GA GUARDIAN

Nashpoint

Contracts Review

Security Assessment

December 10th, 2025

Summary

Audit Firm Guardian

Prepared By Vladimir Zotov, Osman Ozdemir, 0x37 0x, Elhaj, CrypticDefense

Client Firm Nashpoint

Final Report Date December 10, 2025

Audit Summary

Nashpoint engaged Guardian to review the security of their Nashpoint Contracts. From the 27th of October to the 13th of November, a team of 5 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

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. Thus, 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 Overview

Project Summary

Project Name	Nashpoint
Language	Solidity
Codebase	https://github.com/nashpoint/nashpoint-smart-contracts
Commit(s)	Main Review commit: 06d948dd3ac126c389d8cf8f9fd53347a73b5059 Remediation Review commit: 88bbe3ba923d9383c61210418c8f881a016e9903

Audit Summary

Delivery Date	December 10, 2025
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	2	0	0	0	0	2
Medium	14	0	0	2	0	12
Low	30	0	0	18	1	11
Info	20	0	0	10	0	10

Audit Scope & Methodology

```
contract,source,total,comment
nashpoint-smart-contracts/src/Escrow.sol,15,27,8
nashpoint-smart-contracts/src/Node.sol,587,832,102
nashpoint-smart-contracts/src/NodeFactory.sol,37,59,11
nashpoint-smart-contracts/src/NodeRegistry.sol,98,151,27
nashpoint-smart-contracts/src/routers/ERC4626Router.sol,112,221,70
nashpoint-smart-contracts/src/routers/ERC7540Router.sol,147,292,94
nashpoint-smart-contracts/src/routers/FluidRewardsRouter.sol,21,42,14
nashpoint-smart-contracts/src/routers/IncentraRouter.sol,20,34,8
nashpoint-smart-contracts/src/routers/MerklRouter.sol,19,37,11
nashpoint-smart-contracts/src/routers/OneInchV6RouterV1.sol,66,148,52
nashpoint-smart-contracts/src/quoters/QuoterV1.sol,69,163,66
nashpoint-smart-contracts/src/libraries/BaseComponentRouter.sol,84,184,72
nashpoint-smart-contracts/src/libraries/BaseQuoter.sol,24,46,14
nashpoint-smart-contracts/src/libraries/ErrorsLib.sol,61,175,56
nashpoint-smart-contracts/src/libraries/EventsLib.sol,45,126,40
nashpoint-smart-contracts/src/libraries/MathLib.sol,12,16,1
nashpoint-smart-contracts/src/libraries/NodeLib.sol,43,75,26
nashpoint-smart-contracts/src/libraries/PolicyLib.sol,44,59,1
nashpoint-smart-contracts/src/libraries/RegistryAccessControl.sol,31,52,14
nashpoint-smart-contracts/src/policies/CapPolicy.sol,24,40,8
nashpoint-smart-contracts/src/policies/GatePolicy.sol,26,36,6
nashpoint-smart-contracts/src/policies/NodePausingPolicy.sol,57,86,15
nashpoint-smart-contracts/src/policies/PolicyBase.sol,56,79,7
nashpoint-smart-contracts/src/policies/ProtocolPausingPolicy.sol,80,114,15
nashpoint-smart-contracts/src/policies/TransferPolicy.sol,27,35,4
nashpoint-smart-contracts/src/policies/WhitelistBase.sol,48,77,15
nashpoint-smart-contracts/src/adapters/digift/DigiftAdapter.sol,460,1167,523
nashpoint-smart-contracts/src/adapters/digift/DigiftAdapterFactory.sol,14,58,35
nashpoint-smart-contracts/src/adapters/digift/DigiftEventVerifier.sol,135,332,147
source count: {
  total: 4763,
  source: 2462,
  comment: 1462,
  single: 854,
  block: 608,
  mixed: 1,
  empty: 840,
  todo: 0,
  blockEmpty: 0,
  commentToSourceRatio: 0.5938261575954509
}
```

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	🔴 Critical	🟠 High	🟡 Medium
Likelihood: *Medium*	🟠 High	🟡 Medium	🟢 Low
Likelihood: *Low*	🟡 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.

Invariants Assessed

During Guardian's review of Nashpoint, fuzz-testing was performed on the protocol's main functionalities. Given the dynamic interactions and the potential for unforeseen edge cases in the protocol, fuzz-testing was imperative to verify the integrity of several system invariants.

Throughout the engagement the following invariants were assessed for a total of 10,000,000+ runs with a prepared fuzzing suite.

ID	Description	Tested	Passed	Remediation	Run Count
DIGIFT-01	Global Pending Deposit Must Match Forwarded Amount	✅	✅	✅	10M+
DIGIFT-02	Global Pending Redeem Must Match Forwarded Amount	✅	✅	✅	10M+
DIGIFT-03	No Pending Deposits Must Remain After Settle	✅	✅	✅	10M+
DIGIFT-04	No Pending Redemptions Must Remain After Settle	✅	✅	✅	10M+
DIGIFT-05	Max Mintable Shares Must Be Non-Zero After Settle Deposit	✅	✅	✅	10M+
DIGIFT-06	Max Withdrawable Assets Must Be Non-Zero After Settle Redeem	✅	✅	✅	10M+
DIGIFT-07	Total Max Withdrawable Must Match Expected Assets	✅	✅	✅	10M+
DIGIFT-08	Withdraw Assets Must Match Max Withdraw Before	✅	✅	✅	10M+
DIGIFT-09	Node Balance Must Not Decrease After Withdraw	✅	✅	✅	10M+
DIGIFT-10	Max Withdraw Must Be Zero After Withdraw	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
DIGIFT-11	Pending Redeem Must Increase After Request	✅	✅	✅	10M+
DIGIFT-12	Balance Must Not Increase After Request Redeem	✅	✅	✅	10M+
FACTORY-01	Deployed Node Address Must Not Be Zero	✅	✅	✅	10M+
FACTORY-02	Deployed Escrow Address Must Not Be Zero	✅	✅	✅	10M+
FACTORY-03	Node Escrow Link Must Match Deployed Escrow	✅	✅	✅	10M+
FACTORY-04	Node Asset Must Match Init Args Asset	✅	✅	✅	10M+
FACTORY-05	Node Owner Must Match Init Args Owner	✅	✅	✅	10M+
FACTORY-06	Node Total Supply Must Be Zero After Deploy	✅	✅	✅	10M+
FACTORY-07	Node Must Be Registered In Registry After Deploy	✅	✅	✅	10M+
NODE-01	User Share Balance Must Increase After Successful Deposit/Mint	✅	✅	✅	10M+
NODE-02	Escrow Share Balance Must Increase By The Redemption Request Amount	✅	✅	✅	10M+
NODE-03	Escrow Share Balance Must Decrease After A Redeem Is Finalized	✅	✅	✅	10M+
NODE-04	User Asset Balance Must Increase By Requested Asset Amount After Withdraw	✅	✅	✅	10M+
NODE-05	Escrow Asset Balance Must Be >= Sum Of All claimableAssets	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
NODE-06	Component's Asset Ratio Should Not Exceed Target After Invest	✅	✅	✅	10M+
NODE-07	Node's Reserve Should Not Decrease Below Target After Invest	✅	✅	✅	10M+
NODE-08	Receiver Share Balance Must Increase By Minted Shares After Deposit	✅	✅	✅	10M+
NODE-09	Node Asset Balance Must Increase By Deposited Assets	✅	✅	✅	10M+
NODE-10	Node Total Assets Must Increase By Deposited Assets	✅	✅	✅	10M+
NODE-11	Node Total Supply Must Increase By Minted Shares	✅	✅	✅	10M+
NODE-12	Receiver Share Balance Must Increase By Minted Shares	✅	✅	✅	10M+
NODE-13	Receiver Asset Balance Must Decrease By Assets Spent	✅	✅	✅	10M+
NODE-14	Node Total Assets Must Increase By Assets Spent	✅	✅	✅	10M+
NODE-15	Node Total Supply Must Increase By Requested Shares	✅	✅	✅	10M+
NODE-16	Owner Share Balance Must Decrease By Requested Shares	✅	✅	✅	10M+
NODE-17	Pending Redeem Must Increase By Requested Shares	✅	✅	✅	10M+
NODE-18	Claimable Redeem Must Remain Unchanged After Request	✅	✅	✅	10M+
NODE-19	Claimable Assets Must Remain Unchanged After Request	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
NODE-20	Pending Redeem Must Decrease After Fulfill	✅	✅	✅	10M+
NODE-21	Claimable Redeem Must Increase After Fulfill	✅	✅	✅	10M+
NODE-22	Claimable Assets Must Decrease By Withdrawn Amount	✅	✅	✅	10M+
NODE-23	Escrow Asset Balance Must Decrease By Withdrawn Amount	✅	✅	✅	10M+
NODE-24	Pending Redeem Must Decrease By Finalized Shares	✅	✅	✅	10M+
NODE-25	Claimable Redeem Must Increase By Finalized Shares	✅	✅	✅	10M+
NODE-26	Claimable Assets Must Increase By Returned Assets	✅	✅	✅	10M+
NODE-27	Escrow Asset Balance Must Increase By Returned Assets	✅	✅	✅	10M+
NODE-28	Node Asset Balance Must Decrease By Returned Assets	✅	✅	✅	10M+
NODE-29	Claimable Redeem Must Decrease By Redeemed Shares	✅	✅	✅	10M+
NODE-30	Claimable Assets Must Decrease By Returned Assets	✅	✅	✅	10M+
NODE-31	Receiver Asset Balance Must Increase By Returned Assets	✅	✅	✅	10M+
NODE-32	Escrow Asset Balance Must Decrease By Returned Assets	✅	✅	✅	10M+
NODE-33	Component Must Be Registered After Add	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
NODE-34	Component Must Be Unregistered After Remove	✅	✅	✅	10M+
NODE-35	Node Balance Must Decrease By Rescued Amount	✅	✅	✅	10M+
NODE-36	Recipient Balance Must Increase By Rescued Amount	✅	✅	✅	10M+
NODE-37	Policy Must Be Registered After Add	✅	✅	✅	10M+
NODE-38	Policy Must Be Unregistered After Remove	✅	✅	✅	10M+
NODE-39	Component Balance Must Increase By Delta After Gain Backing	✅	✅	✅	10M+
NODE-40	Component Balance Must Decrease By Delta After Lose Backing	✅	✅	✅	10M+
NODE-41	Shares Exiting Must Not Exceed Total Supply	✅	✅	✅	10M+
ONEINCH-01	Asset Token Balance Of Node Must Increase After Successful Swap	✅	✅	✅	10M+
ONEINCH-02	All Incentive Token Input Must Be Used During Swap	✅	✅	✅	10M+
ONEINCH-03	Node Must Receive At Least 99% Of Min Assets Out	✅	✅	✅	10M+
ONEINCH-04	Node Must Spend Exact Incentive Amount	✅	✅	✅	10M+
ONEINCH-05	Executor Must Receive At Least Incentive Amount	✅	✅	✅	10M+
POOL-01	Pending Deposits Must Not Increase After Process	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
POOL-02	Pending Redemptions Must Be Zero After Process	✅	✅	✅	10M+
REGISTRY-01	Protocol Fee Address Must Match Set Value	✅	✅	✅	10M+
REGISTRY-02	Protocol Management Fee Must Match Set Value	✅	✅	✅	10M+
REGISTRY-03	Protocol Execution Fee Must Match Set Value	✅	✅	✅	10M+
REGISTRY-04	Policies Root Must Match Set Value	✅	✅	✅	10M+
REGISTRY-05	Registry Type Status Must Match Set Value	✅	✅	✅	10M+
REGISTRY-06	Owner Must Match After Transfer	✅	✅	✅	10M+
RWD-FLD-01	Claim Recipient Must Be Node Address	✅	✅	✅	10M+
RWD-FLD-02	Claim Cumulative Amount Must Match Params	✅	✅	✅	10M+
RWD-FLD-03	Claim Position ID Must Match Params	✅	✅	✅	10M+
RWD-FLD-04	Claim Cycle Must Match Params	✅	✅	✅	10M+
RWD-FLD-05	Claim Proof Hash Must Match Params	✅	✅	✅	10M+
RWD-INC-01	Last Earner Must Be Node Address	✅	✅	✅	10M+
RWD-INC-02	Campaign Addresses Hash Must Match	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
RWD-INC-03	Rewards Hash Must Match	✅	✅	✅	10M+
RWD-MKL-01	Users Hash Must Match Params	✅	✅	✅	10M+
RWD-MKL-02	Tokens Hash Must Match Params	✅	✅	✅	10M+
RWD-MKL-03	Amounts Hash Must Match Params	✅	✅	✅	10M+
RWD-MKL-04	Proofs Hash Must Match Params	✅	✅	✅	10M+
ROUTER-01	Blacklist Status Must Match Set Value	✅	✅	✅	10M+
ROUTER-02	Whitelist Status Must Match Set Value	✅	✅	✅	10M+
ROUTER-03	Tolerance Value Must Match Set Value	✅	✅	✅	10M+
ROUTER4626-01	Invest Must Return Non-Zero Deposit Amount	✅	✅	✅	10M+
ROUTER4626-02	Node Component Shares Must Not Decrease After Invest	✅	✅	✅	10M+
ROUTER4626-03	Node Asset Balance Must Not Increase After Invest	✅	✅	✅	10M+
ROUTER4626-04	Liquidate Must Return Non-Zero Assets When Expected	✅	✅	✅	10M+
ROUTER4626-05	Node Component Shares Must Not Increase After Liquidate	✅	✅	✅	10M+
ROUTER4626-06	Node Asset Balance Must Not Decrease After Liquidate	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
ROUTER4626-07	Fulfill Must Return Non-Zero Assets	✅	✅	✅	10M+
ROUTER4626-08	Escrow Balance Must Not Decrease After Fulfill	✅	✅	✅	10M+
ROUTER4626-09	Node Asset Balance Must Not Increase After Fulfill	✅	✅	✅	10M+
ROUTER7540-01	Invest Must Request Non-Zero Assets	✅	✅	✅	10M+
ROUTER7540-02	Pending Deposit Must Not Decrease After Invest	✅	✅	✅	10M+
ROUTER7540-03	Node Asset Balance Must Not Increase After Invest	✅	✅	✅	10M+
ROUTER7540-04	Node Component Shares Must Increase By Received Shares After Mint	✅	✅	✅	10M+
ROUTER7540-05	Claimable Must Not Increase After Mint	✅	✅	✅	10M+
ROUTER7540-06	Pending Redeem Must Not Decrease After Request Withdrawal	✅	✅	✅	10M+
ROUTER7540-07	Component Share Balance Must Not Increase After Request Withdrawal	✅	✅	✅	10M+
ROUTER7540-08	Execute Withdrawal Must Return Non-Zero Assets	✅	✅	✅	10M+
ROUTER7540-09	Execute Withdrawal Assets Must Match Max Withdraw Before	✅	✅	✅	10M+
ROUTER7540-10	Claimable Must Not Increase After Execute Withdrawal	✅	✅	✅	10M+
ROUTER7540-11	Node Asset Balance Must Not Decrease After Execute Withdrawal	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
ROUTER7540-12	Max Withdraw Must Be Zero After Execute Withdrawal	✅	✅	✅	10M+
ROUTER7540-13	Fulfill Redeem Must Return Non-Zero Assets When Expected	✅	✅	✅	10M+
ROUTER7540-14	Escrow Balance Must Not Decrease After Fulfill Redeem	✅	✅	✅	10M+
ROUTER7540-15	Node Asset Balance Must Not Increase After Fulfill Redeem	✅	✅	✅	10M+
ROUTER7540-16	Component Shares Must Not Increase After Fulfill Redeem	✅	✅	✅	10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	Nodes Cannot Be Used In Defi Protocols	Compatibility	High	Resolved
H-02	Locked Tokens In Adapter Due To Duplicate Hash	DoS	High	Resolved
M-01	Wrong requestRedeem Authorization Per Spec	Compatibility	Medium	Resolved
M-02	Wrong maxClaimableAssets Can DoS Fulfillments	DoS	Medium	Resolved
M-03	Rebalance Deadlock Due To Management Fee	DoS	Medium	Resolved
M-04	User Can Claim Others' Queued ERC7540	DoS	Medium	Acknowledged
M-05	Owner Can Drain All Funds Via rescueTokens	Trust Assumptions	Medium	Resolved
M-06	Whitelist Bypass Via EIP7702	Gaming	Medium	Resolved
M-07	Cannot Support Hybrid Asynchronicity	DoS	Medium	Acknowledged
M-08	Wrong ERC-7540 Claimable Share Valuation	Logical Error	Medium	Resolved
M-09	Owner Can Drain User Funds Via Swing Pricing	Trust Assumptions	Medium	Resolved
M-10	Users May Be Over-penalized For Withdrawals	Logical Error	Medium	Resolved
M-11	Split Withdrawals To Reduce Penalty	Math	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-12	Redeem Asset Impacted	Math	Medium	Resolved
M-13	Incorrect reserveImpact	Math	Medium	Resolved
M-14	Share Price May Become Inflated	Rounding	Medium	Resolved
L-01	Missing Slippage Control	Validation	Low	Acknowledged
L-02	Users May Pay More Management Fee	Logical Error	Low	Acknowledged
L-03	Fee Calculation Oversight	Logical Error	Low	Acknowledged
L-04	Indexed Dynamic Arrays In Events Not Supported	Events	Low	Resolved
L-05	Whitelist And Blacklist Flags Can Conflict	Logical Error	Low	Acknowledged
L-06	Owner Updates Should Effect In Future	Unexpected Behavior	Low	Acknowledged
L-07	Malicious Rebalancer Can Steal Incentive Tokens	Trust Assumptions	Low	Acknowledged
L-08	Same Price Deviation For Different Assets	Oracle	Low	Resolved
L-09	Manager Can Mis-assign Funds	Censoring	Low	Acknowledged
L-10	Overestimated ComponentAssets For Fee Vault	Logical Error	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-11	Liquidation Queue Can Have Inactive Component	Error	Low	Resolved
L-12	Liquidation Queue Incorrect Ordering	Logical Error	Low	Resolved
L-13	Insufficient Minimum Threshold Check	Logical Error	Low	Partially Resolved
L-14	Partial Withdrawals Are Blocked	DoS	Low	Acknowledged
L-15	Malicious Owner Can Lock Users' Assets	DoS	Low	Acknowledged
L-16	Node Owner Can Steal Funds	Censoring	Low	Acknowledged
L-17	Settlements May Use Incorrect Prices	Warning	Low	Acknowledged
L-18	Users Can Claim Incentra Rewards Directly	Logical Error	Low	Acknowledged
L-19	Griefing Rebalancers Via Dust Requests	Censoring	Low	Acknowledged
L-20	Inaccurate Calculation Of Shares	Logical Error	Low	Acknowledged
L-21	Unbounded Settlement Loops	Gas Griefing	Low	Acknowledged
L-22	Missing cacheTotalAssets Update After Swap	Logical Error	Low	Acknowledged
L-23	Can Invest In Blacklisted Components	Warning	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-24	Unchecked Deviation Upper Bound	Validation	Low	Resolved
L-25	Fee Bypass Via Low Decimal Tokens	Rounding	Low	Acknowledged
L-26	Mint And Withdraw Returns Incorrect Values	Compatibility	Low	Resolved
L-27	Inaccurate Liquidation Queue Enforcement	Logical Error	Low	Resolved
I-01	finalizeRedemption Lacks Rebalancing Modifier	Logical Error	Info	Resolved
I-02	Possible Different Share Price Via Deposit/Mint	Informational	Info	Acknowledged
I-03	rescueTokens Should Restrict Incentive Tokens	Trust Assumptions	Info	Acknowledged
I-04	Inconsistent Liquidation Ordering	Unexpected Behavior	Info	Resolved
I-05	totalAssets() Can Revert Violating ERC7575	Compatibility	Info	Acknowledged
I-06	Division By 0 DoS During Deposit Bonus	Math	Info	Resolved
I-07	Unbounded Loops In Event Verification	Gas Optimization	Info	Resolved
I-08	Unused Named Return Value	Informational	Info	Resolved
I-09	Warning About Virtual Functions	Warning	Info	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
I-10	_safeApprove Can Revert	Informational	Info	Acknowledged
I-11	Users May Be Unable To Withdraw From Adapter	Documentation	Info	Acknowledged
I-12	Unused State Variables In QuoterV1	Informational	Info	Resolved
I-13	Missing Public Functions In Multiple Interfaces	Informational	Info	Acknowledged
I-14	Misleading Comment Regarding deltaClosedPct	Informational	Info	Resolved
I-15	Incorrect Comment In DigiftEventVerifier	Informational	Info	Resolved
I-16	Warning Regarding Dust Accumulation In Digift	Warning	Info	Acknowledged
I-17	Zero Share Minting Allowed	Warning	Info	Resolved

H-01 | Nodes Cannot Be Used In Defi Protocols

Category	Severity	Location	Status
Compatibility	High	Node.sol: 621-638	Resolved

Description PoC

The `transfer`, `transferFrom`, and `approve` functions in the Node contract are supposed to return boolean values.

However, these overridden functions do not explicitly return any value and always return false, even though `super.transfer/super.transferFrom` executes successfully.

As a result, Nodes can never be used in DeFi protocols that utilize safe libraries like Solmate's `SafeTransferLib` or OpenZeppelin's `SafeERC20`, which breaks a core feature of Nodes.

Recommendation

Return true in these functions if they execute successfully.

Resolution

Nashpoint Team: The issue was resolved in commit 48b7373.

H-02 | Locked Tokens In Adapter Due To Duplicate Hash

Category	Severity	Location	Status
DoS	High	DigiftEventVerifier.sol: 261-264	Resolved

Description PoC

The SubRedManagement contract enables multi-token deposits from investors in exchange for security tokens stToken. The Nashpoint protocol handles this via the beacon proxy pattern, deploying different instances of DigiftAdapter contracts.

Once DigiftAdapter request a deposit or redemption, an admin from the SubRedManagement contract finalizes the process by calling settleSubscriber or settleRedemption. At this point, a settleSubscriber or settleRedemption event is emitted, which the Manager role from Nashpoint verifies through DigiftEventVerifier.verifySettlementEvent function.

The event verifier prevents double spending by hashing the blockHash, receiptsRoot, transaction index, log index and reverting if the hash was already used.

The issue is that multiple DigiftAdapter contracts can have the same used hash logged for a valid transaction if they are included in the investorList of the same event. However, since the hash has already been used, this will completely DoS the event verification for the latter adapters, resulting in locked funds.

Consider the following example:

Two DigiftAdapter contracts exist: Adapter A and Adapter B. requestDeposit is executed on both adapters for 20,000 USD each. Admin forwards this request forwardRequestsToDigift to SubRedManagement.

SubRedManagement finalizes this request via settleSubscriber, emitting the SettleSubscriber event which includes both Adapter A and Adapter B investment and share token quantities.

To finalize this settlement, the Manager starts with calling settleDeposit on Adapter A, providing the offchain and onchain parameters to verify that the SettleSubscriber event was emitted to include Adapter A in the list of investors and share tokens minted.

The event is then verified, vars.logHash = _hashLog(vars.blockHash, vars.receiptsRoot, fargs.txIndex, i); is calculated, and usedLogs[vars.logHash] = true; is updated.

Next, the Manager calls settleDeposit on Adapter B. Since Adapter B's subscription event was grouped with Adapter A's subscription event, it will contain the same block hash, receipt root, transaction index, and log index i. This means that it will produce the exact same log hash that was calculated for Adapter A, causing revert LogAlreadyUsed();.

Thus, the event cannot be verified, and funds are subsequently locked in the SubRedManagement contract.

Recommendation

Include the DigiftAdapter addresses in the hash to ensure adapter uniqueness.

Resolution

Nashpoint Team: The issue was resolved in commit db926f6.

M-01 | Wrong requestRedeem Authorization Per Spec

Category	Severity	Location	Status
Compatibility	Medium	Node.sol: 792-799	Resolved

Description

The current `_validateOwner` implementation incorrectly requires operators to have `ERC-20` approval and rejects approved spenders who aren't operators when `requestRedeem`, violating `ERC-7540`.

Per the EIP-7540 spec, operators should be able to execute redemptions without allowance restrictions, while approved spenders (non-operators) should be able to request redemptions within their approved share limit.

Recommendation

Spend allowance only when the spender is not operator.

```
function _validateOwner(address owner, uint256 shares) internal {
if (owner != msg.sender && !isOperator[owner][msg.sender]) {
_spendAllowance(owner, msg.sender, shares);
}
```

Resolution

Nashpoint Team: The issue was resolved in commit 7ff2c88.

M-02 | Wrong maxClaimableAssets Can DoS Fulfillments

Category	Severity	Location	Status
DoS	Medium	ERC7540Router.sol: 69	Resolved

Description

In `erc7540Router.fulfillRedeemRequest`, when no component can fully cover `assetsRequested`, the router enters the partial path and computes:

`claimableShares = claimableRedeemRequest(0, node)`

`maxClaimableAssets = convertToAssets(claimableShares)`

`convertToAssets` can return an inflated amount (cause it calculates the asset with current shares exchange rate, which will be higher than the value of shares at settlement often) vs the actual withdrawable assets (`maxWithdraw(node)`), so `min(assetsRequested, maxClaimableAssets)` will exceed `maxWithdraw(node)`, causing the withdrawal to revert when calling `components.withdraw` and blocking fulfillment.

Recommendation

Use `maxWithdraw` instead of `convertToAssets(claimableShares)`

Resolution

Nashpoint Team: The issue was resolved in commit 2c458ec.

M-03 | Rebalance Deadlock Due To Management Fee

Category	Severity	Location	Status
DoS	Medium	Node.sol: 339	Resolved

Description

`startRebalance` always attempts to pay accrued management fees from the Node's on-hand reserve before opening the rebalance window.

There isn't a balance or reserve ratio enforcement in Nodes, and some Nodes may have all of their balance invested in other components and have zero reserves.

If the Node's balance is less than the computed `feeForPeriod`, `_payManagementFees()` and `startRebalance` reverts, preventing the rebalance window from ever opening.

Since routers can only move funds during the rebalance window, operators cannot free up reserve to pay the fee, causing a deadlock where actions dependent on rebalancing are DoS'd.

Recommendation

Consider setting a minimum reserve ratio or buffer amount that cannot be invested in components, ensuring sufficient funds are available to cover fees.

Alternatively, consider a mechanism that pulls from components to cover fees when the balance is insufficient (e.g. atomically liquidate remainder during `startRebalance`).

Resolution

Nashpoint Team: The issue was resolved in commit 5715448.

M-04 | User Can Claim Others' Queued ERC7540

Category	Severity	Location	Status
DoS	Medium	ERC7540Router.sol: 61-62	Acknowledged

Description

Once users request withdrawals from the Node contract, the Rebalancer role can proceed to request a withdrawal from the ERC7540 vault (via ERC7540Router.requestAsyncWithdrawal), if an ERC7540 vault is used for investments by the Node.

After the request is finalized by the external ERC7540 vault contract (which can take days), the Rebalancer role can proceed to fulfill the redemption request of users via ERC7540Router::fulfillRedeemRequest.

The problem is that a user can timely execute a withdrawal request just before the Rebalancer role calls ERC7540Router.fulfillRedeemRequest, taking up other users' requested redemption for that rebalancing window, causing another withdrawal delay for users which can be at least another 1-2 days, or more.

Consider the following example:
• Bob and Alice both request a withdrawal from the Node contract, each requesting 100 shares to withdraw.
• Rebalancer role initiates a rebalance via startRebalance, then proceeds to call ERC7540Router.requestAsyncWithdrawal with 200 shares specified.
• The component takes 48 hours to finalize the request.
• Bob decides instead of withdrawing only 100 shares, he wants to withdraw a full 200 shares. Rather than waiting for the Rebalancer to call ERC7540Router.fulfillRedeemRequest followed by another 100 shares withdrawal request, he realizes Alice has already queued 100 shares.
• Bob takes advantage and timely requests another withdrawal via Node.requestRedeem specifying another 100 shares just before the next rebalancing window (as the rebalancing time is public).
• Now when the rebalancing window is initiated and Rebalancer calls ERC7540Router.fulfillRedeemRequest with Bob's address, Bob will receive the full 200 shares that was queued for withdrawal, because now his total assets requested is updated.
• Therefore, Alice will receive nothing.

Alice now has to wait for another ERC7540Router.requestAsyncWithdrawal call followed by ERC7540Router.fulfillRedeemRequest for her 100 share withdrawal, which can take days or longer.

Recommendation

A potential solution could be to separate withdrawal requests into epochs denoted by an epochId, so if any user attempts to request a withdrawal last second, it will reflect the withdrawal request for the next epoch rather than the current one.

Resolution

Nashpoint Team: Acknowledged.

M-05 | Owner Can Drain All Funds Via rescueTokens

Category	Severity	Location	Status
Trust Assumptions	Medium	Node.sol: 321-326	Resolved

Description

rescueTokens() allows the owner to recover accidentally sent tokens, blocking only the underlying asset and component share tokens. The implementation assumes that the share address of any component is the component address itself:

```
function rescueTokens(address token, address recipient, uint256 amount) external onlyOwner {
if (token == asset) revert ErrorsLib.InvalidToken();
>>   if (_isComponent(token)) revert ErrorsLib.InvalidToken();
IERC20(token).safeTransfer(recipient, amount);
emit EventsLib.RescueTokens(token, recipient, amount);
}
```

However, for ERC7540 components, the share token of a component (obtained via IERC7575(component).share()) can differ from the component address itself, as seen in ERC7540Router._requestRedeem(), so the owner can withdraw shares for any component that has a share token different from its address.

Example how owner can drain all users funds:
• Owner whitelists an ERC7540 component where share() != component
• Sets allocation to 100% and invests all node funds into that component
• Calls rescueTokens(shareToken, owner, balance) to sweep all share tokens (bypasses the _isComponent check)
• All user funds are drained

Recommendation

Add a loop in rescueTokens() to also block IERC7575(components[i]).share() for each component , you may consider doing it via try catch to avoid reverts from components that doesn't have share() function

Resolution

Nashpoint Team: The issue was resolved in commit d54c056.

M-06 | Whitelist Bypass Via EIP7702

Category	Severity	Location	Status
Gaming	Medium	GatePolicy.sol: 22	Resolved

Description

With EIP-7702, a whitelisted user can set their code to allow non-whitelisted users to interact with the protocol, effectively bypassing the GatePolicy.

Consider this scenario:

• Bob is whitelisted and Alice is not.

• Bob sets his account to interact with a permissioned Node like a router.

• Bob's account takes Alice's assets, deposits to Node, and transfers shares to Alice.

• Since the msg.sender in this case is Bob, whitelist checks pass and any non-whitelisted user can interact with Nodes.

Note that this scenario does not work with TransferPolicy, as the funds must ultimately be transferred to or from a non-whitelisted user, which the policy prevents.

Recommendation

One option is to recommend using TransferPolicy and GatePolicy concurrently for all node owners, as GatePolicy alone does not prevent end-to-end interaction.

Another option would be to use tx.origin as well along with msg.sender for the whitelist check in GatePolicy. However, this may be overly restrictive or introduce new issues, especially if some node owners want to implement custom routers that are whitelisted for their own nodes.

Alternatively, consider tracking users off-chain and removing from the whitelist those who allow others via EIP-7702.

Resolution

Nashpoint Team: The issue was resolved in commit 37f402f.

M-07 | Cannot Support Hybrid Asynchronicity

Category	Severity	Location	Status
DoS	Medium	ERC7540Router.sol: 283-294	Acknowledged

Description

The `_getErc7540Assets` function in the `ERC7540Router` contract assumes that all `ERC7540` components are fully asynchronous in both deposits and redemptions, and directly calls the `pendingRedeemRequest`, `claimableRedeemRequest`, `pendingDepositRequest`, and `claimableDepositRequest` functions.

However, according to the `ERC7540` specifications, implementations can choose whether to include asynchronous flows for deposits, redemptions, or both.

For example, the Node itself is `ERC7540-compatible` but does not implement the `pendingDepositRequest` or `claimableDepositRequest` functions.

Directly calling all four of these functions in `_getErc7540Assets` could cause a DoS when the component is partially asynchronous, as is the case with the Node.

Recommendation

Use `try/catch` when calling `ERC7540` components in the `ERC7540Router`. Alternatively, ensure all components supported by the protocol are fully asynchronous and never include hybrid components.

Resolution

Nashpoint Team: Acknowledged.

M-08 | Wrong ERC-7540 Claimable Share Valuation

Category	Severity	Location	Status
Logical Error	Medium	ERC7540Router.sol: 285-289	Resolved

Description

_getErc7540Assets() values claimable redeem shares at the current share price via convertToAssets(), but claimable shares represent a locked-in redemption value from when the request was processed. From erc7540 spec:

"The assets that will be received on redeem or withdraw MAY NOT be equivalent to the value of convertToAssets(shares) at the time of Request, as the price can change between Pending and Claimed"

Example:
• Node requests redeem of 100 shares when share price = 1.0 → locked value = 100 assets
• Request processed and becomes claimable (100 shares → 100 assets)
• Component NAV increases → new share price = 1.2
• _getErc7540Assets computes: convertToAssets(100 claimable shares) = 120 assets
• Actual claimable via maxWithdraw(): 100 assets
• Overvaluation: 20 assets (20%)

This can inflate or reduce the actual totalAsset, leading to incorrect share valuation and erroneous node share calculations.

Recommendation

Consider using maxWithdraw() for claimable shares. However, there is a tradeoff here and keep in mind that it may underestimate assets if vault has some withdrawal restrictions (e.g., withdraw cap per transaction, paused vault returns 0).

Resolution

Nashpoint Team: The issue was resolved in commit 91db03d.

M-09 | Owner Can Drain User Funds Via Swing Pricing

Category	Severity	Location	Status
Trust Assumptions	Medium	Node.sol: 25	Resolved

Description PoC

Owner can drain all users funds by exploiting the bonus mechanism , and the fact that he can switch swing pricing on and off at will with other important params.

Attack Steps:
1. Set `rebalanceCooldown = 0` and `rebalanceWindow = 1 sec` (no lower bounds enforced) : this to faster the attack
2. Invest all reserves to vault (0% reserve ratio)
3. Enable max swing pricing (99%) and set extreme reserve target (90%)
4. Owner's secondary address deposits to capture ~49.5% bonus shares (dilutes existing users)
5. Disable swing pricing to exit without penalty
6. Liquidate vault and redeem owner's shares at inflated value
7. Repeat each block until node is drained

Recommendation

You may consider queueing activation/deactivation of swing pricing to take effect in future to limit the owner power, so users can opt out and exit the system.

Additionally, consider having global lower bounds for `rebalanceCooldown` and `rebalanceWindow` to decrease the likelihood of such attack by malicious owner.

Resolution

Nashpoint Team: The issue was resolved in commit 6259916.

M-10 | Users May Be Over-penalized For Withdrawals

Category	Severity	Location	Status
Logical Error	Medium	QuoterV1.sol: 88-110	Resolved

Description

When swing pricing is enabled, users' withdrawal may face the penalty. According to https://nashpoint.gitbook.io/nashpoint/swing-pricing-calculations, when reserves fall below target, withdrawals receive progressively worse pricing.

The problem here is that if one withdrawal starts when the reserve ratio is larger than the target reserve ratio, and ends with the reserve ratio is less than the target reserve ratio, the whole withdrawal assets will be punished.

For example:
1. Current reserve cache ratio is 100%. Target reserve ratio is 10%.
2. Alice wants to request all assets, Then all assets will be punished with the final swing factor.
In fact, when Alice requests withdrawal for the first 90% asset, the node should not charge some punish fees.

When users deposit assets, users may get some bonus when the reserve ratio is below the target reserve ratio. Similar issue in deposit, if one deposit operation will raise the reserve ratio from one ratio below the target reserve ratio to one ratio above the target ratio, the system will return the min of deposit and asset shortfall to avoid overpaying the deposit bonus.

When users request withdrawal, one similar mechanism should be applied.

Recommendation

When one withdrawal request will drop the reserve ratio from the ratio which is above target ratio to the ratio which is below the target ratio, the punish amount should be calculated in the range from target reserve ratio to the actual reserve ratio after the withdrawal.

Resolution

Nashpoint Team: The issue was resolved in commit 6259916.

M-11 | Split Withdrawals To Reduce Penalty

Category	Severity	Location	Status
Math	Medium	QuoterV1.sol: 88-110	Resolved

Description PoC

When swing pricing is enabled, users' withdrawal may be punished when the reserves fall below target.

However, the penalty calculation is non-linear. When users split their withdrawal into several smaller transactions, they end up paying a lower total penalty than expected.

Recommendation

Refactor the penalty calculation.

Resolution

Nashpoint Team: The issue was resolved in commit 6259916.

M-12 | Redeem Asset Impacted

Category	Severity	Location	Status
Math	Medium	Node.sol: 436-459	Resolved

Description PoC

When swing pricing is enabled, users' redeem request may face some punishment when current reserve ratio is below than the target ratio.

When the rebalancer finalizes a redemption request, all related shares are burned, and the actual return asset is calculated based on `sharesAdjusted`. After the redemption is finalized, the share price increases slightly.

Therefore, the order of redemption finalizations is important. In this case, later redemption finalizations will receive a higher price. Because the order has importance, the rebalancer should finalize redemption requests in the order they were originally submitted.

The problem is that a single user can submit multiple redemption requests, and the Node contract merges all of them together. When the rebalancer attempts to fulfill redemptions for a controller, it becomes impossible to finalize these requests in their original submission order.

Recommendation

Do not merge a single user’s multiple redemption requests. The rebalancer should fulfill redemption requests in the order they were originally submitted.

Resolution

Nashpoint Team: The issue was resolved in commit 6259916.

M-13 | Incorrect reserveImpact

Category	Severity	Location	Status
Math	Medium	QuoterV1.sol: 105	Resolved

Description

The reserveImpact is computed using the pre-penalty asset value, but the penalty reduces the assets. Therefore, the actual reserveImpact after the redemption differs from the one used during the penalty calculation. As a result, users are always over-penalized.

```
//...
} else {
reserveImpact = int256(Math.mulDiv(reserveCash - assets, WAD, totalAssets - assets));
}
assets = Math.mulDiv(assets, (WAD - _getSwingFactor(reserveImpact, maxSwingFactor,
targetReserveRatio)), WAD); //@audit returned asset value is decreased.
```

For this calculation to be precise, the asset value after the penalty is applied should be the same value used when calculating the reserveImpact.

Recommendation

Consider assetsToReturn as the final value. We need to solve the equation where:

```
reserveImpact = int256(Math.mulDiv(reserveCash - assetsToReturn, WAD, totalAssets -
assetsToReturn));
assetsToReturn = Math.mulDiv(assets, (WAD - _getSwingFactor(reserveImpact, maxSwingFactor,
targetReserveRatio)), WAD);
```

Note that this introduces additional complexity, as the reserveImpact needed to compute the final value is itself unknown and circularly dependent on the final value.

If the additional complexity is undesirable and the current version will be used, this should be documented, as it creates unfair situations for high-value redemption requests.

Resolution

Nashpoint Team: The issue was resolved in commit 6259916.

M-14 | Share Price May Become Inflated

Category	Severity	Location	Status
Rounding	Medium	Node.sol: 746-770	Resolved

Description PoC

When the swing pricing is enabled, users' withdrawal operation may be punished, the protocol will burn all shares for this withdrawal and leave some assets in the node contract as one punishment.

If the last user requests to redeem all shares, and the rebalancer role will start rebalance and finalize the redeem. After the redeem finalization, `share` amount will be 0, and `cacheTotalAssets` will not be reduced to 0.

Based on current condition, if another user wants to deposit, the share's price will be very high(current share = 0, `cacheTotalAssets` might be larger than 1e18).

In Nashpoint, the impact of rounding (down or up) on calculations is typically around 1 wei when the share price is not inflated, which is acceptable. However, if the share price is inflated, such rounding can result in significant losses for users.

Recommendation

Consider enforcing Node owners to mint some dead shares upon Node deployment.

Resolution

Nashpoint Team: The issue was resolved in commit 6259916.

L-01 | Missing Slippage Control

Category	Severity	Location	Status
Validation	Low	Node.sol: 495	Acknowledged

Description

When swing price is enabled and the vault's liquidity reserve is less than the `targetReserveRatio`, users are incentivized to deposit via a deposit bonus, which mints users more shares than normal.

However, the bonus calculation uses various factors that can change before function execution such as cash after redemptions, `totalAssets()`, and owner configurable parameters such as `maxSwingFactor` and `targetReserveRatio`. This can cause users to receive a lower bonus than expected, or receive no bonus at all.

Slippage when exact output amount is not received for deposit/withdraw is also recommended under "Security Considerations" within EIP-4626 documentation.

Similarly, the `requestRedeem` function does not have slippage control, and users may encounter unexpected penalties when swing pricing is enabled.

Recommendation

Add slippage control for these functions.

Resolution

Nashpoint Team: Acknowledged.

L-02 | Users May Pay More Management Fee

Category	Severity	Location	Status
Logical Error	Low	Node.sol: 377-395	Acknowledged

Description

Management fees are paid at the beginning of each rebalance period after `cacheTotalAssets` are updated. For a default 24-hour rebalancing node, this uses the end-of-day asset value for the entire 24-hour period, regardless of when deposits are made during the day.

If a large deposit is made just a second before rebalancing, the system still charges fees as if the funds were managed for the entire 24-hour period.

While the current behavior is similar to fund management in traditional finance, and fees are regularly paid during rebalancing, there is no guarantee that rebalancing will occur frequently enough to make this discrepancy negligible.

Since nodes are permissionless and rebalancing periods can be set by their owners, the issue can be more pronounced, especially when nodes have longer rebalancing periods, such as weeks.

Recommendation

Ideally, every state change that updates `cacheTotalAssets` should charge fees based on the latest cached value and the period since the `lastPayment`. However, this would increase complexity.

If the end-of-day value will still be used for fee payments, document this behavior. Additionally, consider enforcing maximum bounds for rebalancing cooldowns or adding a function that allows a protocol-owned rebalancer to charge fees more frequently, reducing discrepancies even if the node has a longer rebalancing period.

Resolution

Nashpoint Team: Acknowledged.

L-03 | Fee Calculation Oversight

Category	Severity	Location	Status
Logical Error	Low	Node.sol: 307-311	Acknowledged

Description

The `setAnnualManagementFee` function updates the fee rate without first calculating and paying accrued fees for the period from `lastPayment` till now at the old rate. This results in miscalculated fees using the new rate for past periods.

A similar situation occurs when updating protocol management fee with `setProtocolManagementFee` function as well.

Recommendation

Pay management fees with previous fee percent and update the last payment before setting the new fee

Resolution

Nashpoint Team: Acknowledged.

L-04 | Indexed Dynamic Arrays In Events Not Supported

Category	Severity	Location	Status
Events	Low	EventsLib.sol: 121-123	Resolved

Description

`PoliciesAdded` and `PoliciesRemoved` events declare `bytes4[]` indexed sigs. However, Solidity does not support indexed dynamic array parameters in events. When indexed attribute is used in these types, hash of the value is stored in topics.

Reference: "A topic can only hold a single word (32 bytes) so if you use a reference type for an indexed argument, the `Keccak-256` hash of the value is stored as a topic instead."

As a result, these events are emitted with the hash of the sigs array, and might be unexpected for offchain listeners.

For example, the emitted event in the `test_addPolicies` test in the `Node.t.sol` file is:

`emit PoliciesAdded(sigs:0xfb5baaecab62c516763cea2dfba17fbbc24907e4e3b0be426bde71be89af495f, policies: [0x0000000000000000000000000000000000000012])`

Recommendation

Consider removing the indexed attribute from these events or emitting per-element events.

Resolution

Nashpoint Team: The issue was resolved in commit 500d450.

L-05 | Whitelist And Blacklist Flags Can Conflict

Category	Severity	Location	Status
Logical Error	Low	BaseComponentRouter.sol: 44-57	Acknowledged

Description

The admin setters in `BaseComponentRouter` allow a component to be both whitelisted and blacklisted simultaneously. This inconsistent state can cause policy bypass in downstream code that only checks one flag and makes the intended component status ambiguous.

Recommendation

Enforce mutual exclusivity in setters: when setting blacklist to true, force whitelist to false (and vice versa).

Resolution

Nashpoint Team: Acknowledged.

L-06 | Owner Updates Should Effect In Future

Category	Severity	Location	Status
Unexpected Behavior	Low	Node.sol: 278-287	Acknowledged

Description

The node owner can change the rebalance window during an active rebalance or adjust the cooldown period while in an active cooldown.

However, ideally, these changes should only affect future rebalance windows and cooldown periods, and the owner should not be able to increase or decrease the current period without notice.

Recommendation

Do not allow the owner to change the rebalance window or cooldown period when the Node is already in that particular state.

Resolution

Nashpoint Team: Acknowledged.

L-07 | Malicious Rebalancer Can Steal Incentive Tokens

Category	Severity	Location	Status
Trust Assumptions	Low	OneInchV6RouterV1.sol: 111-154	Acknowledged

Description

The `swap()` function in `OneInchV6RouterV1` allows the rebalancer to set `minAssetsOut` to any value, including zero, with no oracle price validation, this allow the rebalancer to steal all incentive tokens.

Example:

• Node has 1000 USDC worth of incentive tokens accumulated

• Malicious rebalancer manipulates swap `pool[asset-incentivetoken]` via flash loan

• Malicious rebalancer calls `swap()` with `minAssetsOut = 0`

• Swap executes at manipulated rate, node receives ~0 assets back

• Rebalancer back run to swap back , extract the whole incentive , and return the flashloan

• Loss: up to 100% of incentive value stolen

Recommendation

Either use whitelisted oracles to validate the `minAmountOut` , or maybe only owner should be able to do swap (since owner can sweep incentive tokens anyway , so i assume he's trusted on that)

Resolution

Nashpoint Team: Acknowledged.

L-08 | Same Price Deviation For Different Assets

Category	Severity	Location	Status
Oracle	Low	DigiftAdapter.sol: 594	Resolved

Description

The `DigiftAdapter` contract uses the same `priceUpdateDeviation` parameter for both the asset token and the Digift token.

However, these tokens may have different update frequencies, especially when real-world asset price updates are considered.

While it may be reasonable for Digift tokens to have an update deviation of over 24 hours, or even several days, such a deviation could render the asset token's price data completely outdated.

Recommendation

Consider using different deviation values for different tokens.

Resolution

Nashpoint Team: The issue was resolved in commit 1ff5440.

L-09 | Manager Can Mis-assign Funds

Category	Severity	Location	Status
Censoring	Low	DigiftAdapter.sol: 634-693	Acknowledged

Description

In `digiFTAdapter`, when `forwardRequestsToDigift` function is called, it snapshots `accumulatedDeposit` into `pendingDepositRequest` globally but does not lock or record which specific nodes contributed to that batch.

While a pending settlement exists, new nodes can still call `requestDeposit()`, which adds to `accumulatedDeposit` (separate from the pending amount).

At settlement via `settleDeposit(nodes, ...)`, the only check is that the sum of `node.pendingDepositRequest` across the provided `nodes[]` equals `globalPendingDepositRequest`. It does not verify that these are the same nodes whose deposits were actually forwarded. As a result, assets can be assigned to other nodes as long as the total sum remains equal.

Example scenario:
• Node1 deposits 100 tokens → manager forwards to DigiFT (pending = 100)
• Node2 deposits 60, Node3 deposits 40 (accumulated = 100, but pending still = 100 from Node1)
• Manager can settle the DigiFT response to Node2 and Node3 instead of Node1, because 60 + 40 = 100
• Node1 remains with pending status even though his batch was settled

Note that the same issue exist with redeem settlements.

Recommendation

Use an auto-incrementing batchId (bumped on each forward); on deposit, snapshot which batchId each node contributed to; at settlement, enforce that returns apply only to nodes tied to that exact batchId.

Resolution

Nashpoint Team: Acknowledged.

L-10 | Overestimated ComponentAssets For Fee Vault

Category	Severity	Location	Status
Logical Error	Low	ERC4626Router.sol: 242	Resolved

Description

The use of `convertToAssets` in `ERC4626Router.sol` can overestimate the actual asset value of a component's shares. Per `ERC-4626` spec, `convertToAssets` does not account for fees or slippage that would apply during a real redemption.

This can inflate the reported `totalAssets` in the Node, leading to incorrect share minting. As a result, users may redeem more assets than they should (causing insolvency), or receive fewer assets than they are entitled to.

Recommendation

Consider using `previewRedeem` instead or `convertToAssets`, as it's inclusive of fees per spec.

Resolution

Nashpoint Team: The issue was resolved in commit 2cd9444.

L-11 | Liquidation Queue Can Have Inactive Component

Category	Severity	Location	Status
Error	Low	Node.sol: 829	Resolved

Description

The protocol docs state that one of the requirements of the liquidation queue is that all addresses must be active components. This is correctly enforced when the Node owner calls `setLiquidationQueue`.

However, if the node owner decides to remove the component via `removeComponent` function, the deleted component is then not removed from the queue, thus breaking this requirement.

The impact is that, in addition to a broken protocol invariant, this will DoS liquidation orders in `_enforceLiquidationOrder`:

```
try IRouter(router).getComponentAssets(candidate, true) returns (uint256 assets) {
candidateAssets = assets;
}
```

`router` for the removed candidate will be `address(0)`, and since a return value is expected, this will not go to the `catch` block and instead entirely revert.

Recommendation

Upon removing a component, consider also removing it from the liquidation queue, or check if the owner cleared it from the queue first.

Resolution

Nashpoint Team: The issue was resolved in commit fc79a10.

L-12 | Liquidation Queue Incorrect Ordering

Category	Severity	Location	Status
Logical Error	Low	Node.sol: 824-841	Resolved

Description

The protocol allows Node Owners to configure liquidation ordering to prioritize components based on, for example, liquidity costs and withdrawal delays.

The following example is provided in the documentation:

```
For queue [A, B, C] where A is ERC7540 and B, C are ERC4626:
Must check A's claimable balance first
Can only use B if A's claimable balance insufficient
Can only use C if both A and B insufficient
Pending (non-claimable) balances in A don't block using B or C
```

There's a case where all of A,B,C may have insufficient claimable assets. In that case, the correct way should be to perform a partial (maximum) withdrawal on A first, followed by partial withdrawal on B, and then C.

However, a direct partial withdrawal on B without following the correct order can succeed with the current implementation.

For example, assume total claimable assets are A= 50tokenA, B = 30tokenA, and C = 30 tokenA. The requested withdrawal is 60 tokenA. The rebalancer role calls ERC4626Router.fulfillRedeemRequest for component B. This calls Node::enforceLiquidationOrder with component = B and 60 tokenA.

_enforceLiquidationOrder will loop through the entire queue list of A,B,C, and since they all hold insufficient tokens, the call succeeds and a partial withdrawal on component B is executed using the minimal balance int256 componentShares = Math.min(IERC4626(component).convertToShares(assetsRequested), IERC20(component).balanceOf(address(node)));.

This can be a common occurrence as investment portfolios are diversified across various components, and the intention of the Node Owner is to perform a partial withdrawal in the queued order.

Recommendation

Within _enforceLiquidationOrder, if each component has insufficient funds, ensure the component specified is equal to the candidate at the top of the queue. This will ensure partial withdrawals can happen in the correct order as the Node Owner intends.

Resolution

Nashpoint Team: The issue was resolved in commit fc79a10.

L-13 | Insufficient Minimum Threshold Check

Category	Severity	Location	Status
Logical Error	Low	BaseComponentRouter.sol: 123	Partially Resolved

Description

Node Owners can configure a `maxDelta` parameter for each allocated component, which is responsible for ensuring that the amount of deposited assets during investments are above this threshold. As the protocol docs state, this is to ensure unnecessary transactions are minimized.

The `maxDelta` parameter is enforced within the `BaseComponentRouter`:

```
// Validate deposit amount exceeds minimum threshold
if (depositAmount < Math.mulDiv(totalAssets,
INode(node).getComponentAllocation(component).maxDelta, WAD)) {
revert ErrorsLib.ComponentWithinTargetRange(node, component);
}
// limit deposit by reserve ratio requirements
// _validateReserveAboveTargetRatio() ensures currentCash >= idealCashReserve
depositAmount = Math.min(depositAmount, currentCash - idealCashReserve);
// subtract execution fee for protocol
depositAmount = _subtractExecutionFee(depositAmount, node);
```

Notice how the `depositAmount` is checked against the `maxDelta` threshold (to ensure it is not too small), but the `depositAmount` is subsequently updated to a potentially lower value in the next lines of code, followed by a fee deduction. This updated `depositAmount`, which is the actual deposit amount, is not checked against the minimum threshold.

The impact is that unnecessary/small investment transactions will continue to occur, breaking protocol invariant.

Recommendation

Check against the `minimum threshold` after updating the `depositAmount` and deducting fees

Resolution

Nashpoint Team: The issue was resolved in commit ae4a663.

L-14 | Partial Withdrawals Are Blocked

Category	Severity	Location	Status
DoS	Low	DigiftAdapter.sol: 849	Acknowledged

Description

The ERC7540router can requests a partial async withdrawal using Math.min(assetsRequested, maxClaimableAssets) in the fulfillRedeemRequest function:

```
assetsReturned = _executeAsyncWithdrawal(node, component, Math.min(assetsRequested,
maxClaimableAssets));
```

if the component is DigiftAdapter and the amount requested is assetRequested which is less than maxWithdraw the tx will always revert, as digiFT adapter enforces full-withdraw-only semantics and reverts unless the requested assets equals node.maxWithdraw exactly. Which will block a request fulfilment.

Recommendation

Consider relaxing the strict requirement in digiFTAdapter and allowing partial withdraw.

Resolution

Nashpoint Team: Acknowledged.

L-15 | Malicious Owner Can Lock Users' Assets

Category	Severity	Location	Status
DoS	Low	NodePausingPolicy.sol: 82-85	Acknowledged

Description

The Node Owner has the ability to block user withdrawals via the following methods:

1. Changing `setRebalanceCooldown` (or `setRebalanceWindow`) to extend cooldown so rebalancer cannot call `startRebalance()` to initiate redemption from components.

2. Implement incorrect component ratio causing `startRebalance()` to fail due to `validateComponentRatios()` returning false. Since rebalancing fails, withdrawals from components will also fail.

3. Utilize policies to, for example, pause the router/rebalancer address for the withdrawal selectors.

This is dangerous as the Node Owner is a permissionless role, thus causing harm to innocent users and trust for the protocol.

Recommendation

1. Set strict limits on how much the owner can change rebalance cooldown and window

2. Enforce component ratio validation upon adding a new component in `Node::addComponent`

3. Ensure that withdrawals cannot be blocked in policies

Alternatively, document these behaviors for users to inform them of the owner’s capabilities.

Resolution

Nashpoint Team: Acknowledged.

L-16 | Node Owner Can Steal Funds

Category	Severity	Location	Status
Censoring	Low	Node.sol: 192-205	Acknowledged

Description

When there is something wrong in one component, the protocol owner will set this component into the blacklist. The node owner is expected to withdraw assets from this component and then remove this component via `removeComponent`.

Normally, component tokens are not allowed to be withdrawn in `rescueTokens`. However, a malicious owner can remove a blacklisted component directly by force and then directly transfer these component tokens.

```
function removeComponent(address component, bool force) external onlyOwner
onlyWhenNotRebalancing {
if (force && !IRouter(router).isBlacklisted(component)) {
revert ErrorsLib.NotBlacklisted();
}
}
function rescueTokens(address token, address recipient, uint256 amount) external onlyOwner {
if (token == asset) revert ErrorsLib.InvalidToken();
if (_isComponent(token)) revert ErrorsLib.InvalidToken();
IERC20(token).safeTransfer(recipient, amount);
emit EventsLib.RescueTokens(token, recipient, amount);
}
```

Recommendation

Consider documenting this behavior. Alternatively, do not allow blacklisted component shares to be rescued either, which forces the owner to withdraw the underlying funds through the regular liquidation flow.

Resolution

Nashpoint Team: Acknowledged.

L-17 | Settlements May Use Incorrect Prices

Category	Severity	Location	Status
Warning	Low	DigiftAdapter.sol: 649	Acknowledged

Description

The `settleDeposit` and `settleRedeem` functions in the `digiftAdapter` contract use the latest oracle prices to determine the settlement value.

While the expected time between the actual settlement event on the DigiFT side and the settlement on the adapter side can be up to 256 blocks, the latest price at the time of adapter settlement may differ from the price at the time of DigiFT settlement.

This discrepancy can become more pronounced if a time gap occurs due to unexpected downtime in the backend or event listeners, requiring manual intervention after 256 blocks. In this case, the real settlement value and the calculated settlement value may differ significantly.

Recommendation

Be aware of this situation. Alternatively, consider using the valid price at the time the event is emitted on DigiFT to determine the settlement value.

Resolution

Nashpoint Team: Acknowledged.

L-18 | Users Can Claim Incentra Rewards Directly

Category	Severity	Location	Status
Logical Error	Low	IncentraRouter.sol: 28-37	Acknowledged

Description

In IncentraRouter, the rebalancer can claim some rewards from Incentra on behalf of the node.

The Incentra distributor on Arb is deployed on:

https://arbiscan.io/address/0x273d0d19eaC2861FCF6B21893AD6d71b018E25aB#code.

Users can claim node's rewards on behalf of the node directly via below function.

```
function claimAll(address earner, address[] calldata campaignAddrs) public {
for (uint256 i = 0; i < campaignAddrs.length; i++) {
IRewardContract(campaignAddrs[i]).claim(earner);
}
}
```

In the current design, only the node rebalancer is allowed to claim rewards. However, regular users can claim rewards directly on behalf of the node, giving them a way to increase the asset amount directly.

e.g.
1. Alice deposits asset with current cache price.
2. Alice claims Incentra rewards via function claim. Although the current share price does not change, this ensures that the share price will be affected after the next rebalance.
3. After the next rebalance, the cache price will be updated, and the user can request withdraw with new price.

If the rebalancers claim these rewards frequently in every rebalancing period, the impact would be minimal. However, if rebalancers do not claim rewards, users could anticipate a larger change in share value in the future and deposit before the value increases.

Recommendation

Rebalancers should claim rewards as frequently as possible to minimize the impact of claimed rewards on the share value, since there is no way to prevent regular users from claiming on behalf of the Node.

Resolution

Nashpoint Team: Acknowledged.

L-19 | Griefing Rebalancers Via Dust Requests

Category	Severity	Location	Status
Censoring	Low	Node.sol: 451	Acknowledged

Description

Share owners create redemption requests, which are expected to be fulfilled by the rebalancer during the next rebalancing period. By default, rebalancers have one hour to perform multiple tasks, such as claiming rewards and fulfilling redemption requests.

Share owners can create redemption requests for any amount and for any controller, and these requests are stored per controller.

It is possible to create thousands of small redemption requests across different controller addresses, which increases the rebalancer's workload in the next rebalancing period.

This not only results in griefing the rebalancer, but can also cause legitimate users' redemption requests to be delayed for several rebalancing periods.

Recommendation

Consider enforcing a minimum request amount to prevent such griefing.

Additionally, consider disabling the creation of requests for arbitrary controllers. If this feature is necessary at the node level, the restriction can be implemented through a custom policy, allowing node owners to maintain greater control.

Resolution

Nashpoint Team: Acknowledged.

L-20 | Inaccurate Calculation Of Shares

Category	Severity	Location	Status
Logical Error	Low	Node.sol: 495	Acknowledged

Description

`totalAssets()` uses `cacheTotalAssets` (refreshed at `startRebalance`) to price deposits/mints. Deposits are allowed while the cache can be stale, so `convertToShares` often uses an outdated denominator to calculate the number of shares to mint.

Depositors can be over-minted when current actual `totalAssets` is greater than `cacheTotalAssets` (yield accumulated from `lastUpdate` until now), or under-minted when current actual `totalAssets` is less than `cacheTotalAssets` (strategy loss from last `cacheTotalAssets` update until now).
Example:

Cache = 1,000,000; actual `totalAssets` with accumulated yield from all components = 1,050,000;

• User deposits 100,000 → receives 100,000 shares

• Shortly after, rebalancer calls `startRebalance`, cache gets updated to 1,150,000

• User shares worth: 100,000 * 1,150,000 / 1,100,000 = 104,545, a 4.5k gain extracted from existing holders without actually contributing to the yield

• This gain for the depositing user is a loss for others who have been staking their assets in the node

• Same applies for strategy losses

• if total assets are frequently updated by rebalancer, impact will minimal

Recommendation

Consider refreshing `totalAsset` before any deposit.

Resolution

Nashpoint Team: Acknowledged.

L-21 | Unbounded Settlement Loops

Category	Severity	Location	Status
Gas Griefing	Low	DigiftAdapter.sol: 634-693	Acknowledged

Description

The DigiftAdapter supports multiple nodes depositing/redeeming in the same batch. When settlement occurs via settleDeposit(nodes, ...) or settleRedeem(nodes, ...), the manager is enforced to provide all contributor nodes to that batch and the function loops through all of them to distribute shares/assets proportionally.

There is no cap on how many nodes can contribute to a single batch. If too many nodes deposit/redeem for the same batch , the settlement transaction could exceed block gas limits and revert, making it impossible to settle and permanently locking all pending funds, unless upgrade

Recommendation

Enforce a maximum number of nodes that can be whitelisted per adapter instance (depending on gas analysis).

Resolution

Nashpoint Team: Acknowledged.

L-22 | Missing cacheTotalAssets Update After Swap

Category	Severity	Location	Status
Logical Error	Low	OneInchV6RouterV1.sol: 111-155	Acknowledged

Description

In Nashpoint, the Node may gain some rewards. The rebalancer can swap these incentive tokens into node asset in the rebalancing window.

One normal rebalancing order may be like as below:
1. Start rebalance.
2. Claim rewards.
3. Swap rewards to node asset token.
4. Fulfill redemptions.
5. Invest into vaults.

The problem here is that when the incentive token is a node asset, or when other incentive tokens are swapped into a node asset, the Node does not update the claimed or swapped node assets in `cacheTotalAssets`.

Then, when the rebalancer fulfills a redemption, the claimed rewards will not be included in the asset calculation, which is unfair to the user who redeemed their shares, as they are also entitled to a portion of these earned incentives. These assets will only be considered in the next rebalancing period.

Recommendation

When the rebalancer claims node assets or swaps other incentive tokens to acquire node assets, these assets should be added to `cacheTotalAssets`. If this is the intended behavior, document this for users.

Resolution

Nashpoint Team: Acknowledged.

L-23 | Can Invest In Blacklisted Components

Category	Severity	Location	Status
Warning	Low	Global	Resolved

Description

The whitelist and blacklist status of a component is only checked when adding or removing a component from a Node. There is no check for the blacklist status when investing in components.

A blacklisted component must be removed from the Node. However, until it is removed, it is still considered a valid Node component.

The rebalancer might unintentionally continue investing in blacklisted components without noticing that a component has been blacklisted by the registry owner, especially if the rebalancing process is automated.

Recommendation

Deposits into blacklisted components should be prevented. This can be achieved by checking the blacklist status in the `_computeDepositAmount` function of the `BaseComponentRouter`, which is used by all investment flows.

Resolution

Nashpoint Team: The issue was resolved in commit 65ac649.

L-24 | Unchecked Deviation Upper Bound

Category	Severity	Location	Status
Validation	Low	DigiftAdapter.sol: 402-403	Resolved

Description

The `withinRange` function in `MathLib` reverts with an underflow error when the `allowedDeviation` value exceeds WAD. The `allowedDeviation` values passed to this function are the `priceDeviation` and `settlementDeviation` values from the `DigiftAdapter` contract.

While the setter functions for these values enforce that the value is less than or equal to WAD, there is no such check during initialization, so these values can exceed WAD, causing reverts during `withinRange` call.

Recommendation

Ensure that `args.priceDeviation` and `args.settlementDeviation` in `initialize` are less than or equal to WAD.

Resolution

Nashpoint Team: The issue was resolved in commit 221274d.

L-25 | Fee Bypass Via Low Decimal Tokens

Category	Severity	Location	Status
Rounding	Low	BaseComponentRouter.sol: 186-195	Acknowledged

Description

The protocol earns fees via the following logic which is executed during investments and swaps:

`uint256 executionFee = transactionAmount * registry.protocolExecutionFee() / WAD;`

The `executionFee` rounds down due to Solidity's default built-in rounding down. With WAD math, a fee as low as 0.0001% would mean `registry.protocolExecutionFee() = 1e12`. Because `executionFee = floor(amount * 1e12 / 1e18)`, any `amount < 1e6` base units rounds the fee down to zero.

This is especially problematic for some high value tokens such as WBTC, where 1e6 of the token is currently worth ~$1110.

In addition, for higher decimal tokens, users can batch swap/investment transactions into small amounts to round down the `executionFee` to 0 due to a lower `transactionAmount`.

The investment/swap calls would proceed as normal since the execution fee function always returns when fee calculated is 0:

```
if (executionFee == 0) {
return transactionAmount;
}
```

Another option could be for users to create wrappers for high decimal tokens (i.e 18 decimals) and turn it into low decimal (i.e 2-6 decimals), allowing fee bypass.

The result is loss of funds/revenue for the protocol.

Recommendation

Consider rounding up the `executionFee` or using fractional protocol fee accounting.

Resolution

Nashpoint Team: Acknowledged.

L-26 | Mint And Withdraw Returns Incorrect Values

Category	Severity	Location	Status
Compatibility	Low	DigiftAdapter.sol: 845	Resolved

Description

DigiftAdapter.mint() and DigiftAdapter.withdraw() return values that don't match the actual assets consumed or shares burned, creating inconsistency with their emitted events and ERC-4626/7575 semantics.

• mint(): Returns claimableDepositRequest (gross assets) but actually consumes assets - assetsToReimburse (net assets). The Deposit event correctly emits the net amount, but the return value overstates consumption.

• withdraw(): Returns claimableRedeemRequest (gross shares) but actually burns shares - sharesToReimburse (net shares). The Withdraw event correctly emits the net amount, but the return value doesn't reflect what was burned. The NatSpec also states "returns (uint256 shares) The number of shares burned", which is wrong.

While the current ERC7540Router does not rely on these return values (it tracks via balance deltas), it is preferable to remain aligned with the specification for any future interactions.

Recommendation

Consider returning net amounts (after reimbursement) to match actual state changes and emitted events.

Resolution

Nashpoint Team: The issue was resolved in commit c8cca04.

L-27 | Inaccurate Liquidation Queue Enforcement

Category	Severity	Location	Status
Logical Error	Low	ERC4626Router.sol: 157-159	Resolved

Description

Liquidation order enforcement uses inflated component capacity

• `ERC4626Router.getComponentAssets()` ignores the `claimableOnly` parameter and always returns `convertToAssets(shareBalance)`, which does not account for withdrawal limits

• When Node's `_enforceLiquidationOrder()` calls `getComponentAssets(candidate, true)` it expects the actual claimable assets, not all the asset invested in a component, but in case of `ERC4626Router` it may receive a value that can be more than what actually can be withdrawn from the component (maxWithdraw), thus it may enforce a wrong liquidation queue .

Recommendation

Return `maxWithdraw(node)` when `claimableOnly == true` in `ERC4626Router.getComponentAssets()`

Resolution

Nashpoint Team: The issue was resolved in commit fc79a10.

I-01 | finalizeRedemption Lacks Rebalancing Modifier

Category	Severity	Location	Status
Logical Error	Info	Node.sol: 426	Resolved

Description

The rebalancer role has two ways of finalizing user requested redemptions:

1. `Node::fulfillRedeemFromReserve`: Utilizes the reserves within the Node contract to finalize redemptions.

2. `Router::fulfillRedeemRequest` (which calls `Node::finalizeRedemption`): Executes withdrawals from components, such as `ERC4626` vaults, to finalize redemptions.

The first method, `fulfillRedeemFromReserve` ensures that the `Rebalancer` role can only execute this after first calling `startRebalance()`. This ensures that the total assets are up to date via (`_updateTotalAssets`) and that fees are distributed (via `_payManagementFees`) using the current cached balance.

The problem is that the second option, `fulfillRedeemRequest`, does not have the requirement that the rebalancing window must have been initiated first (via `startRebalance()`). This means that when redemptions are finalized, an outdated `cacheTotalAssets` may be used, causing an incorrect share to asset calculation, thus causing a loss for users.

In addition, this flow will subsequently skip fee payment (to protocol and node owner). Since `finalizeRedemption` deducts the `cacheTotalAssets`, the fee payment for the withdrawal is skipped permanently, as the fees calculated depend on the `cacheTotalAssets` value. This will cause a loss to the protocol and node owner.

Recommendation

Apply the `onlyWhenRebalancing` modifier to `Node::finalizeRedemption`

Resolution

Nashpoint Team: The issue was resolved in commit e959571.

I-02 | Possible Different Share Price Via Deposit/Mint

Category	Severity	Location	Status
Informational	Info	Node.sol: 491-510	Acknowledged

Description

In Node, users can choose to deposit assets via deposit or mint. When swing pricing is enabled, users may receive a deposit bonus through deposit, but they will not receive a bonus through mint.

Recommendation

Document this behavior to inform users about the difference between deposit and mint.

Resolution

Nashpoint Team: Acknowledged.

I-03 | rescueTokens Should Restrict Incentive Tokens

Category	Severity	Location	Status
Trust Assumptions	Info	Node.sol: 321-326	Acknowledged

Description

The `rescueTokens` function can be used to withdraw tokens from the Node, except for the Node's asset token and component tokens. However, this restriction does not apply to incentive tokens claimed from external integrations.

These incentive tokens should also not be withdrawable by the Node owner, similar to the Node's asset tokens, as they belong to depositors and are intended to be swapped into asset tokens during rebalancing.

Recommendation

Restrict incentive tokens from being withdrawn via the `rescueTokens` function. However, this would require additional functionality to track all incentive token addresses when the Node joins a campaign.

Alternatively, document this behavior for users.

Resolution

Nashpoint Team: Acknowledged.

I-04 | Inconsistent Liquidation Ordering

Category	Severity	Location	Status
Unexpected Behavior	Info	ERC4626Router.sol: 89	Resolved

Description

The order of liquidation set by the Node Owner is correctly enforced within `ERC7540Router::fulfillRedeemRequest`, `ERC4626Router::fulfillRedeemRequest`. These functions execute requested withdrawals and update storage for user withdrawal requests.

However, the `Rebalancer` role can directly withdraw from the components via `ERC4626Router::liquidate` and `ERC7540Router::executeAsyncWithdrawal`, which do not check the liquidation order.

The impact is that withdrawals can happen in a complete different order than what the Node Owner specified, potentially withdrawing from more riskier vaults first.

Recommendation

Consider enforcing the liquidation order in `ERC4626Router::liquidate` and `ERC7540Router::executeAsyncWithdrawal`

Resolution

Nashpoint Team: The issue was resolved in commit fc79a10.

I-05 | totalAssets() Can Revert Violating ERC7575

Category	Severity	Location	Status
Compatibility	Info	DigiftAdapter.sol: 1012-1014	Acknowledged

Description

The `totalAssets()` function calls `convertToAssets(totalSupply())`, which internally fetches prices via `_getAssetPrice()` and `_getPrice()`. These price functions enforce staleness and deviation checks that can revert.

`ERC7575` spec requirement: "`totalAssets()` MUST NOT revert"

Recommendation

Make price fetches in view functions non-reverting by returning cached/last-known values when fresh data is unavailable, or clearly document that this implementation does not fully conform to `ERC7575` view function requirements

Resolution

Nashpoint Team: Acknowledged.

I-06 | Division By 0 DoS During Deposit Bonus

Category	Severity	Location	Status
Math	Info	QuoterV1.sol: 145	Resolved

Description

QuoterV1 executes the following when calculating deposit bonus:

```
uint256 targetReserveAssets = investedAssets.mulDiv(targetReserveRatio, WAD -
targetReserveRatio);
...
uint256 deltaClosedPct = Math.mulDiv(deltaClosed, WAD, targetReserveAssets);
```

In case the investedAssets is a low amount (i.e after liquidations) and for example targetReserveRatio is set to 5% (0.05e18), targetReserveAssets can round down to 0. This would cause a revert due to division by 0 when calculating deltaClosedPct during the deposit bonus calculation, causing DoS to deposits completely.

This edge case can be triggered if getCashAfterRedemptions is depleted due to large withdrawals causing (Math.mulDiv(getCashAfterRedemptions(), WAD, totalAssets()) < targetReserveRatio) thus executing the deposit bonus branch during deposits.

Recommendation

Return safely if targetReserveAssets == 0 when calculating deposit bonus

Resolution

Nashpoint Team: The issue was resolved in commit 2c6e5b9.

I-07 | Unbounded Loops In Event Verification

Category	Severity	Location	Status
Gas Optimization	Info	DigiftEventVerifier.sol: 223-279	Resolved

Description

`DigiftEventVerifier.verifySettlementEvent()` iterates through all logs in the transaction receipt to locate the settlement event, and then iterates through all investors in the event to find the caller's index.

A transaction can emit multiple logs, and a log can include many investors, which may result in excessive gas consumption or even hit block limits in some cases.

Recommendation

Add `logIndex` and `investorIndex` to `OffchainArgs` to allow direct access instead of O(n) scans, which will save a significant amount of gas.

Resolution

Nashpoint Team: The issue was resolved in commit a0ee7e7.

I-08 | Unused Named Return Value

Category	Severity	Location	Status
Informational	Info	ERC4626Router.sol: 194	Resolved

Description

The `_getInvestmentSize` function in the `ERC4626Router` and `ERC7540Router` contracts defines the return value as `uint256 depositAssets`. However, this `depositAssets` value is never used, and delta is returned instead.

Recommendation

Remove unused named value.

Resolution

Nashpoint Team: The issue was resolved in commit c1a5fda.

I-09 | Warning About Virtual Functions

Category	Severity	Location	Status
Warning	Info	BaseComponentRouter.sol: 96-103	Resolved

Description

Both `getComponentAssets` and `_getInvestmentSize` function in `BaseComponentRouter` contract are declared virtual but have empty bodies that implicitly return zero. These functions must be overridden in the inheriting router contract.

While this is not an issue at the moment, forgetting to override these functions when adding new routers causes calls to succeed but return 0, which can lead to unexpected behaviors such as incorrect valuations.

Recommendation

Consider making these virtual functions revert by default, which would prevent unexpected behaviors and enable early detection if a new router fails to override them.

Resolution

Nashpoint Team: The issue was resolved in commit 59402ff.

I-10 | _safeApprove Can Revert

Category	Severity	Location	Status
Informational	Info	BaseComponentRouter.sol: 197-200	Acknowledged

Description

The helper function approves a spender directly to a non-zero amount without first resetting the allowance to zero. Some tokens require that the allowance be set to zero before assigning a new non-zero allowance.

Since all flows in the codebase approve an amount and use the same amount immediately afterward, allowances are expected to be cleared after execution. However, care should be taken when adding components or new routers, as this may cause unexpected reverts in edge cases.

Examples include an underlying component being incompatible with `ERC4626` or `ERC7540` despite expectations (e.g., `requestDeposit` using a different asset value than provided), or a new router not utilizing the full allowance.

Although these are extreme edge cases, they can leave unused allowances, and cause reverts in subsequent actions.

Recommendation

Be aware of this situation and ensure that all components and routers always use the full allowance during execution. Alternatively, consider resetting the allowance to 0 first if any remaining allowance exists.

Resolution

Nashpoint Team: Acknowledged.

I-11 | Users May Be Unable To Withdraw From Adapter

Category	Severity	Location	Status
Documentation	Info	DigiftAdapter.sol: 751	Acknowledged

Description

`DigiftAdapter` enforces minimum amounts for `requestDeposit` and `requestRedeem`.

Assume the minimum for `deposit = 1000e6 USDC` and minimum for `redeem = 10e18 shares` (as configured in tests), users may have deposits where their total shares represent less than the minimum required to request a redemption from the component.

Following the intended design, the rebalancer will be unable to execute a redemption request for these users, unless an admin changes the minimum amount or other users queue redemptions, which will allow the total request to exceed the minimum.

Recommendation

Consider documenting this clearly to ensure users are aware of it.

Resolution

Nashpoint Team: Acknowledged.

I-12 | Unused State Variables In QuoterV1

Category	Severity	Location	Status
Informational	Info	QuoterV1.sol: 33-36	Resolved

Description

`QuoterV1` declares three state variables that are never written to or read from anywhere in the codebase:

```
/* STATE */
mapping(address => bool) public isErc4626;
mapping(address => bool) public isErc7540;
bool public isInitialized;
```

Recommendation

Consider removing unused state variables.

Resolution

Nashpoint Team: The issue was resolved in commit 6259916.

I-13 | Missing Public Functions In Multiple Interfaces

Category	Severity	Location	Status
Informational	Info	N/A	Acknowledged

Description

Several interfaces are incomplete compared to their implementations.

Missing functions include:

• `INode` (9 methods): Ownable functions, `multicall()`, rebalance getters, swing pricing getters

• `INodeRegistry` (9 methods): Ownable functions, UUPS upgrade methods, config getters

• `IQuoterV1` (4 methods): Type checks, initialization state, registry reference

• `INodeFactory` (2 methods): Implementation and registry getters

Recommendation

Consider adding these methods or documenting why they're excluded.

Resolution

Nashpoint Team: Acknowledged.

I-14 | Misleading Comment Regarding deltaClosedPct

Category	Severity	Location	Status
Informational	Info	QuoterV1.sol: 145	Resolved

Description

The `deltaClosedPct` value represents the percentage of the delta relative to the `targetReserveAssets`:

`deltaClosedPct = Math.mulDiv(deltaClosed, WAD, targetReserveAssets)`.

However, the comments “It is the inverse of the percentage of the reserve assets shortfall closed by the deposit” and “Get `reserveImpact` as a measure of how much the deposit helps to close any asset shortfall” gives the impression that `deltaClosedPct` was meant to represent the percentage of the shortfall, not of the `targetReserveAssets`.

Recommendation

Update the comments.

If the intention was to calculate the reserve impact based on the shortfall percentage, `deltaClosedPct` should be `Math.mulDiv(deltaClosed, WAD, shortfall)`

Resolution

Nashpoint Team: The issue was resolved in commit 6259916.

I-15 | Incorrect Comment In DigiftEventVerifier

Category	Severity	Location	Status
Informational	Info	DigiftEventVerifier.sol: 234	Resolved

Description

In line 234 of the `DigiftEventVerifier`, the comment "// Structure: (`stToken`, `investorList`, `quantityList`, `currencyTokenList`, `amountList`, timestamp)" is incorrect, as the last parameter in the structure is `feeList`, not timestamp.

Recommendation

Update the comment.

Resolution

Nashpoint Team: The issue was resolved in commit d48b89f.

I-16 | Warning Regarding Dust Accumulation In Digift

Category	Severity	Location	Status
Warning	Info	DigiftAdapter.sol: 677	Acknowledged

Description

The `settleDeposit` and `settleRedeem` functions in `DigiftAdapter` accumulate the dust amount to the last node in the loop. The adapter defines `minDepositAmount` and `minRedeemAmount`, which are set to non-trivial values in the tests.

Accumulating dust to the last node is not an issue under normal conditions where the minimum amounts are set. However, if these minimum amounts are ever set to 0, it becomes possible to request only 1 wei of shares.

In that case, `sharesToReimburse` may exceed 1 wei due to the dust accumulated on the last node, which can cause an underflow later.

Note that this is an extreme edge case scenario where the minimum amounts must be set to 0, a node must have a request with an extremely low value, and that node must also be the last one during settlement.

Recommendation

Be aware of this and do not set minimum amounts to 0.

Resolution

Nashpoint Team: Acknowledged.

I-17 | Zero Share Minting Allowed

Category	Severity	Location	Status
Warning	Info	Node.sol	Resolved

Description

The Node contract does not revert when the resulting share amount is 0 during a deposit or mint operation. This can cause users to lose their assets without receiving any shares, especially when the share price is inflated.

Recommendation

Do not allow zero share minting.

Resolution

Nashpoint Team: The issue was resolved in commit 93cc526.

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
L-01	Can Invest In Blacklisted Components	Validation	Low	Resolved
L-02	Unsafe External Call In NodeFactory	Validation	Low	Resolved
L-03	Users May Avoid Paying Performance Fees	Logical Error	Low	Acknowledged
I-01	Consider Early Return In _fulfillRedeemFromReserve	Informational	Info	Acknowledged
I-02	Tradeoffs Related To Share Valuation In M-08	Warning	Info	Acknowledged
I-03	Blocked Revoke Actions In Policies With Blacklist	Logical Error	Info	Acknowledged

L-01 | Can Invest In Blacklisted Components

Category	Severity	Location	Status
Validation	Low	BaseComponentRouter.sol: 120	Resolved

Description

The fix for L-23 introduces whitelist status in `_computeDepositAmount` and prevents deposits when a component is not whitelisted.

However, since issue L-05 is acknowledged, whitelist and blacklist statuses can still conflict, allowing a blacklisted component to also remain whitelisted.

Blacklisting does not automatically remove a component from the whitelist, and the issue persists until the component is manually de-whitelisted or removed by the owner.

Recommendation

Check the blacklist status in `_computeDepositAmount` as well or always ensure blacklisting and de-whitelisting happens together.

Resolution

Nashpoint Team: The issue was resolved in commit 2a6584f.

L-02 | Unsafe External Call In NodeFactory

Category	Severity	Location	Status
Validation	Low	NodeFactory.sol: 69	Resolved

Description

`SetupCalls` are introduced to the `NodeFactory` to allow Node deployers to configure their Nodes during deployment. This is mainly intended for performing policy calls that are protected by the `onlyNodeOwner` modifier, since the owner at the time of these calls is the factory contract itself.

However, there is no guarantee that these setup calls will be performed only on policies. Any target address can be called with any payload, without any protection. They could even be used for malicious actions, such as funding an exploiter through the factory contract or interacting with OFAC-sanctioned accounts.

While there is no incentive to perform such actions and the likelihood is extremely low, it could still be legally binding for Nashpoint in the future because, on paper, it would appear that the 'Nashpoint-owned factory' is the entity carrying out these actions.

Recommendation

Do not allow arbitrary target addresses to be called through the factory. Keep a list of policies that can be called during setup, and allow only those addresses as valid targets.

Resolution

Nashpoint Team: The issue was resolved in commit 4fe012b.

L-03 | Users May Avoid Paying Performance Fees

Category	Severity	Location	Status
Logical Error	Low	Node.sol: 359-366	Acknowledged

Description

In the fix for M-03, the management fee payment can be bypassed if there is not enough balance in the Node. This will make sure that `startBalance` operation will not be blocked.

This will raise one new issue here: The users may avoid paying performance fees.

e.g.

1. The node owner starts rebalance in timestamp X. `rebalanceWindow = 1 hour` , `rebalanceCooldown = 23 hours`.

2. Alice deposits 1000 USDC in timestamp X + 1.

3. In timestamp X + 1 hour, the rebalancer invests 1000 USDC into one component. Assume that the target reserve ratio is 0.

4. In timestamp X + 1 hour + 1, Alice requests redeem.

5. In timestamp X + 24 hour, the node rebalancer will start another round of rebalance. The balance in the node is 0. The management fee will be bypassed here.

6. The rebalancer triggers `fulfillRedeemRequest`. Alice can get back her asset without paying any fees here.

Recommendation

In `startRebalance`, if the balance is not enough to pay the management fee, record the management fee as debt, and update the `lastPayment` timely.

Resolution

Nashpoint Team: Acknowledged.

I-01 | Consider Early Return In _fulfillRedeemFromReserve

Category	Severity	Location	Status
Informational	Info	Node.sol: 698	Acknowledged

Description

The _fulfillRedeemFromReserve function sets balance = max(currentBalance, 1) and, if assetsToReturn > balance, sets assetsToReturn = balance. When actual reserve is 0, this forces assetsToReturn = 1, which later fails with ExceedsAvailableReserve in _finalizeRedemption function.

```
function _fulfillRedeemFromReserve(address controller) internal {
...
uint256 balance = Math.max(IERC20(asset).balanceOf(address(this)), 1);
uint256 assetsToReturn = convertToAssets(request.pendingRedeemRequest);
...
if (assetsToReturn > balance) {
sharesPending = (sharesPending * balance - 1) / assetsToReturn + 1;
assetsToReturn = balance; // balance may be 0, but forced to 1 above
}
_finalizeRedemption(controller, assetsToReturn, sharesPending);
}
```

Recommendation

Although the flow eventually reverts when the balance is zero, consider reverting or returning early in that case.

Resolution

Nashpoint Team: Acknowledged.

I-02 | Tradeoffs Related To Share Valuation In M-08

Category	Severity	Location	Status
Warning	Info	ERC7540Router.sol: 278	Acknowledged

Description

The fix for the previous M-08 introduces a mixed approach in which the `maxWithdraw` value at the locked rate is used for the normal case, while the current-rate share conversion is used when withdrawals are paused.

While this approach fixes the issue in most cases and the likelihood of the paused state is low, it is important to note that it will still result in incorrect valuation when withdrawals are paused.

Unfortunately, this is one of the trade-offs of asynchronous share valuations. In a paused scenario, the protocol must either intentionally undervalue the total underlying assets by continuing to use `maxWithdraw` and keeping claimable assets at 0, or accept the risk of overvaluation by using the current rate.

From a security perspective, undervaluing total assets may be the safer option, as it would prevent insolvency if everyone attempted to redeem their shares in an extremely rare scenario. However, this also means that new depositors would receive more shares than their deposits justify.

Recommendation

Note that this issue aims to highlight possible edge-cases after the fix of M-08. The decision to use the current rate when withdrawals are paused should be made deliberately, as it still carries some risk of overvaluation, and it is important to consider the trade-offs of both scenarios.

Resolution

Nashpoint Team: Acknowledged.

I-03 | Blocked Revoke Actions In Policies With Blacklist

Category	Severity	Location	Status
Logical Error	Info	GatePolicyBase.sol: 64	Acknowledged

Description

The updated policies now include a blacklisting feature as well. Both the whitelist and blacklist variants apply actor checks to the spender on every approval, regardless of the amount.

This also blocks `approve(spender, 0)` calls intended to revoke existing allowances if the spender has since become blacklisted.

Similarly, removing an operator via a `setOperator(operator, false)` call is also blocked once the operator becomes blacklisted. Ability to revoke approvals and operators when they become blacklisted is crucial and should not be blocked.

Recommendation

Allow revoking approvals and removing operators when using `GatePolicy` with blacklisting, and do not perform `actorCheck` on spender/operator in these special cases.

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