

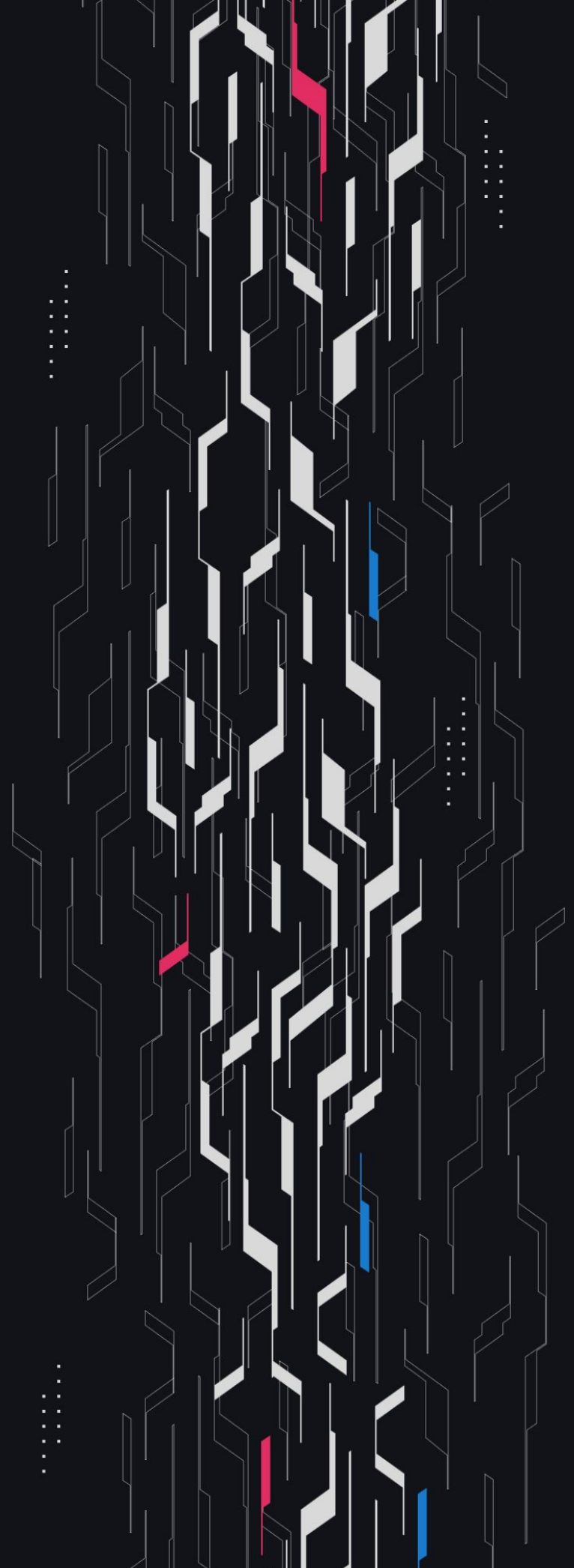
GA GUARDIAN

Nashpoint

**Contract Updates
Review**

Security Assessment

March 1st, 2026



Summary

Audit Firm Guardian

Prepared By Osman Ozdemir, CrypticDefense

Client Firm Nashpoint

Final Report Date March 1, 2026

Audit Summary

Nashpoint engaged Guardian to review the security of their Nashpoint Contract Review. From the 11th of February 2026 to the 19th of February 2026, a team of 2 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the minimal High and Critical issues detected during the main review, Guardian assigns a Confidence Ranking of 4 to the protocol. Guardian advises the protocol to consider periodic review with future changes. For detailed understanding of the Guardian Confidence Ranking, please see the rubric on the following page.

Note: Fixes to the findings uncovered in the remediation review have not been reviewed by Guardian.

✓ Verify the authenticity of this report on Guardian's GitHub: <https://github.com/guardianaudits>

Guardian Confidence Ranking

Confidence Ranking	Definition and Recommendation	Risk Profile
5: Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. Recommendation: Code is highly secure at time of audit. Low risk of latent critical issues.	0 High/Critical findings and few Low/Medium severity findings.
4: High Confidence	Code is clean, well-structured, and adheres to best practices. Only 1 Significant issue was uncovered per week. Design patterns are sound, and test coverage is strong. Recommendation: Suitable for deployment after remediations; consider periodic review with changes.	0-1 High/Critical findings per engagement week and little to no Medium severity issues. Varied Low severity findings.
3: Moderate Confidence	Medium-severity and occasional High-severity issues found. Code is functional, but there are concerning areas (e.g., weak modularity, risky patterns). No critical design flaws, though some patterns could lead to issues in edge cases. Recommendation: Address issues thoroughly and consider a targeted follow-up audit depending on code changes.	1-2 High/Critical findings per engagement week.
2: Low Confidence	Code shows frequent emergence of Critical/High vulnerabilities. Audit revealed recurring anti-patterns, weak test coverage, or unclear logic. These characteristics suggest a high likelihood of latent issues. Recommendation: Post-audit development and a second audit cycle are strongly advised.	2-4 High/Critical findings per engagement week. Or additional High/Critical findings uncovered in remediation review which have not been resolved and confirmed by Guardian.
1: Very Low Confidence	Code has systemic issues. Multiple High/Critical findings (≥ 5/week), poor security posture, and design flaws that introduce compounding risks. Safety cannot be assured. Recommendation: Halt deployment and seek a comprehensive re-audit after substantial refactoring.	≥ 5 High/Critical findings and overall systemic flaws.

Table of Contents

Project Information

Project Overview 5

Audit Scope & Methodology 6

Smart Contract Risk Assessment

Findings & Resolutions 9

Addendum

Disclaimer 33

About Guardian 34

Project Overview

Project Summary

Project Name	Nashpoint
Language	Solidity
Codebase	https://github.com/nashpoint/nashpoint-smart-contracts
Commit(s)	Main Review commit: 6ed091ed827fd0458bd7fa862dcd1be5258bd1ca Remediation Review commit: be28bfa85e7d766d4adb4023f47a4a3e782b295b

Audit Summary

Delivery Date	March 1, 2026
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0
● High	1	0	0	0	0	1
● Medium	2	0	0	0	1	1
● Low	4	0	0	3	0	1
● Info	14	0	0	10	0	4

Audit Scope & Methodology

Contract Updates:

<https://github.com/nashpoint/nashpoint-smart-contracts/compare/88bbe3ba923d9383c61210418c8f881a016e9903...6ed091ed827fd0458bd7fa862dcd1be5258bd1ca>

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: <i>High</i>	Impact: <i>Medium</i>	Impact: <i>Low</i>
Likelihood: <i>High</i>	● Critical	● High	● Medium
Likelihood: <i>Medium</i>	● High	● Medium	● Low
Likelihood: <i>Low</i>	● Medium	● Low	● Low

Impact

- High** Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.
- Medium** A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.
- Low** Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

- High** The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.
- Medium** An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.
- Low** Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.
Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	Event Verification DoS Due To readList Cap	DoS	● High	Resolved
M-01	claimableRedeemRequest Dilutes Dividends	Logical Error	● Medium	Resolved
M-02	No Ex-Dividend Enforcement At Node Level	MEV	● Medium	Partially Resolved
L-01	Unsafe Cast When Setting lastFundPrice	Validation	● Low	Resolved
L-02	Fund Token Clawback Breaks Adapter Redemptions	Compatibility	● Low	Acknowledged
L-03	Strict Equality And BalanceOf Could Cause DoS	Validation	● Low	Acknowledged
I-01	Event Verifier Assumes Standard ERC20 Compliance	Warning	● Info	Acknowledged
I-02	DuplicateComponent Error Is Dead Code	Superfluous Code	● Info	Resolved
I-03	_getBlockHash Can Migrate To OZ Blockhash Lib	Trust Assumptions	● Info	Resolved
I-04	Failed settleRedeem Temporarily Blocks Dividends	Unexpected Behavior	● Info	Acknowledged
I-05	Small Dividends Can Round To Zero	Rounding	● Info	Acknowledged
I-06	De-Whitelisted Nodes Still Receive Dividends	Access Control	● Info	Acknowledged
I-07	New Deposits Can Capture Unearned Dividends	Warning	● Info	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
I-08	Settlement Type Relies On OffChain Parsing	Trust Assumptions	● Info	Acknowledged
I-09	settleDividend Should Use mulDiv For Consistency	Best Practices	● Info	Resolved
I-10	Implementations Can Be Initialized	Best Practices	● Info	Resolved

H-01 | Event Verification DoS Due To readList Cap

Category	Severity	Location	Status
DoS	● High	TransferEventVerifier.sol: 97-101	Resolved

Description [PoC](#)

The `RLPReader` library imposes a hard limit of 32 elements when reading RLP lists and does not support lists exceeding this size. [Reference](#)

Consequently, any transaction that emits more than 32 logs cannot be verified by the current event verification logic, despite being valid.

Note that although `DigiFT`'s event verification has the same limitation, it is unlikely to result in a DoS, as the `SubRedManagement` contract can emit only one `SettleSubscriber` and one `SettleRedemption` event per settlement batch, since investor addresses and quantities included as arrays within the event.

However, this assumption does not hold for `WisdomTree`. Since each transfer or mint emits an individual event, a single dividend batch mint can realistically result in more than 32 event logs, which cannot be verified.

Recommendation

Do not rely on Optimism's `RLPReader` library for verifying `WisdomTree` transfers and dividend events, especially when batch operations are expected. Consider using `OpenZeppelin`'s RLP library, which does not impose a fixed limit on [RLP list length](#)

However, it is important to review and account for the limitations and assumptions of the `OpenZeppelin` implementation, as documented in its [source code](#).

Resolution

Nashpoint Team: The issue was resolved in commit [56cc082](#).

M-01 | claimableRedeemRequest Dilutes Dividends

Category	Severity	Location	Status
Logical Error	● Medium	WTAdapter.sol: 118	Resolved

Description

When Nodes invest in WisdomTree, all WT fund tokens are held by the WTAdapter contract, and Nodes receive corresponding WTAdapter shares as receipts. When WisdomTree mints dividends, it mints all dividends to the adapter contract, which are later distributed to participating Nodes on a pro-rata basis.

Dividend distribution is determined based on a Node's WTAdapter receipt token balance, pendingRedeemRequest, claimableRedeemRequest, and maxMint. However, including claimableRedeemRequest in this calculation results in an incorrect distribution, as the adapter contract never receives dividends corresponding to that amount.

The reason is that the WTAdapter contract does not hold the fund tokens corresponding to the claimableRedeemRequest portion in custody at the time of the WisdomTree dividend minting.

Assuming that WT dividends are minted after deposit and redeem settlements (as in the test cases), the token custody state for the weight calculation parameters is as follows:

- balanceOf(node): WTAdapter receipt tokens are held by the Node, the corresponding WT fund tokens are held by the adapter contract.
- pendingRedeemRequest: WTAdapter receipt tokens are held by the adapter contract, and the corresponding WT fund tokens are also held by the adapter contract.
- maxMint: WTAdapter receipt tokens have not yet been minted; however, the corresponding WT fund tokens are transferred to the adapter contract prior to settleDeposit and are held by the adapter contract.
- claimableRedeemRequest: WTAdapter receipt tokens have not yet been burned and remain in the adapter contract. However, the corresponding WT fund tokens are already transferred to the WT receiver wallet during forwardRequests, well before settlement, and are therefore held by the WT receiver wallet. Not the adapter contract.

As a result, when the WisdomTree protocol mints dividends based on the amount of WT fund tokens custodied by the adapter, dividends corresponding to the claimableRedeemRequest portion are never minted as the corresponding WT fund tokens are not in the adapter at the dividend minting time.

Additionally, the pendingRedeemRequest and maxMint values can also introduce inconsistencies depending on the timing of the dividend mint. The highest-risk period is between the forwardRequests call and the settleDeposit / settleRedeem calls. If dividends are minted during this window, larger miscalculations can occur, as the WT fund tokens corresponding to pendingRedeemRequest are are custodied in the WT receiver wallet, and the adapter would not receive dividends for that portion either.

Risks related to pendingRedeemRequest and maxMint can be mitigated by managers through correct timing of request forwarding. However, including claimableRedeemRequest results in incorrect distribution regardless of when dividends are minted.

Another issue with including claimableRedeemRequest is that the asset-to-share ratio is locked at settlement time. A Node can submit a redemption request before dividend distribution, lock in the pre-dividend asset price, and then withdraw after dividends are distributed. As a result, the Node avoids the post-dividend price decrease and also receive a portion of the dividend, leading to an unfair advantage.

Recommendation

Do not include claimableRedeemRequest in the dividend weight calculation. For pendingRedeemRequest and maxMint, ensure that managers carefully time the request forwarding process and document the associated risks.

Resolution

Nashpoint Team: The issue was resolved in commit [ab25b7d](#).

M-02 | No Ex-Dividend Enforcement At Node Level

Category	Severity	Location	Status
MEV	● Medium	WTAdapter.sol: 136	Partially Resolved

Description

After the `ex-dividend date`, the fund's oracle price drops to reflect the upcoming `dividend distribution`. In traditional finance, investors who purchase shares after the `ex-dividend date` are not entitled to the dividend. However, the Node has no mechanism to enforce this cutoff.

A user can deposit into the Node after the `ex-dividend date` but before the actual dividend distribution. When `settleDividend` executes and mints additional adapter shares to the Node, the Node's underlying value increases (more adapter shares x post `ex-div` price).

After `cacheTotalAssets` refreshes via `startRebalance()` or `updateTotalAssets()`, the depositor's shares reflect this increased value, allowing them to capture dividends they were not entitled to.

The team has indicated they plan to time operations at the Node level to prevent this, and one possible approach is letting `isCacheValid()` expire between the `ex-dividend date` and distribution, which blocks deposits via `maxDeposit()` returning 0.

However, this is only feasible for short windows and relies on deliberately skipping rebalances, which also blocks redemption processing, fee payments, and all rebalance-dependent operations.

Recommendation

Document the requirement to block Node deposits between the `ex-dividend date` and dividend distribution. Consider extending `rebalanceCooldown` around the `ex-dividend date`, or introducing a deposit pause mechanism (at the Node level).

Resolution

Nashpoint Team: The issue was resolved in commit [f0ac6d8](#).

L-01 | Unsafe Cast When Setting lastFundPrice

Category	Severity	Location	Status
Validation	● Low	AdapterBase.sol: 571-572	Resolved

Description

AdapterBase.initialize and forceUpdateLastPrice functions reads the latestRoundData and blindly casts the int256 answer to uint256 when setting the lastFundPrice.

Previously, these actions used the getPrice, which was already returning uint256. If the oracle ever emits a negative value (e.g., stale feed or sentinel), the cast wraps to a huge positive number, corrupting lastFundPrice.

Recommendation

Ensure the returned answer > 0 before casting and revert with BadPriceOracle error otherwise.

Resolution

Nashpoint Team: The issue was resolved in commit [3f96145](#).

L-02 | Fund Token Clawback Breaks Adapter Redemptions

Category	Severity	Location	Status
Compatibility	● Low	WTAdapter.sol: 193	Acknowledged

Description

The WisdomTree fund token implements a clawback function allowing its registrar to forcibly transfer tokens from any holder, including the WTAdapter, for "regulatory compliance, legal orders, emergency asset recovery, correcting erroneous transactions".

If fund shares are clawed back from the adapter, _fundRedeem will revert when attempting to safeTransfer fund tokens it no longer holds, bricking all pending and future redemptions.

The adapter has no mechanism to handle this case. Nodes would hold adapter shares they cannot redeem, and forwardRequests would continue attempting to transfer fund tokens that don't exist.

Recommendation

Consider implementing a function that allows the manager to re-sync adapter state with its actual fund token balance after a clawback event.

Resolution

Nashpoint Team: Acknowledged.

L-03 | Strict Equality And BalanceOf Could Cause DoS

Category	Severity	Location	Status
Validation	● Low	WTAdapter.sol: 125	Acknowledged

Description

For the settleDividend call to be successful, totalWeight must be strictly equal to totalSupply + totalMaxMint. totalWeight is calculated using the balances of each participating node.

This means that the manager must include all nodes that have the WTAdapter as a component when calling the function, even if a node holds only a small amount of WTAdapter shares, in order to satisfy the totalSupply check.

While all nodes must be whitelisted to participate and this does not pose an immediate threat, it may introduce multiple issues in future development:

- 1. totalSupply check: As the number of whitelisted nodes grows, the manager may be unable to supply all nodes to the function due to gas limitations.
- 2. balanceOf(node) usage: Current routers do not transfer WTAdapter shares out of the node.

However, if future development introduces new router types that allow investing component shares (e.g., via looping mechanisms) or allow transfers, dividend settlement would become impossible, as relying on balanceOf(node) would cause the check to fail.

Recommendation

Be aware of these risks when adding nodes to whitelist and during future development processes.

Resolution

Nashpoint Team: Acknowledged.

I-01 | Event Verifier Assumes Standard ERC20 Compliance

Category	Severity	Location	Status
Warning	● Info	TransferEventVerifier.sol: 109-111	Acknowledged

Description

The `TransferEventVerifier` assumes the fund token complies with the `ERC20` standard and emits a `Transfer` event where the `from` and `to` parameters are indexed and the `amount` parameter is non-indexed.

If the underlying fund does not emit standard-compliant events with indexed parameters, event verification will fail.

Although the expected fund contracts are not currently available, it is important to ensure that they emit events with two indexed address parameters and one non-indexed `uint256` amount parameter.

Recommendation

Ensure that all events emitted by the `WisdomTree` funds are `ERC20`-standard compliant.

Resolution

Nashpoint Team: Acknowledged.

I-02 | DuplicateComponent Error Is Dead Code

Category	Severity	Location	Status
Superfluous Code	● Info	ErrorsLib.sol: 136-137	Resolved

Description

ErrorsLib.DuplicateComponent is declared but never referenced anywhere in the codebase. It was previously used by _validateNoDuplicateComponents, which has since been removed. The error definition should have been cleaned up alongside that removal.

Recommendation

Remove the unused DuplicateComponent error from ErrorsLib.

Resolution

Nashpoint Team: The issue was resolved in commit [9028b37](#).

I-03 | _getBlockHash Can Migrate To OZ Blockhash Lib

Category	Severity	Location	Status
Trust Assumptions	● Info	EventVerifierBase.sol: 121-122	Resolved

Description

The `_getBlockHash` function in `EventVerifierBase` relies on the `setBlockHash` admin function to handle blocks older than 256, which may introduce risks as it allows the registry owner to set arbitrary block hashes. `EIP-2935` provides trustless access to the last 8,191 block hashes (rather than 256 limit).

The `TODO` in `_getBlockHash` states that migration to `Blockhash.sol` will be done when `OpenZeppelin` v5.6.0 is stable. However, `EIP-2935` was a part of the `Pectra` upgrade in 2025, where `OpenZeppelin` v5.4.0 shipped a `Blockhash` library that already allows this functionality.

Recommendation

Upgrade to `OpenZeppelin Contracts` v5.4.0+ and replace `_getBlockHash` with the `Blockhash` library. <https://github.com/OpenZeppelin/openzeppelin-contracts/releases/tag/v5.4.0>

Resolution

Nashpoint Team: The issue was resolved in commit [8634234](#).

I-04 | Failed settleRedeem Temporarily Blocks Dividends

Category	Severity	Location	Status
Unexpected Behavior	● Info	AdapterBase.sol: 809-813	Acknowledged

Description

settleDividend requires both pendingDepositRequest == 0 and pendingRedeemRequest == 0. If settleRedeem reverts due to the settlementDeviation check, pendingRedeemRequest remains non-zero, blocking dividend distribution until the redemption is successfully settled.

This means any settlement price deviation can delay dividend distribution to all nodes, even though dividends are unrelated to the redemption flow.

Recommendation

Document this case. Alternatively, consider whether settleDividend needs the pendingRedeemRequest == 0 check, or if dividends could safely be distributed while a redemption is pending.

Resolution

Nashpoint Team: Acknowledged.

I-05 | Small Dividends Can Round To Zero

Category	Severity	Location	Status
Rounding	● Info	WTAdapter.sol: 129	Acknowledged

Description

In `settleDividend`, each node's share is calculated as `dividends * weights[i] / totalWeight`, which rounds down.

When `dividends` is small relative to `totalWeight` (more likely when the fund token has low decimals), all nodes except the last may receive zero. The dust correction assigns the entire remainder to the last node in the array (determined by the manager).

Recommendation

Consider refactoring the dividend distribution to handle cases where any node's calculated share rounds to zero. Additionally, document that fund tokens with low decimals increase the likelihood of rounding issues in dividend distribution.

Resolution

Nashpoint Team: Acknowledged.

I-06 | De-Whitelisted Nodes Still Receive Dividends

Category	Severity	Location	Status
Access Control	● Info	WTAdapter.sol: 98	Acknowledged

Description

settleDividend does not check nodeWhitelisted[nodes[i]] before distributing. The invariant `totalWeight == totalSupply() + totalMaxMint` forces inclusion of every address holding shares, so a de-whitelisted node that still has a share balance continues receiving minted dividend shares with no way to opt it out (as `requestRedeem` enforces whitelisted nodes only).

Recommendation

No changes needed if this is intentional or expected behavior. Otherwise, consider validating whitelist status within the `settleDividend` loop, or providing a mechanism to exclude de-whitelisted nodes from dividend distribution.

Resolution

Nashpoint Team: Acknowledged.

I-07 | New Deposits Can Capture Unearned Dividends

Category	Severity	Location	Status
Warning	● Info	WTAdapter.sol: 98	Acknowledged

Description

`settleDividend` distributes shares based off the share weight of each node at the time of distribution. A node that deposits and settles shortly before a dividend distribution receives the same per-share dividend as nodes that have held shares for the full accrual period.

The `totalWeight == totalSupply() + totalMaxMint` invariant prevents the manager from excluding a node from dividend distribution, so there is no mechanism to differentiate between long held and recently acquired shares.

The team has acknowledged this and plans to handle it by timing `deposit/redeem requests` around dividend payouts (processing new deposits only after dividends are distributed). However, this relies entirely on off-chain coordination.

Recommendation

Document the requirement to time deposits around dividend distributions. Consider if a deposit cooldown period before dividend eligibility would help reduce the need for manual timing.

Resolution

Nashpoint Team: Acknowledged.

I-08 | Settlement Type Relies On OffChain Parsing

Category	Severity	Location	Status
Trust Assumptions	● Info	WTAdapter.sol: 168	Acknowledged

Description

For both the deposit settlement flow and the dividend settlement flow, the verified events are identical Transfer events emitted with the from address set to `address(0)`, and verification is performed using `TransferEventVerifier.OnchainArgs(fund, address(0))`.

Consequently, `WTAdapter` relies on off-chain event parsing to distinguish whether a Transfer log represents a deposit mint or a dividend mint. A misclassified log (e.g., a dividend mint as a deposit, or vice versa) causes the adapter to run the wrong settlement branch, which would revert.

Recommendation

Ensure that the off-chain indexer can reliably distinguish whether a Transfer log represents a dividend settlement or a deposit settlement.

Resolution

Nashpoint Team: Acknowledged.

I-09 | settleDividend Should Use mulDiv For Consistency

Category	Severity	Location	Status
Best Practices	● Info	WTAdapter.sol: 129	Resolved

Description

settleDividend uses `raw dividends * weights[i] / totalWeight` for pro-rata calculation, while `settleDeposit` and `settleRedeem` in `AdapterBase` use `OpenZeppelin's Math.mulDiv` for equivalent calculations.

Although overflow is practically unreachable with realistic token supplies, the inconsistency could become a concern if the `WTAdapter` support tokens with unusual denominations.

Recommendation

Use `Math.mulDiv(dividends, weights[i], totalWeight)` for consistency with the rest of the codebase.

Resolution

Nashpoint Team: The issue was resolved in commit [e52ee94](#).

I-10 | Implementations Can Be Initialized

Category	Severity	Location	Status
Best Practices	● Info	AdapterBase.sol: 378-379	Resolved

Description

Adapters use the Beacon proxy pattern and deployed through factory contracts. All of these proxies are atomically initialized at the time of deployment. However, the implementation contract stay uninitialized and can be initialized by anyone.

Adapters use the Beacon proxy pattern and are deployed through factory contracts. All of these proxies are atomically initialized at the time of deployment. However, the implementation contract remains uninitialized and can be initialized by anyone.

While this does not affect the state of the Beacon proxies or directly affect proxy behavior, it alters the implementation’s storage. Initializing the implementation contract should be disabled as a best practice.

Recommendation

Consider adding `_disableInitializers` in implementation contract constructors.

Resolution

Nashpoint Team: The issue was resolved in commit [616e6f0](#).

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
L-01	Stale Price Arbitrage During Price Freeze	MEV	Low	Acknowledged
I-01	Operational Limitations Related To Price Freeze	Warning	Info	Acknowledged
I-02	Price Update May Fail Due To Freeze	Warning	Info	Acknowledged
I-03	forceUpdateLastPrice Bypasses Freeze	Informational	Info	Acknowledged
I-04	Temporary totalAssets Inflation Risk	Warning	Info	Acknowledged

L-01 | Stale Price Arbitrage During Price Freeze

Category	Severity	Location	Status
MEV	● Low	WTAdapter.sol: 152	Acknowledged

Description

During the price freeze window, `convertToShares` and `convertToAssets` both use the cached `lastFundPrice`. If the underlying NAV moves for non-dividend reasons (i.e., market movements), a discrepancy between the frozen price and real NAV creates arbitrage opportunities.

Redeemers can exit at the stale high price when real value is lower, extracting value from remaining holders. Conversely, depositors benefit if real value rises above the frozen price.

Recommendation

Be aware of this behavior. If this is an accepted trade-off, ensure that the freeze window is as short as possible and monitor for sudden NAV changes during the freeze.

If an unexpected NAV change occurs during the freeze, consider using forced price updates to reflect the real value at the node level.

If this is not acceptable, reconsider the price freeze approach or explore alternatives that do not expose the protocol to stale price arbitrage during the freeze window.

Resolution

Nashpoint Team: Acknowledged.

I-01 | Operational Limitations Related To Price Freeze

Category	Severity	Location	Status
Warning	● Info	WTAdapter.sol: 153-154	Acknowledged

Description

A temporary price-freezing feature is added to protect against post–ex-dividend price drops. For the price to be frozen, there must be no pending forwarded requests, meaning all deposits and redemptions must be settled before the freeze.

Assuming there are 2 days between the ex-dividend date and the dividend distribution on the WisdomTree protocol, and dividends are distributed on day 30:

- Dividend distribution date: Day 30
- Ex-dividend date: Day 28
- Price freeze must occur just before the ex-dividend date (assumed earlier that day): Day 28
- All settlements must be completed on or before: Day 28
- Given that settlements also take time (assumed 2–3 days), the latest forwardRequests call should occur around: ~Day 25

While the price freeze period itself lasts only 2 days, the actual window between the latest request forwarding before the freeze and the first request forwarding after unfreeze could be up to 5 days, or potentially even a week (Note that these figures are illustrative only, and the actual values depend on WisdomTree’s configuration).

Managers should be aware of these limitations and stop forwarding requests even before the expected freeze day. Forwarding requests a few days before the expected freeze date may prevent the price from being frozen if WisdomTree cannot complete settlements within that period.

Recommendation

Consider documenting these behaviors for both managers and users.

Resolution

Nashpoint Team: Acknowledged.

I-02 | Price Update May Fail Due To Freeze

Category	Severity	Location	Status
Warning	● Info	AdapterBase.sol: 603	Acknowledged

Description

The fixes introduce a price-freezing mechanism with a maximum freeze period of 7 days. Once the freeze ends, the first required action is to refresh the price.

However, depending on the length of the freeze period, the price change over this window is more likely to exceed the deviation threshold compared to regular daily price updates.

As a result, regular `updateLastPrice` may revert with `PriceNotInRange` error, forcing the registry owner to use `forceUpdateLastPrice`.

Recommendation

Be aware of this behavior when using the price freeze mechanism.

Resolution

Nashpoint Team: Acknowledged.

I-03 | forceUpdateLastPrice Bypasses Freeze

Category	Severity	Location	Status
Informational	● Info	AdapterBase.sol: 575	Acknowledged

Description

The forceUpdateLastPrice function bypasses priceFreezeActive, similar to how it bypasses deviation checks. No change is needed if this behavior is intentional. This issue is intended only to flag the bypass and confirm that it is expected.

The forceUpdateLastPrice can be used in its current form when required if a situation mentioned in L-01 occurs.

Recommendation

No change is needed if this is the desired behavior.

Resolution

Nashpoint Team: Acknowledged.

I-04 | Temporary totalAssets Inflation Risk

Category	Severity	Location	Status
Warning	● Info	AdapterBase.sol: 1066-1068	Acknowledged

Description

The price freeze mechanism implemented in `WTAdapter` prevents price recovery arbitrage during the ex-dividend window.

After distribution, the oracle price drops to reflect the dividend payout, and the newly minted shares compensate so `totalAssets` remains roughly unchanged once the price updates.

However, if `endPriceFreeze` and `updateLastPrice` are not executed alongside `settleDividend` before a Node-level cache refresh, the Node will temporarily reflect an inflated `totalAssets` (increased shares × stale high price).

If a Node cache refresh occurs during this window, depositors could capture value they were not entitled to.

Recommendation

Ensure `endPriceFreeze` and `updateLastPrice` are executed alongside `settleDividend`, before any Node-level cache refresh occurs.

Resolution

Nashpoint Team: Acknowledged.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian

Founded in 2022 by DeFi experts, Guardian is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit <https://guardianaudits.com>

To view our audit portfolio, visit <https://github.com/guardianaudits>

To book an audit, message <https://t.me/guardianaudits>