79**	**75.2%**	**28**	**212**

3 Summary of Issues

	Finding	Severity	Update
1	The `_totalAssets` cannot be written down to reflect realized losses	Info	Acknowledged
2	Vault design does not account for fee-on-transfer tokens	Info	Acknowledged
3	Incorrect comment for `depositCap`	Best Practices	Fixed
4	The `increaseAssetsPerShare(...)` function could emit an event	Best Practices	Fixed

4 System Overview

The protocol is a yield-generation strategy designed to offer stablecoin-denominated returns. The architecture is centered around two interconnected ERC-4626 vaults that work in tandem with a trusted, off-chain `manager`. Users deposit stablecoins into a primary vault, and the `manager` deploys these assets into a discretionary options-based strategy. The resulting yield is then used to reward liquidity providers in a secondary vault, creating an economic flywheel intended to drive value for the protocol's share token.

4.1 The Deposit and Yield Generation Cycle

The primary entry point for users is the `StratPerpetualBond` vault. Users deposit a whitelisted stablecoin as the underlying `asset` and, in return, receive `SPB` tokens, which represent their share of the vault's total assets under management. Upon deposit, the full amount of the stablecoin assets is immediately transferred from the vault to the control of a designated `manager`.

This `manager`, a multi-signature wallet, is responsible for executing the entirety of the protocol's yield strategy. At launch, this involves a manual, off-chain process where the `manager` uses the deposited funds to purchase ETH, which is then used as collateral to sell covered call options on an external derivatives platform. The premium generated from selling these options constitutes the system's yield. A portion of this yield is periodically sent back to the `StratPerpetualBond` vault by calling the `increaseAssetsPerShare(...)` function. This action increases the vault's `totalAssets`, causing the value of each `SPB` share to appreciate.

4.2 The Liquidity and Staking Flywheel

The second core component of the system is the `StakedStratPerpetualBondLP` vault. This vault is designed to incentivize liquidity for the `SPB` token on a secondary market, such as a decentralized exchange (DEX). To participate, a user first provides liquidity by pairing their `SPB` tokens with a stablecoin on a DEX, receiving LP (liquidity provider) tokens in return.

These LP tokens can then be staked in the `StakedStratPerpetualBondLP` vault to earn yield. A significant portion of the premium generated by the `manager` is directed to this vault to be paid out as staking rewards. The exact split of yield between compounding in the main vault and rewarding LPs is at the `manager`'s discretion. This flexibility allows the protocol to dynamically adjust its incentive structure based on prevailing market conditions and is a key reason for the manual, discretionary approach at launch. The rewards are distributed in the form of more LP tokens, creating a compounding return for stakers.

4.3 The SPB Token and Exit Mechanism

The `SPB` token is an ERC-4626 share token that represents a proportional claim on the assets being managed by the protocol. Its value is directly tied to the performance of the `manager`'s options strategy. As yield is compounded back into the `StratPerpetualBond` vault, the underlying asset value per `SPB` token increases.

A key design feature of the protocol is that direct redemptions from the `StratPerpetualBond` vault are disabled by default. Consequently, the sole exit path for an `SPB` holder is to sell their tokens on the secondary market. The sustainability of this model relies on the liquidity incentivized by the `StakedStratPerpetualBondLP` vault. The price and tradability of `SPB` are therefore dependent on the market's confidence in the `manager`'s ability to consistently generate yield and the effectiveness of the staking reward program in maintaining sufficient DEX liquidity.

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 [Info] The _totalAssets cannot be written down to reflect realized losses

File(s): src/StratPerpetualBond.sol

Description: The StratPerpetualBond vault is designed as an ERC-4626 vault where deposited assets are immediately transferred to a manager. This manager is responsible for deploying the assets into a yield-generating strategy, such as selling ETH call options. The vault itself does not hold the underlying assets; its value is tracked by the _totalAssets state variable, which is increased when the manager directs yield back into the vault via the increaseAssetsPerShare(...) function. By design, withdrawals are disabled, and the primary exit path for users is to sell their PBS (vault share) tokens on a secondary market like a DEX.

The potential issue arises if the project decides to enable withdrawals, for instance, to sunset the vault. The _withdraw(...) function calls super._withdraw(...), which in turn attempts to transfer the underlying asset from the vault contract to the user. This transaction will revert if the vault's actual balance of the asset is insufficient to cover the withdrawal amount. Since the vault sends all assets to the manager upon deposit, its balance is typically zero.

```
function _withdraw(
    address caller,
    address receiver,
    address owner,
    uint256 assets,
    uint256 shares
) internal virtual override nonReentrant {
    // @audit Withdrawals are disabled by default.
    if (withdrawalsDisabled) revert WithdrawalsDisabled();

    // @audit-issue This call will attempt an asset transfer and revert if the contract has an insufficient balance.
    super._withdraw(caller, receiver, owner, assets, shares);

    // @audit The total assets are reduced only after a successful withdrawal.
    _totalAssets -= assets;
}
```

If the manager becomes insolvent or its strategy underperforms, it may be unable to return enough assets to the vault to cover all outstanding _totalAssets. In this scenario, enabling withdrawals would create a race condition where the realized losses are not shared proportionally among all shareholders. The first users to withdraw will successfully redeem their shares for assets at a value calculated from the artificially high _totalAssets. These initial withdrawals will deplete the limited asset balance held within the contract. Once this balance is exhausted, subsequent withdrawal attempts will revert, leaving the remaining shareholders unable to recover any funds, as the contract lacks a mechanism to write down the value of _totalAssets to account for the manager's losses.

Recommendation(s): This design places a high degree of trust in the manager. It is recommended to clearly document these risks for all users, emphasizing that the value of PBS tokens and the viability of an exit are entirely dependent on the manager's performance and solvency.

Adding a function to write down the _totalAssets value would allow for the fair socialization of losses, preventing a "race to the exit." However, such a function introduces a significant centralization risk. If the owner's private key were compromised, an attacker could call this function to maliciously reduce or erase the value of all shares, which would have a severe negative impact on the PBS token's economics.

This finding serves an informational purpose for all stakeholders to understand the design trade-offs. The decision of whether to introduce a loss-socialization mechanism at the cost of increased centralization risk is left to the team.

Status: Acknowledged

Update from the client: Noted, on balance will keep as is, if we need to socialise losses, we can always do so by creating a new proportional withdraw in a different smart contract as opposed to using withdraw directly

Withdraw is left for completeness to the standard, and to ensure flexibility *in the event* we want to sunset and return capital easily without using the intended method of users selling shares in the LP pool.

6.2 [Info] Vault design does not account for fee-on-transfer tokens

File(s): `src/StratPerpetualBond.sol`

Description: The vault's accounting mechanism in the `_deposit(...)` function is incompatible with fee-on-transfer tokens. It credits the vault's `_totalAssets` based on the user-specified deposit amount before the actual token transfer occurs. If the vault were deployed with a fee-on-transfer stablecoin, the amount received by the `manager` would be less than the amount recorded, causing the vault's total asset value to become inflated over time. This would lead to shares being backed by fewer assets than the accounting suggests. While uncommon, some stablecoins like USDT have a dormant fee mechanism, which presents a consideration for future deployments.

Recommendation(s): When deploying new instances of the vault with different stablecoins, care must be taken to ensure the chosen asset does not have a fee-on-transfer mechanism to avoid accounting inaccuracies.

Status: Acknowledged

Update from the client: We'll be using USDS (sky.money) - given the probability sky/maker will add a fee is low, we think this is acceptable risk.

6.3 [Best Practices] Incorrect comment for `depositCap`

File(s): `src/StratPerpetualBond.sol`

Description: The `StratPerpetualBond` contract uses a `depositCap` state variable to enforce a limit on the total assets that can be deposited. The NatSpec comment associated with this variable states that a value of `0` signifies no deposit cap.

```solidity
// ...
/// @dev Maximum total assets that can be deposited (0 = no cap)
// @audit-issue This comment is incorrect. A value of 0 effectively blocks all deposits.
uint256 public depositCap;
// ...
```

However, the implementation within the `_deposit(...)` function contradicts this comment. The function checks if the new total assets would exceed the cap.

```solidity
function _deposit(
    address caller,
    address receiver,
    uint256 assets,
    uint256 shares
) internal virtual override {
    // @audit If depositCap is 0, this check will revert for any deposit with assets > 0.
    if (_totalAssets + assets > depositCap) {
        revert DepositCapExceeded();
    }

    super._deposit(caller, receiver, assets, shares);
    // ...
}
```

If `depositCap` is set to `0`, the condition `_totalAssets + assets > depositCap` will be true for any deposit where `assets` is greater than `0`. This will cause the transaction to revert, effectively disabling all deposits.

This discrepancy between the comment and the code's behavior can mislead the contract owner or any developer interacting with the system. An owner attempting to remove the deposit limit by setting `depositCap` to `0` would unintentionally halt all deposits to the vault.

Recommendation(s): Consider correcting the comment for `depositCap` to accurately reflect its functionality.

Status: Fixed

Update from the client: Resolved in commit `dad2bf03a36d41076d211c02954d67f5c426827d`.

6.4 [Best Practices] The `increaseAssetsPerShare(...)` function could emit an event

File(s): `src/StratPerpetualBond.sol`

Description: The `StratPerpetualBond` contract allows a `manager` to add yield to the vault by calling the `increaseAssetsPerShare(...)` function. This function increases the total assets under management (`_totalAssets`), effectively boosting the value of shares held by depositors.

However, the `increaseAssetsPerShare(...)` function does not emit an event after successfully adding assets.

```solidity
function increaseAssetsPerShare(uint256 assets) external {
    // @audit-issue No event is emitted to signal the addition of assets.
    SafeERC20.safeTransferFrom(IERC20(asset()), msg.sender, address(this), assets);
    SafeERC20.safeTransfer(IERC20(asset()), manager, assets);
    _totalAssets += assets;
}
```

The absence of an event makes it difficult for off-chain services, monitoring tools, and users to track the inflow of yield into the vault. This lack of transparency complicates the analysis of the vault's performance over time.

Recommendation(s): Consider defining and emitting an event within the `increaseAssetsPerShare(...)` function.

Status: Fixed

Update from the client: Resolved in commit `6ccf6ff559e7a23f3c71f673ffaf273e7ca1567f`.

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 ETH Strat's documentation

`ETH Strat's` team provided an overview of the main system components during the kick-off call with a detailed explanation of the intended functionalities along with a written specification for the different contracts. Moreover, the team addressed all questions and concerns raised by the Nethermind Security team, providing valuable insights and a comprehensive understanding of the project's technical aspects.

8 Test Suite Evaluation

```
forge test
[] Compiling...
[] Compiling 98 files with Solc 0.8.20
[] Solc 0.8.20 finished in 102.11s
Compiler run successful!
```

```
Ran 4 tests for test/forge/MintableBurnableTokenTest.sol:MintableBurnableTokenTest
[PASS] testBurnByApprovedActor() (gas: 102077)
[PASS] testBurnByHolder() (gas: 92728)
[PASS] testOnlyMintersCanMint() (gas: 77623)
[PASS] testOnlyOwnerCanManageMinters() (gas: 35791)
Suite result: ok. 4 passed; 0 failed; 0 skipped; finished in 13.07ms (498.66µs CPU time)

Ran 3 tests for test/forge/StratPresaleTest.sol:StratPresaleTest
[PASS] testMintRevertsIfNoEthSent() (gas: 11663)
[PASS] testMintSuccessfully() (gas: 181608)
[PASS] testMultiplePresalers() (gas: 466361)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 13.96ms (1.37ms CPU time)

Ran 6 tests for test/forge/StratTreasuryTest.sol:StratTreasuryTest
[PASS] testManageWithdrawer() (gas: 40182)
[PASS] testSetDeployedEth() (gas: 41378)
[PASS] testSetTreasuryVault() (gas: 24853)
[PASS] testTotal() (gas: 42571)
[PASS] testWithdrawAuthorized() (gas: 47493)
[PASS] testWithdrawUnauthorized() (gas: 11776)
Suite result: ok. 6 passed; 0 failed; 0 skipped; finished in 14.48ms (2.78ms CPU time)

Ran 3 tests for test/forge/PresaleTokenRenderer.t.sol:PresaleTokenRendererTest
[PASS] test_render() (gas: 13145)
[PASS] test_renderSvg() (gas: 71353)
[PASS] test_renderSvg_unlocked() (gas: 71865)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 18.98ms (15.36ms CPU time)

Ran 8 tests for test/forge/StratOptionTest.sol:StratOptionTest
[PASS] testBurnByApprovedActor() (gas: 194186)
[PASS] testBurnByHolder() (gas: 171209)
[PASS] testInvalidTimelockOrExpiryReverts() (gas: 217082)
[PASS] testOnlyMintersCanMint() (gas: 202948)
[PASS] testOnlyOwnerCanManageMinters() (gas: 38508)
[PASS] testOnlyOwnerCanSetRenderer() (gas: 217690)
[PASS] testTokenURIReturnsEmptyIfNoRendererSet() (gas: 210914)
[PASS] testTokenURIUsesRendererIfPresent() (gas: 532054)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 19.29ms (1.39ms CPU time)

Ran 15 tests for test/forge/StratOptionExerciseTest.sol:StratOptionExerciseTest
[PASS] testExerciseSuccess() (gas: 169981)
[PASS] testExerciseWithPartialPremintedStrat() (gas: 178883)
[PASS] testExerciseWithSufficientPremintedStrat() (gas: 175004)
[PASS] testRevertIfNotOptionOwner() (gas: 90287)
[PASS] testRevertIfOptionExpired() (gas: 22380)
[PASS] testRevertIfTimelockActive() (gas: 18420)
[PASS] test_exerciseWithPermit() (gas: 310108)
[PASS] test_exerciseWithPermit_callerNotRecipient() (gas: 347923)
[PASS] test_exerciseWithPermit_callerNotRecipient_permitFromRecipient_reverts() (gas: 298844)
[PASS] test_exerciseWithPermit_deadlineHasPassed_reverts() (gas: 243777)
[PASS] test_exerciseWithPermit_deadlineIsZero_reverts() (gas: 241146)
[PASS] test_exerciseWithPermit_deadlineIsZero_spendingApprovalProvided() (gas: 285500)
[PASS] test_exerciseWithPermit_differentOwner_reverts() (gas: 268980)
[PASS] test_exerciseWithPermit_differentSpender_reverts() (gas: 269479)
[PASS] test_exerciseWithPermit_invalidSignature_reverts() (gas: 268395)
Suite result: ok. 15 passed; 0 failed; 0 skipped; finished in 19.51ms (8.88ms CPU time)
```

```
Ran 14 tests for test/forge/StratPerpetualBondTest.sol:StratPerpetualBondTest
[PASS] test_AddAssetsTransfersFundsToManager() (gas: 84594)
[PASS] test_AssetTransferToContractHasNoEffectOnTotalAssets() (gas: 106578)
[PASS] test_Constructor() (gas: 39642)
[PASS] test_ConvertToAssetsUsesCustomTotalAssets() (gas: 167572)
[PASS] test_ConvertToSharesUsesCustomTotalAssets() (gas: 108058)
[PASS] test_DepositCapWithWithdrawals() (gas: 214399)
[PASS] test_DepositFailsWhenCapExceeded() (gas: 185603)
[PASS] test_DepositTransfersFundsToManager() (gas: 141076)
[PASS] test_NonOwnerCannotSetDepositCap() (gas: 18534)
[PASS] test_OnlyOwnerCanSetManager() (gas: 27877)
[PASS] test_OnlyOwnerCanToggleWithdrawals() (gas: 28724)
[PASS] test_OwnerCanSetDepositCap() (gas: 20861)
[PASS] test_WithdrawFailsWhenDisabled() (gas: 162233)
[PASS] test_WithdrawUpdatesTotalAssets() (gas: 204159)
Suite result: ok. 14 passed; 0 failed; 0 skipped; finished in 19.56ms (18.06ms CPU time)

Ran 14 tests for test/forge/StratOptionRedeemUSDNotionalTest.sol:StratOptionRedeemUSDNotionalTest
[PASS] testRedeemSuccessTreasuryGtDebt() (gas: 142888)
[PASS] testRedeemSuccessTreasuryLtDebt() (gas: 147584)
[PASS] testRevertIfNotOptionOwner() (gas: 106905)
[PASS] testRevertIfOptionUnexpired() (gas: 22031)
[PASS] testRevertIfTimelockActive() (gas: 18392)
[PASS] test_redeemCdtForUsdNotionalWithPermit() (gas: 267384)
[PASS] test_redeemCdtForUsdNotionalWithPermit_callerNotRecipient() (gas: 323982)
[PASS] test_redeemCdtForUsdNotionalWithPermit_callerNotRecipient_permitFromRecipient_reverts() (gas: 264893)
[PASS] test_redeemCdtForUsdNotionalWithPermit_deadlineHasPassed_reverts() (gas: 236599)
[PASS] test_redeemCdtForUsdNotionalWithPermit_deadlineIsZero_reverts() (gas: 235091)
[PASS] test_redeemCdtForUsdNotionalWithPermit_deadlineIsZero_spendingApprovalProvided() (gas: 243174)
[PASS] test_redeemCdtForUsdNotionalWithPermit_differentOwner_reverts() (gas: 262095)
[PASS] test_redeemCdtForUsdNotionalWithPermit_differentSpender_reverts() (gas: 262158)
[PASS] test_redeemCdtForUsdNotionalWithPermit_invalidSignature_reverts() (gas: 261049)
Suite result: ok. 14 passed; 0 failed; 0 skipped; finished in 19.86ms (9.58ms CPU time)

Ran 18 tests for test/forge/StratETHShortBondsTest.sol:StratETHShortBondsTest
[PASS] testBond() (gas: 256106)
[PASS] testBondDataInvariants() (gas: 1303117)
[PASS] testBondDeadlineInPastReverts() (gas: 41381)
[PASS] testBondInsufficientOutputReverts() (gas: 77101)
[PASS] testBondRevertIfNoCDTSent() (gas: 12587)
[PASS] testOnlyOwnerCanSetBCV() (gas: 26604)
[PASS] testStrikePrice() (gas: 46464)
[PASS] testStrikePrice_priceNotEqual() (gas: 50142)
[PASS] test_bondWithPermit() (gas: 279878)
[PASS] test_bondWithPermit_callerNotReceipient() (gas: 281505)
[PASS] test_bondWithPermit_callerNotRecipient_permitFromRecipient_reverts() (gas: 240295)
[PASS] test_bondWithPermit_deadlineIsZero_reverts() (gas: 214877)
[PASS] test_bondWithPermit_deadlineIsZero_spendingApprovalProvided() (gas: 253820)
[PASS] test_bondWithPermit_differentOwner_reverts() (gas: 240409)
[PASS] test_bondWithPermit_differentSpender_reverts() (gas: 243310)
[PASS] test_bondWithPermit_invalidSignature_reverts() (gas: 239833)
[PASS] test_bondWithPermit_permitDeadlineHasPassed_reverts() (gas: 216530)
[PASS] test_bondWithPermit_priceNotEqual() (gas: 291219)
Suite result: ok. 18 passed; 0 failed; 0 skipped; finished in 19.96ms (10.72ms CPU time)

Ran 3 tests for test/forge/DateStringTest.sol:DateStringTest
[PASS] test_date_less_than_10() (gas: 6038)
[PASS] test_month_less_than_10() (gas: 5808)
[PASS] test_standard() (gas: 5724)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 45.03ms (6.39ms CPU time)

Ran 9 tests for test/forge/StratETHLongBondsTest.sol:StratETHLongBondsTest
[PASS] testBond() (gas: 299423)
[PASS] testBondDataInvariants() (gas: 1846479)
[PASS] testBondRevertIfBonderAddressIsZero() (gas: 18947)
[PASS] testBondRevertIfNoETHSent() (gas: 12020)
[PASS] testFuzz_strikePriceWithPCFandGCFupdates(uint256,uint256) (runs: 256, : 57279, ~: 57335)
[PASS] testOnlyOwnerCanSetGCF() (gas: 27879)
[PASS] testOnlyOwnerCanSetPCF() (gas: 27321)
[PASS] testStrikeChangesWhenOracleChanges() (gas: 614585)
[PASS] testStrikePrice() (gas: 37295)
Suite result: ok. 9 passed; 0 failed; 0 skipped; finished in 44.57ms (42.96ms CPU time)
Ran 11 test suites in 172.98ms (248.26ms CPU time): 97 tests passed, 0 failed, 0 skipped (97 total tests)
```

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 conducts cutting-edge internal research and collaborates closely with external partners on cryptographic protocols, consensus design, succinct arguments and folding schemes, elliptic curve-based STARK protocols, post-quantum security and zero-knowledge proofs (ZKPs). Our research has led to influential contributions, including Zinc (Crypto '25), Mova, FLI (Asiacrypt '24), and foundational results in Fiat-Shamir security and STARK proof batching. Complementing this theoretical work, our engineering expertise is demonstrated through implementations such as the Latticefold aggregation scheme, the Labrador proof system, zkvm-benchmarks, and Plonk Verifier in Cairo. This combined strength in theory and engineering enables us to deliver cutting-edge cryptographic solutions to 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.

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