Security Review Report
NM-0290 Worldcoin - USD Vault

NETHERMIND SECURITY

(Aug 28, 2024)

Contents

1 Executive Summary

This document outlines the security review conducted by Nethermind Security for the Worldcoin USD Vault. The Vault is a smart contract allowing World App users to deposit USDC tokens and receive sDAI tokens. After some time, the sDAI tokens can be redeemed back to USDC based on the current *Dai Savings Rate*. Liquidity providers ensure that there are enough tokens for the World App users by keeping the token pool balanced. The USDVault contract introduces volume limits, which constrain the number of tokens the user can deposit or redeem within a given timeframe.

The audited code comprises of 473 lines of code written in the Solidity language, and it was performed using (a) manual analysis of the codebase, (b) automated analysis tools, (c) simulation of the smart contract.

Along this document, we report five points of attention, where two are classified as Informational, and three are classified as Best Practice. 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 compilation, tests, and automated tests. Section 9 concludes the document.

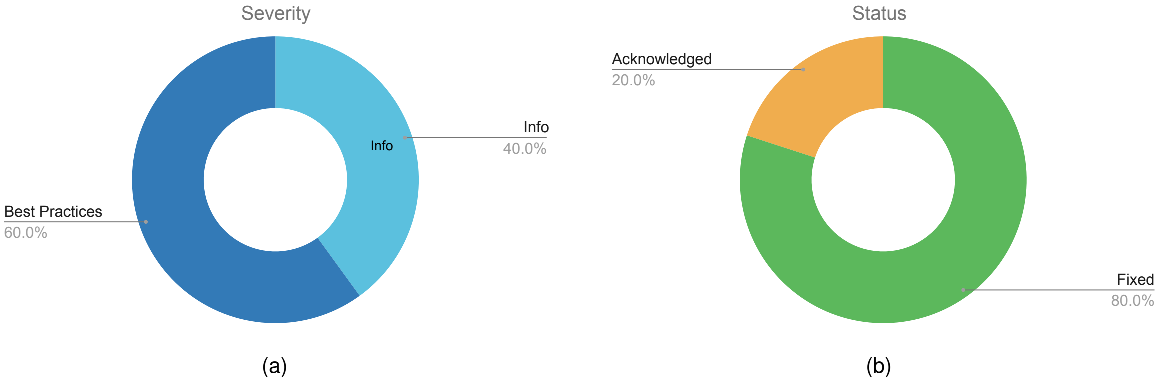

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (0), **Undetermined** (0), **Informational** (2), **Best Practices** (3).
Distribution of status: Fixed (4), **Acknowledged** (1), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Aug 23, 2024
Response from Client	Regular responses during audit engagement
Final Report	Aug 28, 2024
Repository	worldcoin-vault
Commit (Audit)	48595156794ee7c0c846aebce05bb43f2ea017a2
Commit (Final)	f3c1865d2b23f815aa0a21af4de29e7448f05c8d
Documentation Assessment	Low
Test Suite Assessment	Low

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	USDVault.sol	473	165	34.9%	141	779
	Total	**473**	**165**	**34.9%**	**141**	**779**

3 Summary of Issues

	Finding	Severity	Update
1	Incorrect comment	Info	Fixed
2	Users don't have any control over some parameters when using the relayer	Info	Acknowledged
3	The `USDVault` rounds up when giving assets out	Best Practices	Fixed
4	Unused events and errors	Best Practices	Fixed
5	`SafeERC20` is declared but not used	Best Practices	Fixed

4 System Overview

The USDVault contract allows World App users to deposit USDC tokens and earn interest over time. For their deposits, the vault pays the users with the *Savings Dai* token - sDAI, which over time can be redeemed for more USDC. The exact conversion ratio is provided by the DSR Oracle controlled by the Maker DAO governance.

The USDVault contract interacts with the WorldIDAddressBook contract to check the user's verification status. This ensures that only verified users can earn yield based on their deposits.

4.1 Actors of the system

- **Owner** is a privileged USDVault contract's admin. The owner can control the withdrawal fee percentage and change the fee recipient address as well as the USDC token address.
- **Liquidity Providers** are actors responsible for rebalancing the sDAI and USDC token reserves.
- **World App users** deposit their USDC tokens into the USDVault contract and earn interest.
- **Relayer** is a trusted third-party actor that can perform the redemption on behalf of the user so that the user does not have to pay for the gas.

4.2 Deposits and Redemptions

The USDVault contract's entry point for the user is the depositUSDC(...) function. Verified World App users can call this function and receive sDAI tokens in exchange for USDC tokens. If during the USDVault contract's deployment, the USDC_DEPOSIT_LIMIT_DURATION parameter was set to a non-zero value, then the user's deposits are constrained by a deposit volume limit. Any deposits above the limit performed within a specified limit duration will be rejected.

The exchange rate between USDC and sDAI in deposit and redemption operations is based on the *Dai Savings Rate* controlled by Maker DAO governance. The USDVault contract queries the conversion ratio directly from the DSR Oracle contract.

Once the users decide to exit the investment, they sign an off-chain message to allow sDAI token transfers on their behalf. The Worldcoin relayer fulfills their request and calls the redeemSDAI(...) function. Token transfers are facilitated through the Permit2 contract, which ensures that the signed message can only be consumed by the USDVault contract.

Similarly to deposits, if the SDAI_WITHDRAWAL_LIMIT_DURATION parameter was set to a non-zero value, the redemptions are constrained by the volume limit.

4.3 Vault rebalancing

Over time, when users deposit and redeem funds from the USDVault contract, the USDC and sDAI token reserves become imbalanced. The liquidity providers contracted by the Worldcoin team are responsible for rebalancing the vault token holdings. To keep the token ratio around an ideal 1:1 distribution, liquidity providers can call the depositUSDCRebalance(...) and redeemSDAIRebalance(...) functions.

Liquidity providers supply the USDVault contract with initial liquidity by calling the join(...) function. They specify the number of shares they want to receive and provide an appropriate amount of USDC and sDAI tokens.

Liquidity providers can exit the system anytime by calling the exit(...) function. They will receive an equivalent worth of USDC and sDAI tokens in exchange for their shares.

4.4 Oracle risks

The DSR Oracle receives information from Ethereum Mainnet to updates its information. Some delay may occur between when the exchange rate changes in mainnet and it is pushed to the Optimism network.

The oracle is managed by the MakerDAO team in a non-fully-decentralized manner.

The oracle is just receiving message from Ethereum Mainnet so it's address is not expected to be changed. However, because USDVault contract's owner already have multiple ways that could compromise the contract in case of private key leaks, we consider adding a function to be able to change the oracle would not increase surface risk, and would account for possible scenario where Oracle address needs to be changed.

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] Incorrect comment

File(s): `USDVault.sol`

Description: The NatSpec `@notice` comment above the `updateUSDCAddress(...)` states that the function reverts if the new USDC token contract does not have a sufficient liquidity to pay out the users. This is not the case since there is no such check in this function.

Recommendation(s): Consider changing the comment or introducing the missing checks in the `updateUSDCAddress(...)` function.

Status: Fixed.

Update from the client: Function removed in `6d2f23e8d6acf3e9f0b65365ba6b4d7d14443912`.

6.2 [Info] Users don't have any control over some parameters when using the relayer

File(s): `USDVault.sol`

Description: Users can redeem tokens without interacting with the contract directly. This can be done through the relayer provided by Worldcoin. For this, users need to generate a `PERMIT2` signature that will later be used to transfer the `sDAI` tokens from the user to the contract. However, while using this approach, the users don't have guarantees over params `amountOutMin`, `expectedWithdrawalFeeBips`, and `performanceFeeBips`. These parameters are set by the relayer, and they are not part of the sign message provided by the user. A malicious relayer could use these parameters to harm users.

Recommendation(s): Consider adding logic to allow the user to verify these parameters. Optionally, document this behavior so users are aware.

Status: Acknowledged.

Update from Nethermind Security: The withdrawal and performance fee-related logic has been removed entirely in f3c186, but the issue still applies to the `amountOutMin`, which can be controlled by the relayer.

6.3 [Best Practices] The `USDVault` rounds up when giving assets out

File(s): `USDVault.sol`

Description: The `USDVault` contract applies rounding to the nearest integer when depositing and withdrawing `USDC` and `sDAI` respectively. In scenarios where the conversion amount rounds up, the user receives slightly more `sDAI` when depositing or slightly more `USDC` when redeeming. In its current form the accounting logic can't be abused by the user since the token gains caused by rounding are negligible. In the long run however, if no rebalancing were to happen, the last user to redeem his funds might receive a slightly worse exchange rate as compared to other users. As a best practice it is recommended to round in favor of the protocol to reduce the potential area for accounting errors. As described in `EIP-4626`, rounding down should be applied whenever assets are given to the user or when calculating how many shares the user should receive during the deposit.

```
If (1) it's calculating how many shares to issue to a user for a certain amount
of the underlying tokens they provide or (2) it's determining the amount
of the underlying tokens to transfer to them for returning a certain amount
of shares, it should round down.
```

Recommendation(s): Consider rounding down as described in the `EIP-4626`.

Status: Fixed.

Update from the client: Fixed in `b3abf4feef3ba44237fd4417cb2c1f3b8963b9a0`.

6.4 [Best Practices] Unused events and errors

File(s): `USDVault.sol`

Description: The `USDVault` contract declares the `EmptyPool` error and `FundsWithdrawn` event, but they are not used in the codebase.

Recommendation(s): To keep the code clean and readable, consider evaluating if the unused event and error should be used and, if not, remove them from the codebase.

Status: Fixed.

Update from the client: Fixed in `180b0b79276abca4af3e6a6bab5fd2f104848c31`.

6.5 [Best Practices] SafeERC20 is declared but not used

File(s): USDVault.sol

Description: The USDVault contract declares the using SafeERC20 directive for the IERC20 interface to benefit from extra safety while transfering ERC20 tokens. The problem is that the "safe" counterparts of the transfer functions are not used.

Recommendation(s): Consider using the safeTransfer(...) and safeTransferFrom(...) functions as opposed to transfer(...) and transferFrom(...) when handling ERC20 tokens.

Status: Fixed.

Update from the client: Fixed in e37184051099a5edc3a4cfdb3f876d80d004d69e.

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 Worldcoin documentation

The **Worldcoin** team has provided documentation about their protocol in the form of in-line comments within the code. However, these comments did not provide a comprehensive explanation of the architecture and decisions behind various functionalities. Despite this, the Worldcoin team was available to address any questions or concerns from the Nethermind Security team.

8 Test Suite Evaluation

```
forge test --via-ir
[] Compiling...
[] Compiling 13 files with Solc 0.8.25
[] Solc 0.8.25 finished in 28.00s

Ran 37 tests for src/test/USDVault.t.sol:USDVaultTest
[PASS] testFuzz_calculateFee(uint256,uint256) (runs: 256, : 10215, ~: 10215)
[PASS] testFuzz_convertSDAIToUSDC(uint256,uint256) (runs: 256, : 38339, ~: 38345)
[PASS] testFuzz_convertSDAIToUSDC_RevertsIf_InsufficientSDAIAmount(uint256,uint256) (runs: 256, : 38940, ~: 38945)
[PASS] testFuzz_convertUSDCToSDAI(uint256,uint256) (runs: 256, : 39982, ~: 39988)
[PASS] testFuzz_depositUSDC(address,uint256,uint256) (runs: 256, : 237100, ~: 237103)
[PASS] testFuzz_redeemSDAI(uint256,address,uint256,uint256,uint256) (runs: 256, : 373973, ~: 374155)
[PASS] testFuzz_redeemSDAI_RevertsIf_AmountInIsZero(address,uint256,uint256,uint256) (runs: 256, : 10386, ~: 10386)
[PASS] testFuzz_redeemSDAI_RevertsIf_AmountOutMinIsZero(address,uint256,uint256,uint256) (runs: 256, : 10418, ~: 10418)
[PASS] testFuzz_redeemSDAI_RevertsIf_InsufficientBalance(address,uint256,uint256,uint256) (runs: 256, : 278031, ~:
↪  278037)
[PASS] testFuzz_redeemSDAI_RevertsIf_InsufficientOutputAmount(address,uint256,uint256,uint256,uint256,uint256) (runs:
↪  256, : 249363, ~: 249359)
[PASS] testFuzz_redeemSDAI_RevertsIf_MaximumPerformanceFeeExceeded(uint256,address,uint256,uint256,uint256) (runs: 256,
↪  : 26607, ~: 26626)
[PASS] testFuzz_redeemSDAI_RevertsIf_UnexpectedWithdrawalFee(uint256,address,uint256,uint256,uint256) (runs: 256, :
↪  15818, ~: 15818)
[PASS] testFuzz_redeemSDAI_RevertsIf_UserIsUnverified(address,uint256,uint256) (runs: 256, : 21434, ~: 21434)
[PASS] testFuzz_setFeeRecipient(address) (runs: 256, : 25790, ~: 25790)
[PASS] testFuzz_setFeeRecipient_RevertsIf_AddressZero() (gas: 15386)
[PASS] testFuzz_setFeeRecipient_RevertsIf_MsgSenderIsNotOwner(address,address) (runs: 256, : 14507, ~: 14507)
[PASS] testFuzz_setWithdrawalFee(uint256) (runs: 256, : 26643, ~: 26768)
[PASS] testFuzz_setWithdrawalFee_RevertsIf_FeeIsGreaterThanMax(uint256) (runs: 256, : 16238, ~: 16238)
[PASS] testFuzz_setWithdrawalFee_RevertsIf_MsgSenderIsNotOwner(uint256,address) (runs: 256, : 15775, ~: 15873)
[PASS] testFuzz_updateUSDCAddress_RevertsIf_NoBalanceOfFunction(address,uint256) (runs: 256, : 51631, ~: 51631)
[PASS] test_LP_fullFlow() (gas: 337655)
[PASS] test_LP_initAndRebalanceRightAway() (gas: 333556)
[PASS] test_calculateFeeOfMax() (gas: 9885)
[PASS] test_calculateFeeOfOne() (gas: 10174)
[PASS] test_calculateFeeOfTiny() (gas: 9639)
[PASS] test_calculateFeeOfZero() (gas: 10104)
[PASS] test_calculateFee_RevertsIf_Overflow() (gas: 9265)
[PASS] test_convertUSDCToSDAIToUSDC(uint256,uint256) (runs: 256, : 39566, ~: 39566)
[PASS] test_getDSRConversionRate(uint256) (runs: 256, : 37601, ~: 37601)
[PASS] test_getDSRConversionRate_RevertsIf_GtMaxDSRConversionRate(uint256) (runs: 256, : 37863, ~: 37863)
[PASS] test_getDSRConversionRate_RevertsIf_LtMinDSRConversionRate(uint256) (runs: 256, : 38057, ~: 38368)
[PASS] test_redeemSDAI_RevertsIf_VolumeLimitExceeded() (gas: 455165)
[PASS] test_renounceOwnership_RevertsAlways() (gas: 14268)
[PASS] test_renounceOwnership_RevertsIf_MsgSenderIsNotOwner(address) (runs: 256, : 14546, ~: 14546)
[PASS] test_updateUSDC(uint256,address,uint256) (runs: 256, : 801143, ~: 801143)
[PASS] test_updateUSDCAddress_RevertsIf_InvalidDecimals() (gas: 524603)
[PASS] test_updateUSDC_RevertsIf_AddressZero() (gas: 14044)
Suite result: ok. 37 passed; 0 failed; 0 skipped; finished in 419.67ms (1.07s CPU time)

Ran 34 tests for src/test/WLDVault.t.sol:WLDVaultTest
[PASS] testCanDepositForUnverifiedUser() (gas: 143785)
[PASS] testCanDepositIfUnverified() (gas: 141169)
[PASS] testCanRefreshForOtherUser() (gas: 53618)
[PASS] testCanRefreshWithoutDeposit() (gas: 43767)
[PASS] testCannotDepositZeroTokens() (gas: 11550)
[PASS] testCannotDepositZeroTokensForOtherUser() (gas: 13559)
[PASS] testCannotDowngradeVerificationTime() (gas: 175496)
[PASS] testCannotRecoverDepositIfNoDeposit() (gas: 13476)
[PASS] testCannotRenounceOwnership() (gas: 17978)
[PASS] testCannotWithdrawIfNoDeposit() (gas: 13789)
[PASS] testCannotWithdrawLessThanBalance() (gas: 152951)
[PASS] testContractInsolvent() (gas: 155375)
[PASS] testDepositForOtherUser() (gas: 191487)
[PASS] testDepositForOtherUserEarnsInterest() (gas: 162294)
[PASS] testDepositForOtherUserIncreasesExistingDeposit() (gas: 180894)
[PASS] testDepositForOtherUserRespectsCap() (gas: 162578)
[PASS] testDepositYieldIsCapped() (gas: 148400)
[PASS] testInterestIsPausedWhenUnverified() (gas: 172623)
```

```
[PASS] testOwnerCanSetYieldRate() (gas: 29992)
[PASS] testOwnerCanUpdateAddressBook() (gas: 33974)
[PASS] testOwnerCanUpdateMaxYieldAmount() (gas: 25477)
[PASS] testOwnerCanUpdateYieldAccrualDeadline(uint256) (runs: 256, : 48745, ~: 48745)
[PASS] testOwnerCanUpdateYieldSource() (gas: 33686)
[PASS] testRecoverDeposit() (gas: 129148)
[PASS] testSimpleFlow(uint256,uint256) (runs: 256, : 166324, ~: 166615)
[PASS] testUserCanWithdrawAtAnyTime(uint8) (runs: 256, : 126217, ~: 126217)
[FAIL. Reason: assertion failed] testUsersCanIncreaseTheirDeposit() (gas: 180102)
[PASS] testWithdrawAll() (gas: 152546)
[PASS] testWithdrawWithAccruedInterestWhenInsolvent() (gas: 354695)
[PASS] testWithdrawWithSig(uint256,uint256,uint256) (runs: 256, : 171147, ~: 171156)
[PASS] testWithdrawWithSigInvalidSignature(uint256,uint256,uint256) (runs: 256, : 162894, ~: 162894)
[PASS] testWithdrawWithSigNonceReuse(uint256,uint256,uint256) (runs: 256, : 196519, ~: 196519)
[PASS] testYieldAccrualDeadlineAffectsInterestCalculation() (gas: 176695)
[PASS] testYieldAccrualDeadlineDoesNotAffectPrincipal() (gas: 145540)
Suite result: FAILED. 33 passed; 1 failed; 0 skipped; finished in 433.08ms (780.63ms CPU time)

Ran 13 tests for src/test/WorldIDAddressBook.t.sol:WorldIDAddressBookTest
[PASS] testCanReverifyAnytime(uint256) (runs: 256, : 111919, ~: 111919)
[PASS] testCanSwitchVerificationAddressWhenItExpires(address) (runs: 256, : 130943, ~: 130928)
[PASS] testCannotRenounceOwnership() (gas: 16792)
[PASS] testCannotReverifyDifferentAddressBeforeExpiry(address,uint256) (runs: 256, : 128076, ~: 128076)
[PASS] testCannotVerifyWithAProofInTheFuture() (gas: 60286)
[PASS] testCannotVerifyWithInvalidProof() (gas: 92078)
[PASS] testConstructorVerifiesArguments() (gas: 666907)
[PASS] testOwnerCanSetVerificationLength() (gas: 27459)
[PASS] testOwnerCanUpdateGroupId() (gas: 24802)
[PASS] testOwnerCanUpdateMaxProofTime() (gas: 27481)
[PASS] testOwnerCanUpdateRouterAddress() (gas: 30403)
[PASS] testUserCanGetVerified() (gas: 92951)
[PASS] testUserCannotGetVerifiedWithOldProof() (gas: 63102)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 751.07ms (1.40s CPU time)

Ran 5 tests for src/test/USDVault.t.sol:USDVaultForkTest
[PASS] testForkFuzz_depositUSDC(address,uint256) (runs: 100, : 271551, ~: 271554)
[PASS] testForkFuzz_redeemSDAI(uint256) (runs: 100, : 389408, ~: 389412)
[PASS] testForkFuzz_updateUSDCAddress(uint256,address) (runs: 100, : 330215, ~: 330218)
[PASS] testFork_depositUSDC_RevertsIf_VolumeLimitExceeded() (gas: 315499)
[PASS] testFork_updateUSDCAddress_RevertsIf_InvalidDecimals() (gas: 19634)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 48.16s (92.18s CPU time)

Ran 4 test suites in 48.18s (49.77s CPU time): 88 tests passed, 1 failed, 0 skipped (89 total tests)

Failing tests:
Encountered 1 failing test in src/test/WLDVault.t.sol:WLDVaultTest
[FAIL. Reason: assertion failed] testUsersCanIncreaseTheirDeposit() (gas: 180102)

Encountered a total of 1 failing tests, 88 tests succeeded
```

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.