Deployment Verification
NM-0587 Mellow on Ethereum

NETHERMIND
SECURITY

(Nov 14, 2024)

Contents

1 Executive Summary

This document outlines the deployment verification and audit reports' assessment by Nethermind Security for Mellow Flexible Vaults to Ethereum.

The verification reported in this document assesses compliance with the following key areas:

- **Bytecode Verification**: Section 3 provides comprehensive verification of deployed contract bytecode against the audited commit 72f689f965e4ac1a4c2bcfb645a8b5416cf740c7, ensuring all 37 contracts match their expected implementations.
- **Factory Implementation Verification**: Section 4 outlines the verification of factory contract implementations, confirming that correct implementation addresses were proposed and accepted for all 14 factory contracts.
- **Ownership Verification**: Section 5 details the verification of proxy admin ownership across all factory contracts, ensuring proper administrative control is established.
- **Deployment Script Compliance**: Section 6 provides verification that the deployed contracts were created using the correct deployment script parameters, including constructor arguments and external protocol addresses.

Verification Summary

Verification Type	Total Contracts	Verification Status
Bytecode Verification	37	**All Passed**
Implementation Verification	14	**All Passed**
Ownership Verification	14	**All Passed**
Constructor Parameter Verification	37	**All Passed**

=

2 Scope

This report encompasses the verification of the following deployed contracts on Ethereum mainnet:

Contract	Address
Factory Implementation	0x0000000397b71C8f3182Fd40D247330D218fdC72
Factory Factory	0x0000000f9686896836C39cf721141922Ce42639f
Consensus Implementation	0x0000000167598d2C78E2313fD5328E16bD9A0b13
Consensus Factory	0xaEEB06CBd91A18b51a2D30b61477eAeE3a9633C3
DepositQueue Implementation	0x00000006dA9f179BFE250Dd1c51cD2d3581930c8
DepositQueue Factory	0xBB92A7B9695750e1234BaB18F83b73686dd09854
SignatureDepositQueue Implementation	0x00000003887dfBCEbD1e4097Ad89B690de7eFbf9
FeeManager Implementation	0x0000000dE74e5D51651326E0A3e1ACA94bEAF6E1
FeeManager Factory	0xF7223356819Ea48f25880b6c2ab3e907CC336D45
Oracle Implementation	0x0000000F0d3D1c31b72368366A4049C05E291D58
Oracle Factory	0x0CdFf250C7a071fdc72340D820C5C8e29507Aaad
RedeemQueue Implementation	0x0000000285805eac535DADdb9648F1E10DfdC411
RedeemQueue Factory	0xfe76b5fd238553D65Ce6dd0A572C0fda629F8421
SignatureRedeemQueue Implementation	0x0000000b2082667589A16c4cF18e9f923781c471
RiskManager Implementation	0x0000000714cf2851baC1AE2f41871862e9D216fD
RiskManager Factory	0xa51E4FA916b939Fa451520D2B7600c740d86E5A0
TokenizedShareManager Implementation	0x0000000E8eb7173fA1a3ba60eCA325bcB6aaf378
ShareManager Factory	0x952f39AA62E94db3Ad0d1C7D1E43C1a8519E45D8
BasicShareManager Implementation	0x00000005564AAE40D88e2F08dA71CBe156767977
Subvault Implementation	0x0000000E535B4E063f8372933A55470e67910a66
Subvault Factory	0x75FE0d73d3C64cdC1C6449D9F977Be6857c4d011
Verifier Implementation	0x000000047Fc878662006E78D5174FB4285637966
Verifier Factory	0x04B30b1e98950e6A13550d84e991bE0d734C2c61
Vault Implementation	0x0000000615B2771511dAa693aC07BE5622869E01
Vault Factory	0x4E38F679e46B3216f0bd4B314E9C429AFfB1dEE3
BitmaskVerifier	0x0000000263Fb29C3D6B0C5837883519eF05ea20A
EigenLayerVerifier Implementation	0x00000003F82051A8B2F020B79e94C3DC94E89B81
EigenLayerVerifier Factory	0x77A83AcBf7A6df20f1D681b4810437d74AE790F8
ERC20Verifier Implementation	0x00000009207D366cBB8549837F8Ae4bf800Af2D6
ERC20Verifier Factory	0x2e234F4E1b7934d5F4bEAE3fF2FDC109f5C42F1d
SymbioticVerifier Implementation	0x00000000cBC6f5d4348496FfA22Cf014b9DA394B
SymbioticVerifier Factory	0x41C443F10a92D597e6c9E271140BC94c10f5159F
VaultConfigurator	0x000000028be48f9E62E13403480B60C4822C5aa5
BasicRedeemHook	0x0000000637f1b1ccDA4Af2dB6CDDf5e5Ec45fd93
RedirectingDepositHook	0x00000004d3B17e5391eb571dDb8fDF95646ca827
LidoDepositHook	0x000000065d1A7bD71f52886910aaBE6555b7317c
OracleHelper	0x000000005F543c38d5ea6D0bF10A50974Eb55E35

Verification Objectives

The verification process aims to confirm that:

- All deployed contracts correspond to the actual contracts from the GitHub repository at the audited commit hash 72f689f.
- The deployed contracts were deployed using the provided deployment script: `Deploy.s.sol`
- All factory contracts have the correct implementation addresses proposed and accepted
- All factory contracts have proper ownership assigned to the proxy admin
- All constructor parameters match the expected values from the deployment script

Deployment Script Reference

- **Deployment Script**: `scripts/ethereum/Deploy.s.sol`
- **Audited Commit**: 72f689f965e4ac1a4c2bcfb645a8b5416cf740c7

3 Bytecode Verification

Description: The verification process involved comprehensive bytecode comparison between deployed contracts and the audited codebase at commit 72f689f965e4ac1a4c2bcfb645a8b5416cf740c7. This verification ensures that all 37 deployed contracts correspond exactly to the audited versions and were deployed using the correct deployment script parameters.

- **Implementation Contracts**: All 23 implementation contracts were verified to match their audited bytecode, confirming that the deployed code corresponds exactly to the audited commit.
- **Factory Contracts**: All 14 factory contracts were verified to have the correct implementation addresses proposed and accepted, ensuring proper proxy delegation to the audited implementations.
- **Constructor Parameters**: All constructor parameters were verified to match the deployment script specifications, including external protocol addresses and deployment configuration values.
- **Deployment Script Compliance**: The verification confirmed that all contracts were deployed using the provided deployment script with the correct parameters:
  - `DEPLOYMENT_NAME`: "Mellow"
  - `DEPLOYMENT_VERSION`: 1
  - External protocol addresses (Lido, EigenLayer, Symbiotic) match expected values

Verification Results

Contract Type	Total Contracts	Verification Status
Implementation Contracts	23	**All Passed**
Factory Contracts	14	**All Passed**
Total Contracts Verified	37	**All Passed**

Bytecode Verification Summary

4 Factory Implementation Verification

4.1 Implementation Proposal and Acceptance

All factory contracts were verified to have the correct implementation addresses proposed and accepted. This ensures that each factory contract delegates to the proper audited implementation when deploying new contract instances.

Factory Contract	Expected Implementation	Verification Status
Factory Factory	`0x0000000397b71C8f3182Fd40D247330D218fdC72`	Passed
Consensus Factory	`0x0000000167598d2C78E2313fD5328E16bD9A0b13`	Passed
DepositQueue Factory	`0x00000006dA9f179BFE250Dd1c51cD2d3581930c8`	Passed
FeeManager Factory	`0x0000000dE74e5D51651326E0A3e1ACA94bEAF6E1`	Passed
Oracle Factory	`0x0000000F0d3D1c31b72368366A4049C05E291D58`	Passed
RedeemQueue Factory	`0x0000000285805eac535DADdb9648F1E10DfdC411`	Passed
RiskManager Factory	`0x0000000714cf2851baC1AE2f41871862e9D216fD`	Passed
ShareManager Factory	`0x0000000E8eb7173fA1a3ba60eCA325bcB6aaf378`	Passed
Subvault Factory	`0x0000000E535B4E063f8372933A55470e67910a66`	Passed
Verifier Factory	`0x000000047Fc878662006E78D5174FB4285637966`	Passed
Vault Factory	`0x0000000615B2771511dAa693aC07BE5622869E01`	Passed
EigenLayerVerifier Factory	`0x00000003F82051A8B2F020B79e94C3DC94E89B81`	Passed
ERC20Verifier Factory	`0x00000009207D366cBB8549837F8Ae4bf800Af2D6`	Passed
SymbioticVerifier Factory	`0x00000000cBC6f5d4348496FfA22Cf014b9DA394B`	Passed

Factory Implementation Verification Results

4.2 Implementation Storage Verification

The verification process confirmed that all factory contracts correctly store their implementation addresses in the ERC-1967 implementation slot (`0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc`), ensuring proper proxy delegation to the audited implementations.

5 Ownership Verification

5.1 Proxy Admin Ownership

All factory contracts were verified to have proper ownership assigned to the proxy admin address `0x81698f87C6482bF1ce9bFcfC0F103C4A0Adf0Af0`. This ensures that administrative control is properly established and the deployment script executed correctly.

Factory Contract	Owner Address	Verification Status
Factory Factory	`0x81698f87C6482bF1ce9bFcfC0F103C4A0Adf0Af0`	Passed
Consensus Factory	`0x81698f87C6482bF1ce9bFcfC0F103C4A0Adf0Af0`	Passed
DepositQueue Factory	`0x81698f87C6482bF1ce9bFcfC0F103C4A0Adf0Af0`	Passed
FeeManager Factory	`0x81698f87C6482bF1ce9bFcfC0F103C4A0Adf0Af0`	Passed
Oracle Factory	`0x81698f87C6482bF1ce9bFcfC0F103C4A0Adf0Af0`	Passed
RedeemQueue Factory	`0x81698f87C6482bF1ce9bFcfC0F103C4A0Adf0Af0`	Passed
RiskManager Factory	`0x81698f87C6482bF1ce9bFcfC0F103C4A0Adf0Af0`	Passed
ShareManager Factory	`0x81698f87C6482bF1ce9bFcfC0F103C4A0Adf0Af0`	Passed
Subvault Factory	`0x81698f87C6482bF1ce9bFcfC0F103C4A0Adf0Af0`	Passed
Verifier Factory	`0x81698f87C6482bF1ce9bFcfC0F103C4A0Adf0Af0`	Passed
Vault Factory	`0x81698f87C6482bF1ce9bFcfC0F103C4A0Adf0Af0`	Passed
EigenLayerVerifier Factory	`0x81698f87C6482bF1ce9bFcfC0F103C4A0Adf0Af0`	Passed
ERC20Verifier Factory	`0x81698f87C6482bF1ce9bFcfC0F103C4A0Adf0Af0`	Passed
SymbioticVerifier Factory	`0x81698f87C6482bF1ce9bFcfC0F103C4A0Adf0Af0`	Passed

Ownership Verification Results

5.2 Ownership Transfer Verification

The verification confirmed that the deployment script correctly executed the ownership transfer process, ensuring that all factory contracts have their ownership properly assigned to the proxy admin address. This validates that the complete deployment script was executed successfully.

6 Deployment Script Compliance

6.1 Constructor Parameter Verification

All contracts were verified to have been deployed with the correct constructor parameters as specified in the deployment script. This includes verification of:

- **Deployment Configuration**: All contracts use the correct `DEPLOYMENT_NAME` ("Mellow") and `DEPLOYMENT_VERSION` (1) parameters.
- **External Protocol Addresses**: All contracts that interact with external protocols use the correct addresses:
  - `WSTETH`: `0x7f39C581F595B53c5cb19bD0b3f8dA6c935E2Ca0`
  - `WETH`: `0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2`
  - `EIGEN_LAYER_DELEGATION_MANAGER`: `0x39053D51B77DC0d36036Fc1fCc8Cb819df8Ef37A`
  - `EIGEN_LAYER_STRATEGY_MANAGER`: `0x858646372CC42E1A627fcE94aa7A7033e7CF075A`
  - `EIGEN_LAYER_REWARDS_COORDINATOR`: `0x7750d328b314EfFa365A0402CcfD489B80B0adda`
  - `SYMBIOTIC_VAULT_FACTORY`: `0xAEb6bdd95c502390db8f52c8909F703E9Af6a346`
  - `STAKER_REWARDS_FACTORY`: `0xFEB871581C2ab2e1EEe6f7dDC7e6246cFa087A23`
- **Factory Dependencies**: All contracts that depend on factory addresses use the correct factory addresses as specified in the deployment script.
- **Hook Dependencies**: All hook contracts use the correct dependency addresses as specified in the deployment script.

6.2 Deployment Script Execution Verification

The verification process confirmed that the deployment script was executed correctly by verifying:

- All factory contracts have their implementation addresses properly proposed and accepted
- All factory contracts have their ownership transferred to the proxy admin
- All constructor parameters match the expected values from the deployment script
- All external protocol addresses match the expected values from the deployment script

6.3 Deployment Script Reference

- **Deployment Script**: `scripts/ethereum/Deploy.s.sol`
- **Audited Commit**: 72f689f965e4ac1a4c2bcfb645a8b5416cf740c7
- **Deployment Parameters**:
  - `DEPLOYMENT_NAME`: "Mellow"
  - `DEPLOYMENT_VERSION`: 1

6.4 Verification Summary

The comprehensive verification process confirmed that all 37 deployed contracts correspond exactly to the audited codebase and were deployed using the correct deployment script with proper parameters. This ensures that the deployed contracts maintain the security guarantees established during the audit process.

Verification Type	Total Contracts	Verification Status
Bytecode Verification	37	All Passed
Implementation Verification	14	All Passed
Ownership Verification	14	All Passed
Constructor Parameter Verification	37	All Passed

Comprehensive Verification Summary

=

7 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.