
Security Review Report

NM-0854 - Tori



NETHERMIND
SECURITY

(March 26, 2026)

Contents

1	Executive Summary	2
2	Audited Files	3
3	Summary of Issues	3
4	System Overview	4
4.1	Main functionalities	4
4.2	Trust and Access Control Model	4
5	Risk Rating Methodology	5
6	Issues	6
6.0.1	[Medium] Blacklisted users can retrieve silo funds via unstake(), bypassing blacklist restrictions	6
6.0.2	[Low] StakingRewardsDistributor STAKING_VAULT cannot be updated in cases of compromise or if desired.	6
6.0.3	[Low] ToriTimelock delay bounds only enforced at construction, bypassable via inherited updateDelay	7
6.0.4	[Low] TrUsdSilo lacks withdrawal for other assets	7
6.0.5	[Low] removeAllowedToken cannot remove disabled tokens, creating permanently orphaned storage entries	8
6.0.6	[Info] CannotRescueRewardToken error declared but unused; rescueTokens allows owner to extract TrUSD	9
6.0.7	[Info] Cross-chain blacklist systems are independent with no synchronization mechanism	9
6.0.8	[Info] Repeated cooldownAssets/cooldownShares calls reset the cooldown timer for all accumulated assets	10
6.0.9	[Info] StakedTrUSD ERC-4626 max view functions return incorrect values under access restrictions	10
6.0.10	[Info] cancelCooldown() allows zero-share minting causing silent loss of user funds	11
6.0.11	[Info] redistributeLockedAmount() violates MIN_SHARES invariant	11
6.0.12	[Info] reportLoss silently burns partial amount instead of reverting with declared InsufficientAssets error	12
6.0.13	[Info] verifyNonce truncates uint256 nonce to uint128, reducing effective nonce space	12
6.0.14	[Info] verifyStablesLimit divides by trusdAmount before verifyOrder's zero-amount guard, causing uninformative panic	13
7	Documentation Evaluation	14
8	Test Suite Evaluation	15
8.1	Tests Output	15
8.2	Automated Tools	25
8.2.1	AuditAgent	25
9	About Nethermind	26

1 Executive Summary

This document presents the results of the security review conducted by [Nethermind Security](#) for [Tori Finance's](#) EVM contracts.

Tori Finance is a synthetic dollar protocol that issues trUSD, a stablecoin backed by trading positions and stablecoin collateral (USD-C/USDT). Users interact with the protocol through a whitelisted, signature-based minting and redemption flow managed by ToriMinting, which requires EIP-712 or EIP-1271 signed orders and execution by privileged MINTER/REDEEMER roles. Collateral is routed to custodian addresses via configurable ratios, and both per-asset and global per-block limits gate mint/redeem volumes.

trUSD holders can stake into StrUSD via StakedTrUSD, an ERC-4626 vault that accrues yield through an 8-hour linear vesting mechanism. Unstaking requires a cooldown period (default 7 days, admin-configurable up to 90 days) during which assets are held in a dedicated TrUsdSilo contract. The protocol also supports loss reporting, where the REWARDER_ROLE can burn trUSD from the vault to reflect off-chain losses.

The protocol extends across multiple chains via LayerZero v2's OFT (Omnichain Fungible Token) standard:

- **Home chain (Ethereum):** TrUSDAdapter and StakedTrUSDAdapter lock tokens and relay messages to destination chains.
- **Destination chains:** TrUSD0FT and StakedTrUSD0FT mint/burn synthetic representations. All cross-chain transfers are gated by a linear-decay rate limiter.

Governance is enforced through a tiered access control model using OpenZeppelin's AccessControlDefaultAdminRules and a ToriTimeLock with configurable delay, while a GATEKEEPER_ROLE provides emergency circuit-breaker capability.

The audit comprises 1555 lines of the Solidity code. The audit was performed using (a) manual analysis of the codebase, and (b) automated analysis tools.

Along this document, we report fourteen points of attention, where one is classified as Medium, four are classified as Low and nine 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 presents the summary of issues. Section 5 discusses the risk rating methodology. Section 6 details the issues. Section 7 discusses the documentation provided by the client for this audit. Section 8 presents the test suite evaluation and automated tools used. Section 9 concludes the document.

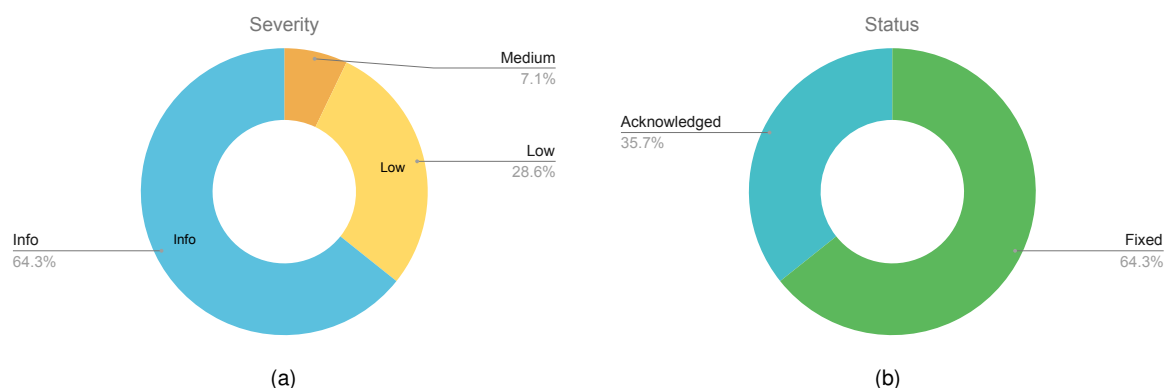


Fig. 1: Distribution of issues: Critical (0), High (0), Medium (1), Low (4), Undetermined (0), Informational (9), Best Practices (0). Distribution of status: Fixed (9), Acknowledged (5), Mitigated (0), Unresolved (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	March 24, 2026
Final Report	March 26, 2026
Initial Commit	c47ec27
Final Commit	4ac4e51
Documentation Assessment	Medium
Test Suite Assessment	Medium

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	core/TrUsdSilo.sol	29	6	20%	9	44
2	core/StakedTrUSD.sol	380	45	11%	93	516
3	core/ToriMinting.sol	566	61	10%	129	750
4	core/StakingRewardsDistributor.sol	90	58	64%	29	177
5	core/TrUSD.sol	74	28	37%	22	124
6	libraries/OFTAdapterOwnable2Step.sol	34	31	91%	10	75
7	libraries/RateLimiter.sol	49	12	24%	13	73
8	libraries/OFTOwnable2Step.sol	35	31	88%	10	76
9	omnichain/StakedTrUSDOfT.sol	54	39	72%	13	106
10	omnichain/TrUSDAdapter.sol	40	4	10%	10	54
11	omnichain/TrUSDOfT.sol	41	4	9%	10	55
12	omnichain/StakedTrUSDAdapter.sol	27	6	22%	7	40
13	governance/ToriTimeLock.sol	43	1	2%	12	56
14	interfaces/ITrUsdSiloDefinitions.sol	4	2	50%	1	7
15	interfaces/IOmnichainOfT.sol	14	21	150%	3	38
16	interfaces/IStakedTrUSD.sol	25	20	80%	8	53
17	interfaces/ITrUSDDefinitions.sol	7	5	71%	2	14
18	interfaces/IStakedTrUSDCoolDown.sol	19	9	47%	9	37
19	interfaces/IRateLimiter.sol	17	18	105%	5	40
20	interfaces/IOmnichainAdapter.sol	7	28	400%	4	39
	Total	1555	429	27.6%	399	2374

3 Summary of Issues

	Finding	Severity	Update
1	Blacklisted users can retrieve silo funds via unstake(), bypassing blacklist restrictions	Medium	Fixed
2	StakingRewardsDistributor STAKING_VAULT can't be updated in cases of compromise or if desired.	Low	Acknowledged
3	ToriTimeLock delay bounds only enforced at construction, bypassable via inherited updateDelay	Low	Fixed
4	TrUsdSilo lacks withdrawal for other assets	Low	Fixed
5	removeAllowedToken cannot remove disabled tokens, creating permanently orphaned storage entries	Low	Fixed
6	CannotRescueRewardToken error declared but unused; rescueTokens allows owner to extract TRUSD	Info	Fixed
7	Cross-chain blacklist systems are independent with no synchronization mechanism	Info	Acknowledged
8	Repeated cooldownAssets/cooldownShares calls reset the cooldown timer for all accumulated assets	Info	Acknowledged
9	StakedTrUSD ERC-4626 max view functions return incorrect values under access restrictions	Info	Acknowledged
10	cancelCooldown() allows zero-share minting causing silent loss of user funds	Info	Fixed
11	redistributeLockedAmount() violates MIN_SHARES invariant	Info	Fixed
12	reportLoss silently burns partial amount instead of reverting with declared InsufficientAssets error	Info	Fixed
13	verifyNonce truncates uint256 nonce to uint128, reducing effective nonce space	Info	Acknowledged
14	verifyStablesLimit divides by trusdAmount before verifyOrder's zero-amount guard, causing uninformative panic	Info	Fixed

4 System Overview

Tori Finance is a synthetic dollar protocol that issues trUSD, a stablecoin backed by trading positions and stablecoin collateral (USD-C/USDT). The smart contracts are deployed on EVM chains and leverage the UUPS proxy pattern with ERC-7201 namespaced storage for upgradeability.

The protocol provides whitelisted, signature-based minting and redemption of trUSD, an ERC-4626 staking vault (strUSD) with reward vesting and cooldown mechanics, and cross-chain token bridging via LayerZero v2's OFT standard.

Governance is enforced through a tiered access control model using OpenZeppelin's AccessControlDefaultAdminRules and a ToriTimelock with configurable delay (1–30 days). A GATEKEEPER_ROLE provides emergency circuit-breaker capability.

4.1 Main functionalities

- **Minting and Redemption:** Users sign EIP-712 or EIP-1271 orders specifying collateral and trUSD amounts. Privileged MINTER/REDEEMER roles submit these orders on-chain via ToriMinting. Collateral is routed to custodian addresses via configurable ratios, and both per-asset and global per-block limits gate mint/redeem volumes. Bitmap-based nonce tracking prevents order replay.
- **Staking and Reward Vesting:** trUSD holders stake into strUSD via StakedTrUSD, an ERC-4626 vault. Rewards are distributed through transferInRewards and vest linearly over 8 hours. The totalAssets() calculation subtracts unvested amounts from the contract's trUSD balance, ensuring the exchange rate increases gradually.
- **Cooldown and Unstaking:** Unstaking requires a cooldown period (default 7 days, admin-configurable up to 90 days). During cooldown, trUSD is transferred to a dedicated TrUsdSilo contract, preventing cooldown assets from being counted in totalAssets(). Users may cancel the cooldown, re-minting shares at the current exchange rate.
- **Loss Reporting:** The REWARDER_ROLE can call reportLoss to burn trUSD from the vault, reflecting off-chain losses. Unvested rewards are cancelled first before touching vested assets, directly reducing the strUSD exchange rate.
- **Blacklist and Redistribution:** Blacklisted addresses (FULL_RESTRICTED_STAKER_ROLE) cannot deposit, withdraw, or transfer. Admin can redistribute blacklisted user shares to another address or burn them (to address(0)) to increase remaining stakers' share value.
- **Cross-chain Bridging:** On Ethereum (home chain), TrUSDAdapter and StakedTrUSDAdapter lock tokens and relay messages via LayerZero v2. On destination chains, TrUSDOfT and StakedTrUSDOfT mint/burn synthetic representations. All cross-chain transfers are gated by a linear-decay rate limiter tracking per-destination in-flight amounts. Inbound strUSD for blacklisted recipients is redirected to the contract owner.
- **Reward Distribution:** StakingRewardsDistributor automates reward distribution to StakedTrUSD with operator delegation, transferring trUSD rewards from a funding source to the vault.

4.2 Trust and Access Control Model

- **Tiered admin roles:** Core contracts use AccessControlDefaultAdminRules with a 1-day admin transfer delay. Sensitive operations (upgrades, adding collateral, setting the minter) are gated behind a TIMELOCK_ADMIN_ROLE backed by ToriTimelock.
- **Signature-based order execution:** ToriMinting relies on off-chain EIP-712/EIP-1271 signatures verified on-chain. The MINTER/REDEEMER backends submit pre-signed user orders, creating a trust assumption on the backend's collateral pricing and order construction.
- **Emergency controls:** The GATEKEEPER_ROLE can zero out global mint/redeem limits and revoke minter/redeemer roles without timelock delay, providing a circuit-breaker mechanism.
- **No ownership renunciation:** renounceRole and renounceOwnership are overridden to revert across all contracts, preventing accidental loss of admin access.

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

6.0.1 [Medium] Blacklisted users can retrieve silo funds via unstake(), bypassing blacklist restrictions

File(s): [contracts/core/StakedTrUSD.sol](#)

Description: The unstake() function does not check whether the caller has the FULL_RESTRICTED_STAKER_ROLE before releasing funds from the TrUsdSilo. When a user initiates a cooldown via cooldownAssets(...) or unstake(), their strUSD shares are burned and the underlying TrUSD is transferred to the TrUsdSilo contract. If the user is subsequently blacklisted, they can still call unstake(...) after the cooldown expires to withdraw the TrUSD to any receiver address.

The blacklist system correctly blocks unstake(...), _deposit(), _withdraw(), and _update() for restricted users, but unstake() was not included in these protections. This creates an inconsistency where the blacklist is enforced across all vault operations except the final silo withdrawal step.

Critically, there is no admin mechanism to seize assets already in the silo. The redistributeLockedAmount(...) function only operates on strUSD share balances, but the blacklisted user's shares were already burned when they initiated the cooldown. The TrUsdSilo contract itself has no admin functions — its withdraw(...) is callable only by StakedTrUSD, and only through the unstake() or unstake() code paths. The rescueTokens(...) function on StakedTrUSD cannot reach the silo since it is a separate contract.

Additionally, a user who observes an addToBlacklist(...) transaction in the mempool can front-run it by calling cooldownAssets(maxWithdraw()) to move all their assets into the silo before the blacklist takes effect, guaranteeing an escape path for the full balance.

```
function unstake(address receiver) external {
    StakedTrUSDStorage storage $ = _getStakedTrUSDStorage();
    UserCooldown storage userCooldown = $.cooldowns[msg.sender];
    uint256 assets = userCooldown.underlyingAmount;

    // @audit-issue No FULL_RESTRICTED_STAKER_ROLE check - blacklisted users can withdraw from silo.
    if (block.timestamp >= userCooldown.cooldownEnd || $.cooldownDuration == 0) {
        userCooldown.cooldownEnd = 0;
        userCooldown.underlyingAmount = 0;
        $.silo.withdraw(receiver, assets);
    } else {
        revert InvalidCooldown();
    }
}
```

```
function cancelCooldown() external nonReentrant {
    // ....
    // @audit Blacklist IS checked here, but NOT in unstake().
    if (hasRole(FULL_RESTRICTED_STAKER_ROLE, msg.sender)) {
        revert OperationNotAllowed();
    }
    // ....
}
```

Recommendation(s): Consider adding a FULL_RESTRICTED_STAKER_ROLE check to the unstake(...) function that reverts if the caller is blacklisted, consistent with the protections already present in unstake(...), _deposit(...), _withdraw(...), and _update(...). Additionally, consider adding an admin function that can redirect a blacklisted user's silo cooldown assets to a designated address, similar to how redistributeLockedAmount(...) handles strUSD shares. This ensures the admin has recourse for assets that entered the silo before the user was blacklisted.

Status: Fixed

Update from the client: Fixed in commit [e1f16c2](#).

6.0.2 [Low] StakingRewardsDistributor STAKING_VAULT cannot be updated in cases of compromise or if desired.

File(s): [contracts/core/StakingRewardsDistributor.sol](#)

Description: StakingRewardsDistributor contract has functions to revoke and update the approval amount for a staking vault in cases where staking vault is compromised or needs updating.

The comments in the file clearly shows that the protocol anticipates such scenarios. But the STAKING_VAULT address is defined as an immutable value (set only in constructor), meaning it will not be updated in cases that requires it. This means for any compromise, StakingRewardsDistributor will need to be deployed again with a new STAKING_VAULT address.

Recommendation(s): Consider making the STAKING_VAULT address updatable with a setter function.

Status: Acknowledged.

Update from the client: Acknowledged.

6.0.3 [Low] ToriTimelock delay bounds only enforced at construction, bypassable via inherited updateDelay

File(s): [contracts/governance/ToriTimelock.sol](#)

Description: ToriTimelock defines MIN_DELAY (1 day) and MAX_DELAY (30 days) constants and enforces them in the constructor. However, the contract inherits TimelockController from OpenZeppelin, which exposes an updateDelay(...) function that allows the timelock itself (via self-administered governance proposals) to set the delay to any value, including 0.

The updateDelay function requires msg.sender == address(this), meaning it must be called through the timelock's schedule-and-execute flow. Since the timelockAdmin is both proposer and executor, they can propose a call to updateDelay(0), wait the current delay period, and execute it. After execution, the timelock's delay becomes 0, allowing all subsequent proposals to be executed immediately without any waiting period.

The MIN_DELAY and MAX_DELAY constants create a false sense of security by suggesting that delay bounds are always enforced. ToriTimelock does not override updateDelay(...) to enforce these bounds, so they are only checked once at deployment and never again.

```
contract ToriTimelock is TimelockController {
    uint256 public constant MIN_DELAY = 1 days;
    uint256 public constant MAX_DELAY = 30 days;

    constructor(
        uint256 minDelay,
        address timelockAdmin,
        address canceller
    ) TimelockController(
        minDelay,
        _toArray(timelockAdmin),
        _toArray(timelockAdmin),
        address(this)
    ) {
        require(minDelay >= MIN_DELAY, "Delay too short");
        require(minDelay <= MAX_DELAY, "Delay too long");
    }
}
```

Recommendation(s): Consider overriding the updateDelay(...) function in ToriTimelock to enforce the MIN_DELAY and MAX_DELAY bounds. The override should validate that any new delay value falls within the defined range before calling the parent implementation. This ensures the delay bounds remain enforced throughout the contract's lifetime, not just at deployment.

Status: Fixed.

Update from the client: Fixed in commit [aebac49](#).

6.0.4 [Low] TrUsdSilo lacks withdrawal for other assets

File(s): [contracts/core/TrUsdSilo.sol](#)

Description: TrUsdSilo stores TrUSD assets during cooldown period and staking vault to withdraw it. But the downside is that if any other asset is sent to the silo, it is lost as there is no way to withdraw the assets.

Recommendation(s): Consider making the withdrawal function flexible, so that other assets can be withdrawn.

Status: Fixed

Update from the client: Fixed in commits [548969a](#) and [4ac4e51](#).

6.0.5 [Low] removeAllowedToken cannot remove disabled tokens, creating permanently orphaned storage entries

File(s): `contracts/core/ToriMinting.sol`

Description: ToriMinting's `removeAllowedToken()` requires `tokenConfig[_token].isActive == true` before deleting the token config, but `StakingRewardsDistributor` sets `isActive` to false. This means the natural admin workflow of disable-then-remove always reverts, because the removal guard rejects any token that has already been disabled.

The workaround is to re-enable the token via `enableAllowedToken()` before removing it, but this requires the `TIMELOCK_ADMIN_ROLE` — a different and more privileged role than the `DEFAULT_ADMIN_ROLE` used for both `disableAllowedToken` and `removeAllowedToken`. The `DEFAULT_ADMIN` who disabled the token cannot independently complete the removal without coordinating with the timelock admin.

A more severe edge case creates a permanently irrecoverable state. If the `DEFAULT_ADMIN` calls `setMaxMintPerBlock` with a value of 0 for an active token (there is no zero-value guard in that function), then calls `disableAllowedToken`, the token enters a state where `isActive` is false and `maxMintPerBlock` is 0. In this state, `removeAllowedToken` reverts because `isActive` is false, and `enableAllowedToken` reverts because `maxMintPerBlock` is 0. No function can transition the token out of this state. The only escape hatch is for the `TIMELOCK_ADMIN` to call `addAllowedToken` to overwrite the orphaned entry with a fresh active configuration, but this re-adds the token rather than removing it.

The `isActive` guard in `removeAllowedToken` conflates "token exists" with "token is currently active." The function's intent is to delete a known token config, but the guard prevents deletion of any token that is not in the active state, which directly contradicts the removal workflow.

```
function removeAllowedToken(address _token) external onlyRole(DEFAULT_ADMIN_ROLE) {
    if (_token == address(0)) revert ZeroAddress();
    if (!tokenConfig[_token].isActive) revert TokenNotAllowed(_token);
    delete tokenConfig[_token];
    emit TokenRemoved(_token);
}
```

```
function disableAllowedToken(address _token) external onlyRole(DEFAULT_ADMIN_ROLE) {
    if (_token == address(0)) revert ZeroAddress();
    if (!tokenConfig[_token].isActive) revert TokenNotAllowed(_token);
    tokenConfig[_token].isActive = false;
}
```

Recommendation(s): Consider changing the existence check in `removeAllowedToken (...)` so that it verifies the token config entry exists in storage (for example, by checking that at least one struct field has a non-default value) rather than requiring `isActive` to be true. This allows removal of tokens regardless of whether they are currently active or disabled, which aligns with the function's intended purpose of cleaning up token configurations.

Status: Fixed

Update from the client: Fixed in commit [32ad8f2](#).

6.0.6 [Info] CannotRescueRewardToken error declared but unused; rescueTokens allows owner to extract TrUSD

File(s): [contracts/core/StakingRewardsDistributor.sol](#)

Description: StakingRewardsDistributor declares error CannotRescueRewardToken() at line 55 but never uses it. The rescueTokens(...) function performs no check against the TRUSD_TOKEN immutable, allowing the owner to call rescueTokens(address(TRUSD_TOKEN), to, amount) and extract the contract's operational TrUSD balance.

The analogous function in StakedTrUSD guards the underlying asset with `if (address(token) == asset()) revert InvalidToken()`, preventing the admin from draining the vault's backing token. The declared but unused error in StakingRewardsDistributor indicates the developers intended to implement the same safeguard but did not wire it up.

The contract is designed to hold TrUSD between funding events and distribution to the staking vault via `transferInRewards(...)`. Without the guard, the owner (multisig) can bypass the normal `transferInRewards(...)` flow — which has operator restrictions, amount checks, and event logging — and extract TrUSD directly through the rescue mechanism. Both the solidity-auditor and sc-auditor AI tools independently identified this issue.

```
error CannotRescueRewardToken();
```

```
function rescueTokens(address _token, address _to, uint256 _amount) external nonReentrant onlyOwner {
    if (_token == address(0)) revert InvalidZeroAddress();
    if (_to == address(0)) revert InvalidZeroAddress();
    if (_amount == 0) revert InvalidAmount();

    // Contract can only receive ETH via selfdestruct, allow rescuing it
    if (_token == _ETH_ADDRESS) {
        (bool success,) = _to.call{ value: _amount }("");
        if (!success) revert TransferFailed();
    } else {
        IERC20(_token).safeTransfer(_to, _amount);
    }
    emit TokensRescued(_token, _to, _amount);
}
```

Recommendation(s): Consider adding a guard to `rescueTokens(...)` that reverts with `CannotRescueRewardToken` when the token parameter equals `address(TRUSD_TOKEN)`. This prevents accidental or intentional extraction of the contract's operational TrUSD through the rescue mechanism, consistent with the pattern already implemented in `StakedTrUSD.rescueTokens(...)`.

Status: Fixed

Update from the client: Fixed in commit [a7e9140](#).

6.0.7 [Info] Cross-chain blacklist systems are independent with no synchronization mechanism

File(s): [contracts/omnichain/StakedTrUSDOFT.sol](#), [contracts/omnichain/StakedTrUSDAdapter.sol](#)

Description: The protocol uses two independent blacklist systems across its cross-chain deployment with no synchronization mechanism:

1. StakedTrUSD (home chain): Uses `AccessControl` with `FULL_RESTRICTED_STAKER_ROLE` ;
2. StakedTrUSDOFT (remote chains): Uses a local mapping(`address => bool`) managed by a separate `blackLister` address ;

Blacklisting a user on the home chain does not propagate to remote chains. A user restricted on the home chain can still hold, transfer, and receive StakedTrUSDOFT tokens on any remote chain where they have not been independently blacklisted. Similarly, a user blacklisted on one remote chain remains unrestricted on all other chains.

The operational burden of maintaining consistent blacklists across all deployed chains creates a window during which restricted users can move assets cross-chain before administrators can propagate the restriction.

```
mapping(address => bool) public blackList;
```

```
bytes32 private constant FULL_RESTRICTED_STAKER_ROLE = keccak256("FULL_RESTRICTED_STAKER_ROLE");
```

Recommendation(s): Consider documenting the operational requirement for administrators to maintain blacklists on all deployed chains simultaneously. Consider implementing a cross-chain blacklist synchronization message via `LayerZero` so that blacklisting on the home chain automatically propagates to remote chains.

Status: Acknowledged

Update from the client: This is a known design trade-off. Our off-chain services will maintain blacklists through a periodic synchronization process, ensuring consistency is enforced at the operational level.

6.0.8 [Info] Repeated cooldownAssets/cooldownShares calls reset the cooldown timer for all accumulated assets

File(s): [contracts/core/StakedTrUSD.sol](#)

Description: In StakedTrUSD, both `cooldownAssets()` and `unstake(...)` accumulate `underlyingAmount` via `+=` but overwrite `cooldownEnd(...)` with a fresh timestamp on every call. If a user calls `cooldownAssets(...)` a second time before the first cooldown has matured, the timer resets for all accumulated assets — including those from the first call that may have been close to maturity.

For example, with a 7-day cooldown: a user who calls `cooldownAssets(100)` at $T=0$ and then `cooldownAssets(50)` at $T=6d$ will have `cooldownEnd(...)` reset to $T+13d$ for all 150 assets. The first 100 assets that were 6 days into their 7-day cooldown are now locked for an additional 7 days.

This can cause users to accidentally extend their own cooldown period without realizing it. The workaround is to call `unstake()` for matured assets before starting a new cooldown.

This is a standard design pattern inherited from Ethena's `sUSDe` (StakedUSDe), chosen for storage simplicity over per-cooldown tracking.

```
$.cooldowns[msg.sender].cooldownEnd = uint104(block.timestamp) + $.cooldownDuration;
$.cooldowns[msg.sender].underlyingAmount += assets;
```

Recommendation(s): Document the cooldown timer reset behavior clearly in NatSpec and user-facing documentation. Consider adding a view function that lets users check their current cooldown state before initiating a new one, reducing the risk of accidental self-griefing.

Status: Acknowledged

Update from the client: This is an intentional design choice.

6.0.9 [Info] StakedTrUSD ERC-4626 max view functions return incorrect values under access restrictions

File(s): [contracts/core/StakedTrUSD.sol](#)

Description: StakedTrUSD does not override the ERC-4626 `maxWithdraw`, `maxRedeem`, `maxDeposit`, or `maxMint` view functions to account for its custom access restrictions. This violates several ERC-4626 MUST requirements.

When `cooldownDuration` is non-zero (the default configuration of 7 days), the `withdraw()` and `redeem()` functions revert via the modifier. However, `maxWithdraw(...)` and `maxRedeem(...)` still return non-zero values based on the owner's share balance because the inherited OpenZeppelin `ERC4626Upgradeable` implementation is used without modification. ERC-4626 mandates that `maxWithdraw(...)` and `maxRedeem(...)` MUST return the maximum amount that would not cause a revert. Since the standard withdrawal path reverts when cooldown is active, both functions should return 0. Users must use the `cooldownAssets(...)/cooldownShares(...)` flow instead, but integrating protocols and aggregators checking `maxWithdraw(...)/maxRedeem(...)` will receive incorrect values, attempt withdrawals, and have transactions revert.

Addresses with the `FULL_RESTRICTED_STAKER_ROLE` are blocked from depositing in the `_deposit` override, which reverts with `OperationNotAllowed`. However, `maxDeposit(...)` and `maxMint(...)` return `type(uint256).max` for these addresses. ERC-4626 requires these functions to return 0 when deposit or mint would revert for the given receiver.

```
function withdraw(uint256 assets, address receiver, address _owner)
    public
    virtual
    override
    ensureCooldownOff
    returns (uint256)
{
    return super.withdraw(assets, receiver, _owner);
}
```

```
function _deposit(address caller, address receiver, uint256 assets, uint256 shares)
    internal
    override
    nonReentrant
    notZero(assets)
    notZero(shares)
{
    if (hasRole(FULL_RESTRICTED_STAKER_ROLE, caller) || hasRole(FULL_RESTRICTED_STAKER_ROLE, receiver)) {
        revert OperationNotAllowed();
    }
    super._deposit(caller, receiver, assets, shares);
    _checkMinShares();
}
```

Recommendation(s): Consider overriding `maxWithdraw(...)` and `maxRedeem(...)` in StakedTrUSD to return 0 when `cooldownDuration` is non-zero, since the standard withdrawal path is disabled in that mode. Additionally, consider overriding `maxDeposit(...)` and `maxMint(...)`

to return 0 when the receiver has the FULL_RESTRICTED_STAKER_ROLE. This ensures ERC-4626 compliance and correct composability with external protocols that rely on these view functions to determine deposit and withdrawal limits.

Status: Acknowledged.

Update from the client: Acknowledged.

6.0.10 [Info] cancelCooldown() allows zero-share minting causing silent loss of user funds

File(s): [contracts/core/StakedTrUSD.sol](#)

Description: The cancelCooldown() function calls _mint(msg.sender, shares) directly without checking if shares == 0. When previewDeposit/assets returns 0 due to Floor rounding (which can occur with dust amounts or extreme exchange rates), the function succeeds without reverting, causing the user to permanently lose their cooldown assets while the vault is silently enriched.

```
function cancelCooldown() external nonReentrant {
    // ...
    // ...

    uint256 shares = previewDeposit/assets;

    // Transfer assets from Silo back to vault
    $.silo.withdraw(address(this), assets);

    // Mint shares
    _mint(msg.sender, shares);

    // Ensure MIN_SHARES invariant
    _checkMinShares();

    emit CooldownCancelled(msg.sender, assets, shares);
}
```

Recommendation(s): Consider checking shares minted is not zero.

Status: Fixed

Update from the client: Fixed in commit [3968ec5](#).

6.0.11 [Info] redistributeLockedAmount() violates MIN_SHARES invariant

File(s): [contracts/core/StakedTrUSD.sol](#)

Description: The redistributeLockedAmount() function burns shares from a blacklisted user and optionally mints to a recipient (or burns to address(0) to distribute value to remaining stakers). However, it lacks a _checkMinShares(...) call after the burn/mint operations.

```
function redistributeLockedAmount(address from, address to) external nonReentrant onlyRole(DEFAULT_ADMIN_ROLE) {
    if (hasRole(FULL_RESTRICTED_STAKER_ROLE, from) && !hasRole(FULL_RESTRICTED_STAKER_ROLE, to)) {
        uint256 amountToDistribute = balanceOf(from);

        _burn(from, amountToDistribute);

        if (to != address(0)) {
            _mint(to, amountToDistribute);
        }
        // When to == address(0): assets remain in vault, share value increases instantly for remaining stakers

        emit LockedAmountRedistributed(from, to, amountToDistribute);
    } else {
        revert OperationNotAllowed();
    }
}
```

This allows the vault to be left in an invalid state where $0 < \text{totalSupply}() < \text{MIN_SHARES}$ ($1e18$), which causes all subsequent deposits and withdrawals to revert due to the MIN_SHARES invariant check. This issue can only be triggered by admin redistributing small blacklisted amounts to address(0).

Recommendation(s): Consider adding a the minimum shares check after the burn and mint operations.

Status: Fixed

Update from the client: Fixed in commit [3968ec5](#).

6.0.12 [Info] reportLoss silently burns partial amount instead of reverting with declared InsufficientAssets error

File(s): `contracts/core/StakedTrUSD.sol`

Description: `IStakedTrUSD` declares an `InsufficientAssets()` error with the NatSpec comment 'Error emitted when trying to report loss greater than available assets', indicating that `reportLoss` should revert when the loss exceeds the vault's balance. However, the implementation in `StakedTrUSD.reportLoss()` silently handles this case by burning only the available balance instead of reverting.

When the computed `totalBurn` (loss amount plus cancelled unvested rewards) exceeds the vault's token balance, the function burns only what is available and emits `LossReported` with the available amount rather than the requested amount. The upstream `StakingRewardsDistributor.reportLoss(...)` then emits its own `LossReported` event with the originally requested amount, creating a discrepancy in the event trail between the requested and actual loss.

```
uint256 available = IERC20(asset()).balanceOf(address(this));

if (available >= totalBurn) {
    IERC20Burnable(asset()).burn(totalBurn);
    emit LossReported(amount);
} else {
    IERC20Burnable(asset()).burn(available);
    emit LossReported(available);
}
```

```
/// @notice Error emitted when trying to report loss greater than available assets
error InsufficientAssets();
```

Recommendation(s): Consider either using the declared `InsufficientAssets` error to revert when the loss exceeds the vault's available balance, consistent with the interface's documented intent, or remove the unused error from the interface if the silent partial-burn behavior is intentional. If keeping the partial-burn approach, consider returning the actual burned amount to callers like `StakingRewardsDistributor`.

Status: Fixed

Update from the client: Fixed in commit [51e15dd](#).

6.0.13 [Info] verifyNonce truncates uint256 nonce to uint128, reducing effective nonce space

File(s): `contracts/core/ToriMinting.sol`

Description: In `ToriMinting`, `verifyNonce`, the nonce parameter is declared as `uint256` in both the `Order` struct and the EIP-712 type hash, but the bitmap slot computation truncates it to `uint128` before shifting. Specifically, `uint128(nonce) >> 8` discards the upper 128 bits (bits 128-255) of the nonce entirely. This means two distinct `uint256` nonces that share the same lower 128 bits will map to the same bitmap slot and bit position, causing the second order to revert with `InvalidNonce` even though it has a unique nonce from the EIP-712 signature perspective.

The nonce is generated randomly off-chain rather than sequentially, which increases the practical relevance of the truncation. With random 256-bit nonces, the upper 128 bits are discarded silently, so two nonces that differ only in those bits will collide. The effective nonce space is reduced from 2^{256} to 2^{128} (2^{120} slots $\times$ 256 bit positions per slot). While 2^{128} is still an enormous space — birthday collisions require approximately 2^{64} nonces which is practically infeasible — the mismatch between the declared `uint256` type and the 128-bit effective space creates a correctness issue. Off-chain systems that generate nonces using the full 256-bit range (e.g., combining chain IDs, timestamps, or counters in upper bits) could inadvertently produce collisions in the lower 128 bits, leading to silent order rejections that are difficult to diagnose.

```
function verifyNonce(address sender, uint256 nonce) public view returns (uint256, uint256, uint256) {
    if (nonce == 0) revert InvalidNonce();
    // @audit-issue Nonce is truncated from uint256 to uint128, discarding upper 128 bits.
    uint256 invalidatorSlot = uint128(nonce) >> 8;
    uint256 invalidatorBit = 1 << uint8(nonce);
    uint256 invalidator = _orderBitmaps[sender][invalidatorSlot];
    if (invalidator & invalidatorBit != 0) revert InvalidNonce();

    return (invalidatorSlot, invalidator, invalidatorBit);
}
```

Recommendation(s): Consider using a sequential nonce scheme instead of random generation, starting from 1, the truncation to `uint128` is harmless for the first 2^{128} nonces per sender, which is practically inexhaustible. As a contract-level alternative, use the full `uint256` nonce for the bitmap slot computation instead of truncating to `uint128`.

Status: Acknowledged

Update from the client: Acknowledged

6.0.14 [Info] verifyStablesLimit divides by trusdAmount before verifyOrder's zero-amount guard, causing uninformative panic

File(s): [contracts/core/ToriMinting.sol](#)

Description: In ToriMinting verifyOrder(...), the call to verifyStablesLimit(...) precedes the zero-amount guard (if (order.collateral_amount == 0 || order.trusd_amount == 0) revert InvalidAmount()). Inside verifyStablesLimit(...) computes (difference * STABLES_RATIO_MULTIPLIER) / trusdAmount. When TrUSD_amount is 0, this causes a Solidity Panic(0x12) — division by zero — before the code reaches the intended InvalidAmount revert.

Both code paths revert and no state change occurs, so there is no fund risk. However, the Panic error is uninformative compared to the intended InvalidAmount custom error. Off-chain systems parsing revert reasons will see an opaque arithmetic panic rather than the descriptive application error.

The modifiers belowMaxMintPerBlock and belowGlobalMaxMintPerBlock do not block zero amounts — the check current + 0 > max passes for any non-exceeded block — so there is no earlier guard preventing this path.

```
if (!verifyStablesLimit(order.collateral_amount, order.trusd_amount, order.collateral_asset, order.order_type))
{
    revert InvalidStablePrice();
}

if (order.collateral_amount == 0 || order.trusd_amount == 0) revert InvalidAmount();
```

```
uint128 differenceInBps = (difference * STABLES_RATIO_MULTIPLIER) / trusdAmount;
```

Recommendation(s): Consider moving the zero-amount guard above the verifyStablesLimit(...) call so that orders with TrUSD_amount=0 or collateral_amount=0 are rejected with the descriptive InvalidAmount error before any arithmetic is performed on those values.

Status: Fixed

Update from the client: Fixed in commit [a1cec27](#).

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 Tori Finance's EVM smart contracts documentation

The Tori Finance's team provided an overview of the system components during the kick-off call with a detailed explanation of the intended functionalities and use cases of the system. 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

8.1 Tests Output

```
Ran 3 tests for test/Setup.t.sol:SetupTest
[PASS] test_EnvironmentSetup() (gas: 317)
[PASS] test_ForgeStdImport() (gas: 340)
[PASS] test_SolidityVersion() (gas: 273)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 9.86ms (1.76ms CPU time)

Ran 15 tests for test/OmnichainDirect.t.sol:OmnichainDirectTest
[PASS] test_StakedTrUSDAdapter_Inherits_TrUSDAdapter() (gas: 34290)
[PASS] test_StakedTrUSDOfT_SetBlackLister() (gas: 36064)
[PASS] test_StakedTrUSDOfT_SetBlackLister_RevertIf_NotOwner() (gas: 14635)
[PASS] test_StakedTrUSDOfT_UpdateBlackList() (gas: 37291)
[PASS] test_StakedTrUSDOfT_UpdateBlackList_Remove() (gas: 28241)
[PASS] test_StakedTrUSDOfT_UpdateBlackList_RevertIf_NotBlackLister() (gas: 15510)
[PASS] test_TrUSDAdapter_Pause() (gas: 31953)
[PASS] test_TrUSDAdapter_SetRateLimiter() (gas: 34379)
[PASS] test_TrUSDAdapter_SetRateLimits_ByOwner() (gas: 29623)
[PASS] test_TrUSDAdapter_SetRateLimits_ByRateLimiter() (gas: 56959)
[PASS] test_TrUSDAdapter_SetRateLimits_RevertIf_NotAuthorized() (gas: 40563)
[PASS] test_TrUSDAdapter_Unpause() (gas: 23288)
[PASS] test_TrUSDOfT_Pause() (gas: 31998)
[PASS] test_TrUSDOfT_SetRateLimiter() (gas: 34356)
[PASS] test_TrUSDOfT_Unpause() (gas: 23289)
Suite result: ok. 15 passed; 0 failed; 0 skipped; finished in 13.69ms (1.32ms CPU time)

Ran 2 tests for test/TestDoubleBurnClaim.t.sol:TestDoubleBurnClaim
[PASS] test_ClaimAnalysis_CommentScenario() (gas: 304072)
[PASS] test_WhatIfOnlyLossAmountBurned() (gas: 263840)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 14.24ms (1.79ms CPU time)

Ran 5 tests for test/TestAll4Scenarios.t.sol:TestAll4Scenarios
[PASS] test_AllScenarios() (gas: 9598)
[PASS] test_Scenario1_Loss20() (gas: 276211)
[PASS] test_Scenario2_Loss200() (gas: 266996)
[PASS] test_Scenario3_Loss1100() (gas: 264821)
[PASS] test_Scenario4_Loss1500() (gas: 281743)
Suite result: ok. 5 passed; 0 failed; 0 skipped; finished in 15.00ms (2.47ms CPU time)

Ran 2 tests for test/TestBurnAmount.t.sol:TestBurnAmount
[PASS] test_CompareApproaches() (gas: 297024)
[PASS] test_WhatIfOnlyBurn20() (gas: 281588)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 15.10ms (2.26ms CPU time)

Ran 18 tests for test/ToriTimelock.t.sol:ToriTimelockTest
[PASS] test_BatchScheduleAndExecute() (gas: 145857)
[PASS] test_Cancel_ByCanceller() (gas: 43681)
[PASS] test_Deployment() (gas: 28242)
[PASS] test_GetOperationStateString_Unknown() (gas: 89870)
[PASS] test_GetTimeRemaining() (gas: 54576)
[PASS] test_GetTimeRemaining_DoneOperation() (gas: 80733)
[PASS] test_GetTimeRemaining_UnsetOperation() (gas: 7879)
[PASS] test_Revert_DelayTooLong() (gas: 165278)
[PASS] test_Revert_DelayTooShort() (gas: 165307)
[PASS] test_Revert_ExecuteBeforeDelay() (gas: 64891)
[PASS] test_Revert_InvalidCanceller() (gas: 163263)
[PASS] test_Revert_InvalidTimelockAdmin() (gas: 163201)
[PASS] test_Revert_TimelockAdmin_CannotCancel() (gas: 54109)
[PASS] test_Revert_UnauthorizedCancel() (gas: 56094)
[PASS] test_Revert_UnauthorizedExecute() (gas: 57543)
[PASS] test_Revert_UnauthorizedSchedule() (gas: 16327)
[PASS] test_Scenario_AdminCompromised_CancelMaliciousOperation() (gas: 54424)
[PASS] test_ScheduleAndExecute_SetMinter() (gas: 100749)
Suite result: ok. 18 passed; 0 failed; 0 skipped; finished in 4.18ms (2.53ms CPU time)

Ran 51 tests for test/TrUSD_Omnichain.t.sol:TrUSD_OmnichainTest
[PASS] test_Adapter_AcceptOwnership_RevertIf_NotPendingOwner() (gas: 40071)
[PASS] test_Adapter_CancelPendingOwnership() (gas: 41127)
[PASS] test_Adapter_Constructor_EmptyConfigs() (gas: 2600397)
[PASS] test_Adapter_Constructor_MultipleDestinations() (gas: 2754106)
```

```
[PASS] test_Adapter_GetAmountCanBeSent() (gas: 14634)
[PASS] test_Adapter_Initialization() (gas: 15926)
[PASS] test_Adapter_OnlyPendingOwnerCanAccept() (gas: 43305)
[PASS] test_Adapter_OwnershipTransfer_TwoStepPattern() (gas: 36922)
[PASS] test_Adapter_Pause_BlocksOperations() (gas: 31981)
[PASS] test_Adapter_Pause_RevertIf_NotOwner() (gas: 13097)
[PASS] test_Adapter_Pause_Success() (gas: 38112)
[PASS] test_Adapter_RateLimit_AfterLimitChange() (gas: 31918)
[PASS] test_Adapter_RateLimit_MultipleDestinations() (gas: 73992)
[PASS] test_Adapter_RateLimit_UpdatesOnTransfer() (gas: 16509)
[PASS] test_Adapter_RateLimit_WindowTransition() (gas: 27321)
[PASS] test_Adapter_RateLimit_ZeroWindowReverts() (gas: 14252)
[PASS] test_Adapter_RateLimits_View() (gas: 14509)
[PASS] test_Adapter_SetRateLimiter_RevertIf_NotOwner() (gas: 13340)
[PASS] test_Adapter_SetRateLimiter_Success() (gas: 37658)
[PASS] test_Adapter_SetRateLimits_ByOwner() (gas: 29407)
[PASS] test_Adapter_SetRateLimits_ByRateLimiter() (gas: 50624)
[PASS] test_Adapter_SetRateLimits_RevertIf_NotAuthorized() (gas: 16128)
[PASS] test_Adapter_TransferOwnership() (gas: 31658)
[PASS] test_Adapter_UnpauseAllowsOperations() (gas: 31888)
[PASS] test_Adapter_Unpause_Success() (gas: 28914)
[PASS] test_OFT_AcceptOwnership_RevertIf_NotPendingOwner() (gas: 40114)
[PASS] test_OFT_Constructor_EmptyConfigs() (gas: 2929583)
[PASS] test_OFT_Constructor_MultipleDestinations() (gas: 3083576)
[PASS] test_OFT_Constructor_VerifyDecimals() (gas: 5705)
[PASS] test_OFT_GetAmountCanBeSent() (gas: 14712)
[PASS] test_OFT_Initialization() (gas: 28889)
[PASS] test_OFT_OnlyPendingOwnerCanAccept() (gas: 43257)
[PASS] test_OFT_OwnershipTransfer_TwoStepPattern() (gas: 36884)
[PASS] test_OFT_Pause_BlocksOperations() (gas: 32043)
[PASS] test_OFT_Pause_RevertIf_NotOwner() (gas: 13118)
[PASS] test_OFT_Pause_Success() (gas: 38196)
[PASS] test_OFT_PendingOwner() (gas: 34911)
[PASS] test_OFT_RateLimit_AfterLimitChange() (gas: 63993)
[PASS] test_OFT_RateLimit_MultipleDestinations() (gas: 74051)
[PASS] test_OFT_RateLimit_UpdatesOnTransfer() (gas: 16565)
[PASS] test_OFT_RateLimit_WindowTransition() (gas: 67124)
[PASS] test_OFT_RateLimit_ZeroWindowReverts() (gas: 14339)
[PASS] test_OFT_RateLimits_View() (gas: 14521)
[PASS] test_OFT_SetRateLimiter_RevertIf_NotOwner() (gas: 13285)
[PASS] test_OFT_SetRateLimiter_Success() (gas: 37695)
[PASS] test_OFT_SetRateLimits_ByOwner() (gas: 63654)
[PASS] test_OFT_SetRateLimits_ByRateLimiter() (gas: 84825)
[PASS] test_OFT_SetRateLimits_RevertIf_NotAuthorized() (gas: 16084)
[PASS] test_OFT_TransferOwnership() (gas: 30736)
[PASS] test_OFT_UnpauseAllowsOperations() (gas: 31212)
[PASS] test_OFT_Unpause_Success() (gas: 28978)
Suite result: ok. 51 passed; 0 failed; 0 skipped; finished in 3.90ms (2.77ms CPU time)
```

```
Ran 2 tests for test/TestScenario4.t.sol:TestScenario4
[PASS] test_Scenario4_Loss1500_DetailedAnalysis() (gas: 310994)
[PASS] test_Scenario4_StepByStep() (gas: 279874)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 39.61ms (27.25ms CPU time)
```

```
Ran 3 tests for test/TestReportLossScenario.t.sol:TestReportLossScenario
[PASS] test_ReportLoss_1000Assets_100Unvested_20Loss() (gas: 284184)
[PASS] test_ReportLoss_3Quarters_Vested() (gas: 274856)
[PASS] test_ReportLoss_ExactMath_100Unvested_20Loss() (gas: 273786)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 46.26ms (2.68ms CPU time)
```

```
Ran 27 tests for test/StakingRewardsDistributor.t.sol:StakingRewardsDistributorTest
[PASS] test_Constructor_RevertIf_InvalidStakingVault() (gas: 97053)
[PASS] test_Constructor_RevertIf_NotAContract() (gas: 94506)
[PASS] test_Initialize() (gas: 21252)
[PASS] test_RenounceOwnership_Reverts() (gas: 13172)
[PASS] test_ReportLoss_RevertIf_NotOperator() (gas: 18209)
[PASS] test_ReportLoss_RevertIf_ZeroAmount() (gas: 18197)
[PASS] test_ReportLoss_Success() (gas: 256574)
[PASS] test_RescueTokens() (gas: 79691)
[PASS] test_RescueTokens_ETH() (gas: 52859)
[PASS] test_RescueTokens_EmitsEvent() (gas: 79788)
[PASS] test_RescueTokens_RevertIf_NotOwner() (gas: 71125)
[PASS] test_RescueTokens_RevertIf_ZeroAmount() (gas: 22780)
[PASS] test_RescueTokens_RevertIf_ZeroTo() (gas: 20605)
[PASS] test_RescueTokens_RevertIf_ZeroToken() (gas: 20630)
```

```
[PASS] test_RescueTokens_TrUSD() (gas: 90681)
[PASS] test_RevokeStakingVaultApproval() (gas: 32463)
[PASS] test_RevokeStakingVaultApproval_RevertIf_NotOwner() (gas: 13503)
[PASS] test_SetOperator() (gas: 26116)
[PASS] test_SetOperator_RevertIf_Unchanged() (gas: 17467)
[PASS] test_SetOperator_ZeroAddress() (gas: 15766)
[PASS] test_TransferInRewards_FailsAfterApprovalRevoked() (gas: 261105)
[PASS] test_TransferInRewards_RevertIf_InsufficientFunds() (gas: 28625)
[PASS] test_TransferInRewards_RevertIf_NotOperator() (gas: 79829)
[PASS] test_TransferInRewards_Success() (gas: 239296)
[PASS] test_TransferOwnership() (gas: 33809)
[PASS] test_UpdateStakingVaultApproval() (gas: 37803)
[PASS] test_UpdateStakingVaultApproval_RevertIf_NotOwner() (gas: 13609)
Suite result: ok. 27 passed; 0 failed; 0 skipped; finished in 35.37ms (55.07ms CPU time)
```

```
Ran 143 tests for test/ToriMinting.t.sol:ToriMintingTest
[PASS] test_AddAllowedToken_RevertIf_MaxMintPerBlockZero() (gas: 478876)
[PASS] test_AddAllowedToken_RevertIf_MaxRedeemPerBlockZero() (gas: 478949)
[PASS] test_AddAllowedToken_RevertIf_NotTimelockAdmin() (gas: 436860)
[PASS] test_AddAllowedToken_RevertIf_ZeroAddress() (gas: 63235)
[PASS] test_AddAllowedToken_Success() (gas: 530950)
[PASS] test_AddCustodianAddress_RevertIf_AlreadyExists() (gas: 15484)
[PASS] test_AddCustodianAddress_RevertIf_ContractItself() (gas: 11106)
[PASS] test_AddCustodianAddress_RevertIf_NotAdmin() (gas: 14943)
[PASS] test_AddCustodianAddress_RevertIf_ZeroAddress() (gas: 11141)
[PASS] test_AddCustodianAddress_Success() (gas: 72359)
[PASS] test_AddToWhitelistBatch_RevertIf_NotAdmin() (gas: 17077)
[PASS] test_AddToWhitelistBatch_SkipsZeroAddress() (gas: 69069)
[PASS] test_AddToWhitelistBatch_Success() (gas: 92591)
[PASS] test_AddToWhitelist_RevertIf_NotAdmin() (gas: 14898)
[PASS] test_AddToWhitelist_RevertIf_ZeroAddress() (gas: 11070)
[PASS] test_AddToWhitelist_Success() (gas: 38278)
[PASS] test_ApprovedRecipient_Coverage() (gas: 317357)
[PASS] test_BelowMaxMintPerBlock_RevertIf_InactiveAsset() (gas: 76783)
[PASS] test_BelowMaxRedeemPerBlock_RevertIf_InactiveAsset() (gas: 288768)
[PASS] test_DisableAllowedToken_RevertIf_AlreadyDisabled() (gas: 20558)
[PASS] test_DisableAllowedToken_RevertIf_NotAdmin() (gas: 15611)
[PASS] test_DisableAllowedToken_RevertIf_ZeroAddress() (gas: 11071)
[PASS] test_DisableAllowedToken_Success() (gas: 22236)
[PASS] test_DisableAndEnable_MintingBlocked() (gas: 20930)
[PASS] test_DisableMintRedeem_RevertIf_NotGatekeeper() (gas: 13396)
[PASS] test_DisableMintRedeem_Success() (gas: 15402)
[PASS] test_EnableAllowedToken_RevertIf_NotTimelockAdmin() (gas: 24403)
[PASS] test_EnableAllowedToken_RevertIf_TokenNotExists() (gas: 64732)
[PASS] test_EnableAllowedToken_RevertIf_ZeroAddress() (gas: 61017)
[PASS] test_EnableAllowedToken_Success() (gas: 75761)
[PASS] test_EncodeOrder_ReturnsConsistentValue() (gas: 18436)
[PASS] test_GetDomainSeparator_DifferentChain() (gas: 10168)
[PASS] test_GetDomainSeparator_ReturnsConsistentValue() (gas: 6744)
[PASS] test_GrantRole_RevertIf_TimelockAdminRole() (gas: 11151)
[PASS] test_HashOrder_DifferentOrdersDifferentHashes() (gas: 17977)
[PASS] test_HashOrder_ReturnsConsistentValue() (gas: 17476)
[PASS] test_Initialization() (gas: 17471)
[PASS] test_Initialization_RevertIf_EmptyTokens() (gas: 136931)
[PASS] test_Initialization_RevertIf_MismatchedArrays() (gas: 141467)
[PASS] test_Initialization_RevertIf_TokenAddressIsTrUSD() (gas: 161127)
[PASS] test_Initialization_RevertIf_ZeroMaxMintPerBlock() (gas: 165791)
[PASS] test_Initialization_RevertIf_ZeroMaxRedeemPerBlock() (gas: 165854)
[PASS] test_Initialization_RevertIf_ZeroTrUSDAddress() (gas: 138476)
[PASS] test_Initialization_Roles() (gas: 26959)
[PASS] test_Initialization_TokenConfig() (gas: 13007)
[PASS] test_Initialization_Whitelist() (gas: 15556)
[PASS] test_InitializeTimelockAdmin_RevertIf_AlreadyInitialized() (gas: 14777)
[PASS] test_InitializeTimelockAdmin_RevertIf_NotAdmin() (gas: 4017272)
[PASS] test_InitializeTimelockAdmin_RevertIf_ZeroAddress() (gas: 4011406)
[PASS] test_Mint_RevertIf_ExceedsGlobalLimit() (gas: 84826)
[PASS] test_Mint_RevertIf_ExceedsPerBlockLimit() (gas: 81126)
[PASS] test_Mint_RevertIf_ExpiredSignature() (gas: 75712)
[PASS] test_Mint_RevertIf_InvalidNonce() (gas: 288658)
[PASS] test_Mint_RevertIf_InvalidRoute() (gas: 106886)
[PASS] test_Mint_RevertIf_InvalidSignature() (gas: 86511)
[PASS] test_Mint_RevertIf_NotMinter() (gas: 34793)
[PASS] test_Mint_RevertIf_NotWhitelisted() (gas: 49204)
[PASS] test_Mint_RevertIf_Paused() (gas: 58976)
[PASS] test_Mint_RevertIf_RecipientNotApproved() (gas: 91116)
```

```
[PASS] test_Mint_RevertIf_WrongOrderType() (gas: 77359)
[PASS] test_Mint_RevertIf_ZeroCollateralAmount() (gas: 102965)
[PASS] test_Mint_RevertIf_ZeroTrUSDAmount() (gas: 100534)
[PASS] test_Mint_Success() (gas: 242824)
[PASS] test_Mint_ToDifferentRecipient_WithApproval() (gas: 338759)
[PASS] test_Mint_WithEIP1271Signature() (gas: 461917)
[PASS] test_Mint_WithEIP1271_RevertIf_InvalidSignature() (gas: 332127)
[PASS] test_Pause_RevertIf_NotAdmin() (gas: 13411)
[PASS] test_Pause_Success() (gas: 32339)
[PASS] test_Redeem_RevertIf_ExceedsGlobalLimit() (gas: 320871)
[PASS] test_Redeem_RevertIf_ExceedsPerBlockLimit() (gas: 321206)
[PASS] test_Redeem_RevertIf_NotRedeemer() (gas: 31522)
[PASS] test_Redeem_RevertIf_NotWhitelisted() (gas: 292675)
[PASS] test_Redeem_RevertIf_Paused() (gas: 55771)
[PASS] test_Redeem_RevertIf_WrongOrderType() (gas: 45883)
[PASS] test_Redeem_Success() (gas: 291482)
[PASS] test_RemoveAllowedToken_RevertIf_NotAdmin() (gas: 15598)
[PASS] test_RemoveAllowedToken_RevertIf_TokenNotActive() (gas: 17152)
[PASS] test_RemoveAllowedToken_RevertIf_ZeroAddress() (gas: 11029)
[PASS] test_RemoveAllowedToken_Success() (gas: 17575)
[PASS] test_RemoveCollateralManagerRole_RevertIf_NotGatekeeper() (gas: 15716)
[PASS] test_RemoveCollateralManagerRole_Success() (gas: 21016)
[PASS] test_RemoveCustodianAddress_RevertIf_NotAdmin() (gas: 15665)
[PASS] test_RemoveCustodianAddress_RevertIf_NotExists() (gas: 14720)
[PASS] test_RemoveCustodianAddress_Success() (gas: 34928)
[PASS] test_RemoveFromWhitelist_RevertIf_NotAdmin() (gas: 15579)
[PASS] test_RemoveFromWhitelist_Success() (gas: 17123)
[PASS] test_RemoveMinterRole_RevertIf_NotGatekeeper() (gas: 15651)
[PASS] test_RemoveMinterRole_Success() (gas: 21013)
[PASS] test_RemoveRedeemerRole_RevertIf_NotGatekeeper() (gas: 15671)
[PASS] test_RemoveRedeemerRole_Success() (gas: 21015)
[PASS] test_RevokeRole_RevertIf_TimelockAdminRole() (gas: 11843)
[PASS] test_RevokeRole_Success_NonTimelockRole() (gas: 32836)
[PASS] test_SetApprovedRecipient_Remove() (gas: 66602)
[PASS] test_SetApprovedRecipient_RevertIf_AlreadyApproved() (gas: 83173)
[PASS] test_SetApprovedRecipient_RevertIf_NotApproved() (gas: 15189)
[PASS] test_SetApprovedRecipient_RevertIf_ZeroAddress() (gas: 11394)
[PASS] test_SetApprovedRecipient_Success() (gas: 82462)
[PASS] test_SetGlobalLimits_RevertIf_NotAdmin() (gas: 13653)
[PASS] test_SetGlobalLimits_Success() (gas: 16699)
[PASS] test_SetMaxMintPerBlock_RevertIf_NotAdmin() (gas: 15741)
[PASS] test_SetMaxMintPerBlock_RevertIf_TokenNotAllowed() (gas: 15157)
[PASS] test_SetMaxMintPerBlock_Success() (gas: 21417)
[PASS] test_SetMaxRedeemPerBlock_RevertIf_NotAdmin() (gas: 15715)
[PASS] test_SetMaxRedeemPerBlock_RevertIf_TokenNotAllowed() (gas: 15164)
[PASS] test_SetMaxRedeemPerBlock_Success() (gas: 21418)
[PASS] test_SetStablesDeltaLimit_RevertIf_NotAdmin() (gas: 13510)
[PASS] test_SetStablesDeltaLimit_Success() (gas: 31426)
[PASS] test_SetTimelockAdmin_RevertIf_ZeroAddress() (gas: 61025)
[PASS] test_SetTimelockAdmin_Success() (gas: 97587)
[PASS] test_TransferCollateral_NoRemainder() (gas: 275183)
[PASS] test_TransferCollateral_RemainingBalance() (gas: 109227)
[PASS] test_TransferCollateral_WithRemainder() (gas: 364733)
[PASS] test_TransferCollateral_WithRemainingBalance() (gas: 272874)
[PASS] test_TransferToCustody_RevertIf_InvalidCustodianAddress() (gas: 59640)
[PASS] test_TransferToCustody_RevertIf_InvalidRoute_EmptyAddresses() (gas: 54753)
[PASS] test_TransferToCustody_RevertIf_InvalidRoute_MismatchedLengths() (gas: 59609)
[PASS] test_TransferToCustody_RevertIf_InvalidRoute_RatioNotTenThousand() (gas: 67106)
[PASS] test_TransferToCustody_RevertIf_NotCollateralManager() (gas: 56795)
[PASS] test_TransferToCustody_RoundingRemainder() (gas: 195494)
[PASS] test_TransferToCustody_SingleCustodian() (gas: 68037)
[PASS] test_TransferToCustody_Success() (gas: 104069)
[PASS] test_TransferToCustody_ZeroAmountToTransfer() (gas: 169947)
[PASS] test_Unpause_RevertIf_NotAdmin() (gas: 40021)
[PASS] test_Unpause_Success() (gas: 23829)
[PASS] test_VerifyNonce_RevertIf_AlreadyUsed() (gas: 238214)
[PASS] test_VerifyNonce_RevertIf_ZeroNonce() (gas: 10988)
[PASS] test_VerifyNonce_Success() (gas: 10620)
[PASS] test_VerifyOrder_RevertIf_RecipientZeroAddress() (gas: 31333)
[PASS] test_VerifyOrder_RevertIf_TokenNotActive() (gas: 38613)
[PASS] test_VerifyOrder_TokenTypeStableCheck() (gas: 239965)
[PASS] test_VerifyRoute_ReturnsFalse_EmptyAddresses() (gas: 7038)
[PASS] test_VerifyRoute_ReturnsFalse_InvalidCustodian() (gas: 10546)
[PASS] test_VerifyRoute_ReturnsFalse_MismatchedLengths() (gas: 11743)
[PASS] test_VerifyRoute_ReturnsFalse_RatioNot10000() (gas: 14348)
```

```
[PASS] test_VerifyRoute_ReturnsFalse_ZeroAddress() (gas: 10505)
[PASS] test_VerifyRoute_ReturnsFalse_ZeroRatio() (gas: 13594)
[PASS] test_VerifyRoute_Success() (gas: 21327)
[PASS] test_VerifyStablesLimit_Mint_CollateralMoreThanTrUSD() (gas: 48578)
[PASS] test_VerifyStablesLimit_Mint_ExceedsLimit() (gas: 48623)
[PASS] test_VerifyStablesLimit_Mint_WithinLimit() (gas: 48665)
[PASS] test_VerifyStablesLimit_Redeem_ExceedsLimit() (gas: 48678)
[PASS] test_VerifyStablesLimit_Redeem_TrUSDMoreThanCollateral() (gas: 48536)
[PASS] test_VerifyStablesLimit_Redeem_WithinLimit() (gas: 48718)
Suite result: ok. 143 passed; 0 failed; 0 skipped; finished in 46.45ms (44.05ms CPU time)

Ran 33 tests for test/StakedTrUSDOmnichain.t.sol:StakedTrUSDOmnichainTest
[PASS] testFuzz_StakedOFT_RedistributeBlackListedFunds(uint128) (runs: 256, : 172737, ~: 172740)
[PASS] testFuzz_StakedOFT_UpdateBlackList(address,bool) (runs: 256, : 31027, ~: 40822)
[PASS] test_StakedAdapter_CreditToBlacklistedUser_RedirectsToOwner() (gas: 52816)
[PASS] test_StakedAdapter_CreditToNonBlacklistedUser() (gas: 7814)
[PASS] test_StakedAdapter_CreditWithFailedCall_RedirectsToOwner() (gas: 9911)
[PASS] test_StakedAdapter_GetAmountCanBeSent() (gas: 14633)
[PASS] test_StakedAdapter_Initialization() (gas: 15959)
[PASS] test_StakedAdapter_Pause_Success() (gas: 38093)
[PASS] test_StakedAdapter_RateLimits_View() (gas: 14575)
[PASS] test_StakedAdapter_TransferOwnership() (gas: 31713)
[PASS] test_StakedAdapter_Unpause_Success() (gas: 28198)
[PASS] test_StakedOFT_CreditToBlacklistedUser_RedirectsToOwner() (gas: 40543)
[PASS] test_StakedOFT_GetAmountCanBeSent() (gas: 14689)
[PASS] test_StakedOFT_Initialization() (gas: 26005)
[PASS] test_StakedOFT_MultipleBlacklistUpdates() (gas: 53437)
[PASS] test_StakedOFT_Pause_Success() (gas: 38159)
[PASS] test_StakedOFT_RedistributeBlackListedFunds_PartialAmount() (gas: 174910)
[PASS] test_StakedOFT_RedistributeBlackListedFunds_RevertIf_NotBlacklisted() (gas: 16447)
[PASS] test_StakedOFT_RedistributeBlackListedFunds_RevertIf_NotOwner() (gas: 42752)
[PASS] test_StakedOFT_RedistributeBlackListedFunds_Success() (gas: 175839)
[PASS] test_StakedOFT_SetBlackLister_RevertIf_NotOwner() (gas: 15339)
[PASS] test_StakedOFT_SetBlackLister_Success() (gas: 38607)
[PASS] test_StakedOFT_TransferOwnership() (gas: 31663)
[PASS] test_StakedOFT_Transfer_RevertIf_FromBlacklisted() (gas: 43460)
[PASS] test_StakedOFT_Transfer_RevertIf_ToBlacklisted() (gas: 45680)
[PASS] test_StakedOFT_Transfer_Success() (gas: 51896)
[PASS] test_StakedOFT_Unpause_Success() (gas: 28184)
[PASS] test_StakedOFT_UpdateBlackList_ByBlackLister() (gas: 66684)
[PASS] test_StakedOFT_UpdateBlackList_ByOwner() (gas: 41897)
[PASS] test_StakedOFT_UpdateBlackList_RemoveFromBlacklist() (gas: 33088)
[PASS] test_StakedOFT_UpdateBlackList_RevertIf_NotAuthorized() (gas: 17621)
[PASS] test_StakedOFT_UpdateBlackList_RevertIf_OwnerBlacklisted() (gas: 15283)
[PASS] test_StakedOFT_UpdateBlackList_RevertIf_OwnerBlacklisted_ByBlacklist() (gas: 40146)
Suite result: ok. 33 passed; 0 failed; 0 skipped; finished in 73.63ms (103.30ms CPU time)

Ran 18 tests for test/scenarios/ToriMintingScenarios.t.sol:ToriMintingScenarioTest
[PASS] test_Scenario_BasicMint_Success() (gas: 238954)
[PASS] test_Scenario_BasicRedeem_Success() (gas: 294417)
[PASS] test_Scenario_ExpiredSignature() (gas: 99481)
[PASS] test_Scenario_FullMintRedeemCycle() (gas: 376872)
[PASS] test_Scenario_GatekeeperDisablesMintRedeem() (gas: 80719)
[PASS] test_Scenario_GlobalRateLimit_Exceeded() (gas: 965292)
[PASS] test_Scenario_HighVolumeDay() (gas: 1544950)
[PASS] test_Scenario_InvalidSignature() (gas: 84156)
[PASS] test_Scenario_MintToDifferentRecipient() (gas: 314232)
[PASS] test_Scenario_MultiUserMintSameBlock() (gas: 451407)
[PASS] test_Scenario_NonSequentialNonces() (gas: 423149)
[PASS] test_Scenario_NonWhitelistedUser() (gas: 103681)
[PASS] test_Scenario_NonceReplayProtection() (gas: 284382)
[PASS] test_Scenario_PausedContractBlocksMinting() (gas: 84815)
[PASS] test_Scenario_PerTokenRateLimit_Exceeded() (gas: 276236)
[PASS] test_Scenario_RateLimitResetsNextBlock() (gas: 356506)
[PASS] test_Scenario_SequentialNonces() (gas: 442854)
[PASS] test_Scenario_SplitRouteCustodians() (gas: 271641)
Suite result: ok. 18 passed; 0 failed; 0 skipped; finished in 58.93ms (27.98ms CPU time)

Ran 47 tests for test/TrUSD.t.sol:TrUSDTest
[PASS] testFuzz_Approve(address,uint256) (runs: 256, : 42565, ~: 42954)
[PASS] testFuzz_Burn(uint256,uint256) (runs: 256, : 137507, ~: 137632)
[PASS] testFuzz_Mint(address,uint256) (runs: 256, : 130962, ~: 130962)
[PASS] testFuzz_Transfer(uint256) (runs: 256, : 141875, ~: 141875)
[PASS] test_AcceptOwnership_RevertIf_NotPendingOwner() (gas: 34527)
[PASS] test_Approve_IncreaseAllowance() (gas: 48694)
```

```
[PASS] test_Approve_Success() (gas: 46673)
[PASS] test_BurnFrom_RevertIf_InsufficientAllowance() (gas: 160428)
[PASS] test_BurnFrom_Success() (gas: 170570)
[PASS] test_Burn_EntireBalance() (gas: 104963)
[PASS] test_Burn_RevertIf_InsufficientBalance() (gas: 131648)
[PASS] test_Burn_Success() (gas: 138921)
[PASS] test_GrantRole_NonTimelockRole_Success() (gas: 49093)
[PASS] test_GrantRole_RevertIf_TimelockAdminRole() (gas: 17576)
[PASS] test_InitializeTimelockAdmin_RevertIf_CalledTwice() (gas: 22042)
[PASS] test_Initialize_CannotReinitialize() (gas: 17956)
[PASS] test_Initialize_RevertIf_ZeroAddress() (gas: 2391357)
[PASS] test_Initialize_Success() (gas: 36264)
[PASS] test_Mint_MultipleRecipients() (gas: 162419)
[PASS] test_Mint_OwnerCannotMint() (gas: 20299)
[PASS] test_Mint_RevertIf_MinterToZeroAddress() (gas: 81051)
[PASS] test_Mint_RevertIf_NoMinterSet() (gas: 18352)
[PASS] test_Mint_RevertIf_NotMinter() (gas: 83027)
[PASS] test_Mint_Success() (gas: 134508)
[PASS] test_Permit_RevertIf_ExpiredDeadline() (gas: 28955)
[PASS] test_Permit_RevertIf_InvalidSignature() (gas: 55606)
[PASS] test_Permit_Success() (gas: 198369)
[PASS] test_RenounceRole_RevertAlways() (gas: 17408)
[PASS] test_RevokeRole_NonTimelockRole_Success() (gas: 39420)
[PASS] test_RevokeRole_RevertIf_TimelockAdminRole() (gas: 18145)
[PASS] test_SetMinter_CanSetToZeroAddress() (gas: 120496)
[PASS] test_SetMinter_RevertIf_NotTimelockAdmin() (gas: 31889)
[PASS] test_SetMinter_Success() (gas: 80094)
[PASS] test_SetMinter_UpdateExistingMinter() (gas: 124698)
[PASS] test_SetTimelockAdmin_RevertIf_ZeroAddress() (gas: 65925)
[PASS] test_SetTimelockAdmin_Success() (gas: 103515)
[PASS] test_Timelock_CanCancelSetMinter() (gas: 51679)
[PASS] test_Timelock_ExecuteBeforeDelay_Reverts() (gas: 56854)
[PASS] test_TransferFrom_RevertIf_InsufficientAllowance() (gas: 160639)
[PASS] test_TransferFrom_Success() (gas: 193329)
[PASS] test_TransferOwnership_RevertIf_NotOwner() (gas: 20533)
[PASS] test_TransferOwnership_TwoStepProcess() (gas: 61818)
[PASS] test_Transfer_RevertIf_InsufficientBalance() (gas: 134511)
[PASS] test_Transfer_Success() (gas: 163704)
[PASS] test_Upgrade_RevertIf_NotTimelockAdmin() (gas: 2338622)
[PASS] test_Upgrade_Success() (gas: 2387081)
[PASS] test_Version() (gas: 14414)
Suite result: ok. 47 passed; 0 failed; 0 skipped; finished in 82.13ms (201.27ms CPU time)
```

```
Ran 13 tests for test/OmnichainCoverage.t.sol:OmnichainCoverageTest
[PASS] test_StakedOFT_RedistributeBlackListedFunds() (gas: 245349)
[PASS] test_StakedOFT_RedistributeBlackListedFunds_FullBalance() (gas: 245368)
[PASS] test_StakedOFT_Redistribute_RevertIf_NotBlacklisted() (gas: 181489)
[PASS] test_StakedOFT_Redistribute_RevertIf_NotOwner() (gas: 201964)
[PASS] test_StakedOFT_Redistribute_TemporaryUnblacklist() (gas: 241840)
[PASS] test_StakedOFT_SetBlackLister() (gas: 19061)
[PASS] test_StakedOFT_Transfer_BlocksBlacklisted() (gas: 203754)
[PASS] test_StakedOFT_Transfer_BlocksBlacklistedRecipient() (gas: 205095)
[PASS] test_StakedOFT_UpdateBlackList_ByBlackLister() (gas: 42196)
[PASS] test_StakedOFT_UpdateBlackList_ByOwner() (gas: 37353)
[PASS] test_StakedOFT_UpdateBlackList_RemoveFromBlacklist() (gas: 29448)
[PASS] test_StakedOFT_UpdateBlackList_RevertIf_Owner() (gas: 15310)
[PASS] test_StakedOFT_UpdateBlackList_RevertIf_Unauthorized() (gas: 15563)
Suite result: ok. 13 passed; 0 failed; 0 skipped; finished in 82.13ms (5.41ms CPU time)
```

```
Ran 24 tests for test/scenarios/StakedTrUSDScenarios.t.sol:StakedTrUSDScenarioTest
[PASS] test_Scenario_BasicStake() (gas: 185769)
[PASS] test_Scenario_BlacklistedCannotDeposit() (gas: 198951)
[PASS] test_Scenario_BlacklistedCannotTransfer() (gas: 218557)
[PASS] test_Scenario_CannotAddRewardsDuringVesting() (gas: 227462)
[PASS] test_Scenario_CooldownDisabledByAdmin() (gas: 271399)
[PASS] test_Scenario_CooldownFlow() (gas: 271682)
[PASS] test_Scenario_CooldownWithShares() (gas: 273731)
[PASS] test_Scenario_DirectWithdrawCooldownOff() (gas: 206207)
[PASS] test_Scenario_ERC20Permit() (gas: 260428)
[PASS] test_Scenario_FullLifecycleSimulation() (gas: 472608)
[PASS] test_Scenario_LastStakerGetsAllRewards() (gas: 344996)
[PASS] test_Scenario_MultipleCooldowns() (gas: 300209)
[PASS] test_Scenario_MultipleUsersStaking() (gas: 302929)
[PASS] test_Scenario_PartialCooldown() (gas: 268379)
[PASS] test_Scenario_ProtocolOwnedLiquidityProtection() (gas: 369256)
```

```
[PASS] test_Scenario_RateNeverBelowOne() (gas: 322276)
[PASS] test_Scenario_RedistributeBlacklistedFunds() (gas: 243555)
[PASS] test_Scenario_RedistributeToVesting() (gas: 222614)
[PASS] test_Scenario_RewardsDuringVesting() (gas: 206045)
[PASS] test_Scenario_ShareTransferBetweenUsers() (gas: 289164)
[PASS] test_Scenario_StakingWithRewards() (gas: 262258)
[PASS] test_Scenario_UserExitsDuringVesting() (gas: 337596)
[PASS] test_Scenario_UserJoinsAfterRewards() (gas: 319685)
[PASS] test_Scenario-WithdrawBlockedWhenCooldownOn() (gas: 192543)
Suite result: ok. 24 passed; 0 failed; 0 skipped; finished in 82.19ms (18.19ms CPU time)

Ran 63 tests for test/PendleYTLock.t.sol:PendleYTLockTest
[PASS] testFuzz_Deposit(uint256) (runs: 256, : 171146, ~: 171322)
[PASS] test_AddToBlacklist() (gas: 49164)
[PASS] test_Blacklist_BlocksDeposit() (gas: 93859)
[PASS] test_Blacklist_BlocksDepositAsReceiver() (gas: 98285)
[PASS] test_Blacklist_BlocksTransfer() (gas: 203463)
[PASS] test_Blacklist_BlocksTransferTo() (gas: 205823)
[PASS] test_Blacklist_BlocksWithdraw() (gas: 213480)
[PASS] test_Blacklist_CantBlacklistOwner() (gas: 20891)
[PASS] test_Blacklist_OnlyManager() (gas: 20528)
[PASS] test_Decimals() (gas: 10610)
[PASS] test_Deposit_RevertIf_EmergencyPaused() (gas: 82350)
[PASS] test_Deposit_RevertIf_ResultsInTotalSupplyBelowMinShares() (gas: 175118)
[PASS] test_Deposit_SendsYTToHolder() (gas: 178577)
[PASS] test_EmergencyPause_BlocksDeposits() (gas: 82374)
[PASS] test_EmergencyPause_BlocksWithdrawals() (gas: 203911)
[PASS] test_EmergencyPause_OnlyAdmin() (gas: 18398)
[PASS] test_EmergencyUnpause_AllowsOperations() (gas: 178247)
[PASS] test_GetUserShares() (gas: 174530)
[PASS] test_GrantRole_RevertIf_TimelockAdminRole() (gas: 17572)
[PASS] test_Initialize() (gas: 37442)
[PASS] test_InitializeTimelockAdmin_RevertIf_AlreadyInitialized() (gas: 55322)
[PASS] test_InitializeTimelockAdmin_RevertIf_ZeroAddress() (gas: 18372)
[PASS] test_InitializeTimelockAdmin_Success() (gas: 3859280)
[PASS] test_Initialize_RevertIf_ZeroYTHolder() (gas: 3515384)
[PASS] test_MaxRedeem_WhenPaused() (gas: 14859)
[PASS] test_MaxRedeem_WhenUnpaused() (gas: 176505)
[PASS] test_MaxWithdraw_WhenPaused() (gas: 14912)
[PASS] test_MaxWithdraw_WhenUnpaused() (gas: 177681)
[PASS] test_NotOwner_Modifier_Coverage() (gas: 20814)
[PASS] test_NotZero_Modifier_Coverage() (gas: 28754)
[PASS] test_PauseUnpause-Withdrawals() (gas: 25639)
[PASS] test_PauseWithdrawals_EmitsEvent() (gas: 26419)
[PASS] test_PauseWithdrawals_RevertIf_NotAdmin() (gas: 18221)
[PASS] test_Redeem_RevertIf_Paused() (gas: 172208)
[PASS] test_Redeem_Success() (gas: 206319)
[PASS] test_Redeem_WithApproval_SpendAllowance() (gas: 215303)
[PASS] test_RemoveFromBlacklist() (gas: 39410)
[PASS] test_RenounceRole_Reverts() (gas: 16215)
[PASS] test_RescueTokens_RevertIf_AssetToken() (gas: 23079)
[PASS] test_RescueTokens_RevertIf_NotAdmin() (gas: 565953)
[PASS] test_RescueTokens_RevertIf_ZeroAddress() (gas: 568031)
[PASS] test_RescueTokens_Success() (gas: 595553)
[PASS] test_RevokeRole_RevertIf_TimelockAdminRole() (gas: 53363)
[PASS] test_RevokeRole_Success_NonTimelockRole() (gas: 40507)
[PASS] test_SetTimelockAdmin_RevertIf_ZeroAddress() (gas: 53541)
[PASS] test_SetTimelockAdmin_Success() (gas: 67487)
[PASS] test_SetYTHolder() (gas: 31698)
[PASS] test_SetYTHolder_RevertIf_NotAdmin() (gas: 19930)
[PASS] test_SetYTHolder_RevertIf_ZeroAddress() (gas: 18417)
[PASS] test_TotalAssets_ReadsFromHolder() (gas: 172859)
[PASS] test_Transfer_PreservesTimestamp() (gas: 259008)
[PASS] test_UnpauseWithdrawals_EmitsEvent() (gas: 20980)
[PASS] test_UnpauseWithdrawals_RevertIf_NotAdmin() (gas: 18306)
[PASS] test_Upgrade_PreservesUserBalances() (gas: 3645329)
[PASS] test_Upgrade_RevertIf_NotTimelockAdmin() (gas: 3490619)
[PASS] test_Upgrade_RevertIf_TimelockNotInitialized() (gas: 3446117)
[PASS] test_Upgrade_Success() (gas: 3495483)
[PASS] test_Version() (gas: 14434)
[PASS] test-Withdraw_DeletesUserLock_WhenAllSharesRedeemed() (gas: 204355)
[PASS] test-Withdraw_PreservesUserLock_WhenPartialWithdrawal() (gas: 207333)
[PASS] test-Withdraw_PullsFromHolder() (gas: 201564)
[PASS] test-Withdraw_RevertIf_Paused() (gas: 172250)
[PASS] test-Withdraw_WithApproval_SpendAllowanceBranch() (gas: 216042)
```

Suite result: ok. 63 passed; 0 failed; 0 skipped; finished in 82.23ms (44.84ms CPU time)

```
Ran 22 tests for test/TrUsdSilo.t.sol:TrUsdSiloTest
[PASS] testFuzz_Withdraw(uint128) (runs: 256, : 62043, ~: 62043)
[PASS] testFuzz_Withdraw_ToRandomRecipient(address,uint128) (runs: 256, : 60682, ~: 60682)
[PASS] test_Constructor_Success() (gas: 10841)
[PASS] test_EdgeCase_NoBalanceInSilo() (gas: 55181)
[PASS] test_EdgeCase_WithdrawToSiloItself() (gas: 29208)
[PASS] test_EdgeCase_WithdrawToZeroAddress_Reverts() (gas: 19366)
[PASS] test_Immutability() (gas: 10796)
[PASS] test_Integration_DepositAndWithdrawCycle() (gas: 57770)
[PASS] test_Integration_MultipleWithdrawals() (gas: 79812)
[PASS] test_OnlyStakingVault_Modifier() (gas: 61506)
[PASS] test_STAKING_VAULT_Getter() (gas: 7720)
[PASS] test_TRUSD_Getter() (gas: 7851)
[PASS] test_Version() (gas: 9312)
[PASS] test_ViewFunctions_ConsistentReturns() (gas: 8262)
[PASS] test_ViewFunctions_Immutability() (gas: 54260)
[PASS] test_Withdraw_FullBalance() (gas: 52833)
[PASS] test_Withdraw_RevertIf_InsufficientBalance() (gas: 27660)
[PASS] test_Withdraw_RevertIf_NotStakingVault() (gas: 11186)
[PASS] test_Withdraw_RevertIf_Owner() (gas: 13218)
[PASS] test_Withdraw_Success() (gas: 59412)
[PASS] test_Withdraw_ToMultipleRecipients() (gas: 85629)
[PASS] test_Withdraw_ZeroAmount() (gas: 36586)
Suite result: ok. 22 passed; 0 failed; 0 skipped; finished in 52.36ms (81.41ms CPU time)
```

```
Ran 30 tests for test/scenarios/StakedTrUSDEdgeCases.t.sol:StakedTrUSDEdgeCasesTest
[PASS] test_EdgeCase_CanWithdrawAll() (gas: 136596)
[PASS] test_EdgeCase_ConcurrentDepositsAndRewards() (gas: 311694)
[PASS] test_EdgeCase_DonationAttackPrevention() (gas: 208592)
[PASS] test_EdgeCase_ERC4626_MaxDeposit() (gas: 12739)
[PASS] test_EdgeCase_ERC4626_MaxMint() (gas: 12686)
[PASS] test_EdgeCase_ERC4626_MaxRedeem() (gas: 129059)
[PASS] test_EdgeCase_ERC4626_MaxWithdraw() (gas: 132443)
[PASS] test_EdgeCase_ERC4626_PreviewFunctions() (gas: 213512)
[PASS] test_EdgeCase_EmptyVault_FirstDepositor() (gas: 131805)
[PASS] test_EdgeCase_LargeDeposit() (gas: 144103)
[PASS] test_EdgeCase_LargeRewards() (gas: 215835)
[PASS] test_EdgeCase_MinSharesViolation_OnDeposit() (gas: 151167)
[PASS] test_EdgeCase_MinSharesViolation_OnWithdraw() (gas: 152257)
[PASS] test_EdgeCase_OrphanFunds_AllExitDuringVesting() (gas: 316156)
[PASS] test_EdgeCase_OrphanFunds_ExitDuringVesting() (gas: 390975)
[PASS] test_EdgeCase_ProtocolOwnedLiquidity_PreventsOrphanFunds() (gas: 441132)
[PASS] test_EdgeCase_RateNeverBelowOne() (gas: 283612)
[PASS] test_EdgeCase_RescueOrphanFunds_AfterAllExit() (gas: 320508)
[PASS] test_EdgeCase_RescueOrphanFunds_EmptyVault() (gas: 40285)
[PASS] test_EdgeCase_RescueOrphanFunds_RevertIf_HasStakers() (gas: 149807)
[PASS] test_EdgeCase_RescueOrphanFunds_ZeroRecipient() (gas: 65097)
[PASS] test_EdgeCase_Rounding_PreferVault() (gas: 253842)
[PASS] test_EdgeCase_SingleStaker_GetsAllRewards() (gas: 430524)
[PASS] test_EdgeCase_SmallDeposit_AboveMinShares() (gas: 127328)
[PASS] test_EdgeCase_VaultBecomesEmpty_ThenNewDeposit() (gas: 231515)
[PASS] test_EdgeCase_VaultRecovery_AfterOrphanFundsRescue() (gas: 330237)
[PASS] test_EdgeCase_VerySmallReward() (gas: 199346)
[PASS] test_EdgeCase_Vesting_CanAddAfterFullyVested() (gas: 218938)
[PASS] test_EdgeCase_Vesting_CannotAddDuringVesting() (gas: 224658)
[PASS] test_EdgeCase_Vesting_ExactBoundary() (gas: 202583)
Suite result: ok. 30 passed; 0 failed; 0 skipped; finished in 90.92ms (31.43ms CPU time)
```

```
Ran 33 tests for test/RateLimiter.t.sol:RateLimiterTest
[PASS] testFuzz_AmountCanBeSent_LinearDecay(uint256,uint256,uint256,uint256) (runs: 256, : 9944, ~: 10164)
[PASS] testFuzz_AmountCanBeSent_WindowElapsed(uint256,uint256) (runs: 256, : 10565, ~: 10570)
[PASS] testFuzz_SetRateLimits_ArbitraryValues(uint256,uint256) (runs: 256, : 63941, ~: 64252)
[PASS] test_AmountCanBeSent_AfterLimitReduction() (gas: 6646)
[PASS] test_AmountCanBeSent_DecayCalculation_LargeWindow() (gas: 9731)
[PASS] test_AmountCanBeSent_DecayExactlyEqualsInflight() (gas: 6544)
[PASS] test_AmountCanBeSent_DecayExceedsInflight() (gas: 6604)
[PASS] test_AmountCanBeSent_DecayLessThanInflight() (gas: 6641)
[PASS] test_AmountCanBeSent_HalfWindowElapsed() (gas: 6632)
[PASS] test_AmountCanBeSent_LimitEqualsInflight() (gas: 6496)
[PASS] test_AmountCanBeSent_LimitExceedsInflight() (gas: 6588)
[PASS] test_AmountCanBeSent_LimitLessThanInflight() (gas: 6398)
[PASS] test_AmountCanBeSent_PrecisionCheck_OneWei() (gas: 6516)
[PASS] test_AmountCanBeSent_QuarterWindowElapsed() (gas: 6676)
```

```
[PASS] test_AmountCanBeSent_VerySmallTimeElapsed() (gas: 9784)
[PASS] test_AmountCanBeSent_WindowElapsed_LargeInflight() (gas: 6321)
[PASS] test_AmountCanBeSent_WindowElapsed_MaxUint256() (gas: 6166)
[PASS] test_AmountCanBeSent_WindowElapsed_ZeroInflight() (gas: 6133)
[PASS] test_AmountCanBeSent_WindowExactlyElapsed() (gas: 6284)
[PASS] test_AmountCanBeSent_WindowMuchLongerElapsed() (gas: 6258)
[PASS] test_AmountCanBeSent_WindowOverElapsed() (gas: 6344)
[PASS] test_AmountCanBeSent_ZeroLimit() (gas: 6537)
[PASS] test_AmountCanBeSent_ZeroTimeElapsed() (gas: 6632)
[PASS] test_CheckAndUpdate_ExactLimit() (gas: 101307)
[PASS] test_CheckAndUpdate_RevertIf_ExceedsLimit() (gas: 62564)
[PASS] test_CheckAndUpdate_SequentialTransfers() (gas: 112584)
[PASS] test_CheckAndUpdate_Success() (gas: 101386)
[PASS] test_CheckAndUpdate_UnconfiguredDestination() (gas: 17288)
[PASS] test_GetAmountCanBeSent_PublicFunction() (gas: 104884)
[PASS] test_SetRateLimits_MultipleDestinations_Success() (gas: 163585)
[PASS] test_SetRateLimits_RevertIf_ZeroWindow() (gas: 9788)
[PASS] test_SetRateLimits_SingleDestination_Success() (gas: 65223)
[PASS] test_SetRateLimits_UpdateExisting_PreservesInflight() (gas: 108292)
Suite result: ok. 33 passed; 0 failed; 0 skipped; finished in 90.99ms (120.92ms CPU time)
```

```
Ran 139 tests for test/StakedTrUSD.t.sol:StakedTrUSDTest
[PASS] testFuzz_Deposit(uint128) (runs: 256, : 106506, ~: 106506)
[PASS] testFuzz_DepositAndWithdraw_CooldownOff(uint128) (runs: 256, : 131523, ~: 131525)
[PASS] testFuzz_ReportLoss(uint128) (runs: 256, : 54900, ~: 54900)
[PASS] testFuzz_Transfer(uint128) (runs: 256, : 137777, ~: 137777)
[PASS] test_AddToBlacklistBatch_RevertIf_ContainsOwner() (gas: 53600)
[PASS] test_AddToBlacklistBatch_RevertIf_EmptyArray() (gas: 18804)
[PASS] test_AddToBlacklistBatch_RevertIf_NotBlacklistManager() (gas: 21423)
[PASS] test_AddToBlacklistBatch_Success() (gas: 113736)
[PASS] test_AddToBlacklist_RevertIf_Owner() (gas: 20914)
[PASS] test_AddToBlacklist_RevertIf_TargetIsOwner() (gas: 20848)
[PASS] test_AddToBlacklist_Success() (gas: 49107)
[PASS] test_Blacklist_RevertIf_BlacklistOwner() (gas: 22879)
[PASS] test_Blacklist_RevertIf_NotBlacklistManager() (gas: 20627)
[PASS] test_CancelCooldown_NoShareLoss_SmallVault() (gas: 226445)
[PASS] test_CancelCooldown_PriceChange() (gas: 278498)
[PASS] test_CancelCooldown_RevertIf_Blacklisted() (gas: 193310)
[PASS] test_CancelCooldown_RevertIf_NoCooldown() (gas: 23249)
[PASS] test_CancelCooldown_Success() (gas: 205236)
[PASS] test_ConvertToAssets() (gas: 28616)
[PASS] test_ConvertToShares() (gas: 28626)
[PASS] test_CooldownAssets_RevertIf_CooldownDisabled() (gas: 119091)
[PASS] test_CooldownAssets_RevertIf_CooldownOff() (gas: 119157)
[PASS] test_CooldownAssets_RevertIf_ExcessiveAmount() (gas: 112125)
[PASS] test_CooldownAssets_RevertIf_SoftRestricted() (gas: 178960)
[PASS] test_CooldownAssets_Success() (gas: 165450)
[PASS] test_CooldownShares_RevertIf_ExcessiveAmount() (gas: 109285)
[PASS] test_CooldownShares_Success() (gas: 162940)
[PASS] test_Cooldowns_View() (gas: 138956)
[PASS] test_Deposit_MultipleUsers() (gas: 153380)
[PASS] test_Deposit_RevertIf_Blacklisted() (gas: 102421)
[PASS] test_Deposit_RevertIf_FullRestrictedCallerToInnocentReceiver() (gas: 104466)
[PASS] test_Deposit_RevertIf_InnocentCallerToFullRestrictedReceiver() (gas: 125305)
[PASS] test_Deposit_RevertIf_InnocentCallerToSoftRestrictedReceiver() (gas: 125238)
[PASS] test_Deposit_RevertIf_SoftRestrictedCallerToInnocentReceiver() (gas: 122931)
[PASS] test_Deposit_RevertIf_ZeroAmount() (gas: 68670)
[PASS] test_Deposit_Success() (gas: 92899)
[PASS] test_GrantRole_RevertIf_TimelockAdminRole() (gas: 17612)
[PASS] test_InitializeTimelockAdmin_RevertIf_AlreadyInitialized() (gas: 22084)
[PASS] test_InitializeTimelockAdmin_RevertIf_ZeroAddress() (gas: 4839724)
[PASS] test_Initialize_CannotReinitialize() (gas: 22553)
[PASS] test_Initialize_RevertIf_ZeroAddress() (gas: 4325476)
[PASS] test_Initialize_RolesGranted() (gas: 30975)
[PASS] test_Initialize_SiloCreated() (gas: 10910)
[PASS] test_Initialize_Success() (gas: 38300)
[PASS] test_LastDistributionTimestamp() (gas: 99737)
[PASS] test_MaxDeposit() (gas: 12754)
[PASS] test_MaxMint() (gas: 12741)
[PASS] test_Mint_Success() (gas: 94973)
[PASS] test_PreviewDeposit() (gas: 28580)
[PASS] test_PreviewMint() (gas: 28577)
[PASS] test_PreviewRedeem() (gas: 28582)
[PASS] test_PreviewWithdraw() (gas: 28580)
[PASS] test_Redeem_CooldownOff_Success() (gas: 116887)
```

```
[PASS] test_Redeem_RevertIf_CallerFullRestricted() (gas: 140267)
[PASS] test_Redeem_RevertIf_OwnerFullRestricted() (gas: 140243)
[PASS] test_Redeem_RevertIf_OwnerParameterIsFullRestricted() (gas: 168319)
[PASS] test_Redeem_RevertIf_ReceiverFullRestricted() (gas: 144373)
[PASS] test_Redeem_RevertIf_SoftRestrictedCaller() (gas: 142307)
[PASS] test_RedistributeLockedAmount_BurnDuringVesting_Success() (gas: 208471)
[PASS] test_RedistributeLockedAmount_FromNotFullRestricted() (gas: 114513)
[PASS] test_RedistributeLockedAmount_Success() (gas: 155121)
[PASS] test_RedistributeLockedAmount_ToAlsoFullRestricted() (gas: 172982)
[PASS] test_RemoveFromBlacklistBatch_Success() (gas: 94746)
[PASS] test_RemoveFromBlacklist_CallerIsOwner() (gas: 40943)
[PASS] test_RemoveFromBlacklist_Success() (gas: 39364)
[PASS] test_RemoveFromBlacklist_WhenOwner() (gas: 20825)
[PASS] test_RenounceRole_RevertAlways() (gas: 17453)
[PASS] test_ReportLoss_AfterDepositDuringVesting() (gas: 186435)
[PASS] test_ReportLoss_AtExactVestingBoundary() (gas: 121219)
[PASS] test_ReportLoss_AtVestingStart() (gas: 123484)
[PASS] test_ReportLoss_BurnsOnlyAvailableWhenInsufficient() (gas: 55039)
[PASS] test_ReportLoss_CanReportAfterVestingComplete() (gas: 116620)
[PASS] test_ReportLoss_CanReportMaxAssets() (gas: 49363)
[PASS] test_ReportLoss_DustAmount() (gas: 118455)
[PASS] test_ReportLoss_ExactlyEqualsUnvested() (gas: 115764)
[PASS] test_ReportLoss_LargeAmount_CancelsAllUnvestedAndBurns() (gas: 117728)
[PASS] test_ReportLoss_MinimalAmount_1Wei() (gas: 121832)
[PASS] test_ReportLoss_MultiplePartialCancellations() (gas: 141504)
[PASS] test_ReportLoss_NearVestingEnd() (gas: 122870)
[PASS] test_ReportLoss_PartialCancellation_Midway() (gas: 123833)
[PASS] test_ReportLoss_PartialCancellation_QuarterPoint() (gas: 123914)
[PASS] test_ReportLoss_PartialCancellation_ThreeQuarterPoint() (gas: 124002)
[PASS] test_ReportLoss_Precision_EdgeCase() (gas: 128668)
[PASS] test_ReportLoss_PreventDoS() (gas: 199857)
[PASS] test_ReportLoss_ProportionalLossForStakers() (gas: 177796)
[PASS] test_ReportLoss_RevertIf_NotRewarder() (gas: 39432)
[PASS] test_ReportLoss_RevertIf_ZeroAmount() (gas: 23426)
[PASS] test_ReportLoss_SequentialLosses_SameBlock() (gas: 135445)
[PASS] test_ReportLoss_ShareValueDecreases() (gas: 119093)
[PASS] test_ReportLoss_Success() (gas: 60624)
[PASS] test_ReportLoss_VeryLargeAmount() (gas: 129901)
[PASS] test_ReportLoss_VestingAmountCalculation() (gas: 117910)
[PASS] test_ReportLoss_WithActiveCooldown() (gas: 271140)
[PASS] test_ReportLoss_WithMultipleStakers() (gas: 191023)
[PASS] test_ReportLoss_WithNoVesting() (gas: 55858)
[PASS] test_ReportLoss_WithUnvestedRewards_CancelsAndBurns() (gas: 128407)
[PASS] test_RescueOrphanFunds_ResetsVestingState() (gas: 156065)
[PASS] test_RescueOrphanFunds_RevertIf_HasStakers() (gas: 23981)
[PASS] test_RescueOrphanFunds_RevertIf_NoOrphanFunds() (gas: 76372)
[PASS] test_RescueOrphanFunds_RevertIf_NotAdmin() (gas: 93341)
[PASS] test_RescueOrphanFunds_RevertIf_ZeroAddress() (gas: 89490)
[PASS] test_RescueOrphanFunds_Success() (gas: 129134)
[PASS] test_RescueTokens_RevertIf_AssetToken() (gas: 25012)
[PASS] test_RescueTokens_RevertIf_NotAdmin() (gas: 20752)
[PASS] test_RescueTokens_Success() (gas: 2716786)
[PASS] test_RevokeRole_RevertIf_TimelockAdminRole() (gas: 18262)
[PASS] test_RevokeRole_Success_NonTimelockRole() (gas: 40394)
[PASS] test_SetCooldownDuration_CanSetToZero() (gas: 25937)
[PASS] test_SetCooldownDuration_RevertIf_ExceedsMax() (gas: 20602)
[PASS] test_SetCooldownDuration_RevertIf_NotOwner() (gas: 18379)
[PASS] test_SetCooldownDuration_Success() (gas: 27757)
[PASS] test_SetTimelockAdmin_RevertIf_NotTimelockAdmin() (gas: 19831)
[PASS] test_SetTimelockAdmin_RevertIf_ZeroAddress() (gas: 18418)
[PASS] test_SetTimelockAdmin_Success() (gas: 50337)
[PASS] test_SharePriceIncreasesDuringVesting() (gas: 167560)
[PASS] test_TotalAssets_IncludesUnvestedRewards() (gas: 25401)
[PASS] test_TransferInRewards_RevertIf_NotRewarder() (gas: 58713)
[PASS] test_TransferInRewards_RevertIf_StillVesting() (gas: 124271)
[PASS] test_TransferInRewards_RevertIf_ZeroAmount() (gas: 58851)
[PASS] test_TransferInRewards_Success() (gas: 101699)
[PASS] test_TransferInRewards_VestingMechanism() (gas: 113529)
[PASS] test_TransferOwnership_TwoStepProcess() (gas: 61533)
[PASS] test_Transfer_RevertIf_FromBlacklisted() (gas: 124833)
[PASS] test_Transfer_RevertIf_ToBlacklisted() (gas: 127106)
[PASS] test_Transfer_Success() (gas: 123271)
[PASS] test_UnstakeAfterCooldown_Success() (gas: 174805)
[PASS] test_Unstake_RevertIf_CooldownNotReady() (gas: 165484)
[PASS] test_Unstake_WhenCooldownDisabled_Success() (gas: 183485)
```

```
[PASS] test_Upgrade_RevertIf_NotTimelockAdmin() (gas: 4267866)
[PASS] test_Upgrade_Success() (gas: 4319106)
[PASS] test_Version() (gas: 14407)
[PASS] test_VestingAmount() (gas: 100989)
[PASS] test_Withdraw_CooldownOff_Success() (gas: 119219)
[PASS] test_Withdraw_RevertIf_InnocentCallerAndReceiverFullRestrictedOwner() (gas: 193061)
[PASS] test_Withdraw_RevertIf_InnocentCallerFullRestrictedReceiver() (gas: 163424)
[PASS] test_Withdraw_RevertIf_OwnerFullRestricted() (gas: 168347)
[PASS] test_Withdraw_RevertIf_OwnerParameterIsFullRestricted() (gas: 168296)
[PASS] test_Withdraw_RevertIf_SoftRestrictedCaller() (gas: 145328)
[PASS] test_Withdraw_RevertIf_SoftRestrictedReceiver() (gas: 149774)
Suite result: ok. 139 passed; 0 failed; 0 skipped; finished in 144.02ms (198.79ms CPU time)

Ran 21 test suites in 161.42ms (1.08s CPU time): 693 tests passed, 0 failed, 0 skipped (693 total tests)
```

8.2 Automated Tools

8.2.1 AuditAgent

The AuditAgent is an AI-powered smart contract auditing tool that analyses code, detects vulnerabilities, and provides actionable fixes. It accelerates the security analysis process, complementing human expertise with advanced AI models to deliver efficient and comprehensive smart contract audits. Available at <https://app.auditagent.nethermind.io>.

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

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.