GA GUARDIAN

FastLane

Protocol Review

Security Assessment

December 8th, 2025

Summary

Audit Firm Guardian

Prepared By Curiousapple, 0xCiphky, Manu, Michael Lett

Client Firm FastLane

Final Report Date December 8, 2025

Audit Summary

FastLane engaged Guardian to review the security of their FastLane Protocol. From the 27th of October to the 12th of November, a team of 4 auditors reviewed the source code in scope. All findings have been recorded in the following report.

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

Security Recommendation Guardian recommends a follow-up security review of the protocol at a finalized frozen commit. The Fastlane team has acknowledged this and followed our recommendation with a secondary review.

Table of Contents

Project Information

Project Overview 5

Audit Scope & Methodology 6

Smart Contract Risk Assessment

Invariants Assessed 9

Findings & Resolutions 14

Addendum

Disclaimer 58

About Guardian 59

Project Overview

Project Summary

Project Name	FastLane
Language	Solidity
Codebase	https://github.com/FastLane-Labs
Commit(s)	Main Review commit: 65cb4bfeda9943638a333f15497f52fbeb7e09be Remediation Review commit: 2c329cb98ec45679fb922cfb070f38ec108868cd

Audit Summary

Delivery Date	December 5, 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	4	0	0	0	0	4
Medium	10	0	0	0	1	9
Low	7	0	0	0	0	7

Audit Scope & Methodology

```
contract,source,total,comment
fastlane-contracts-internal/src/shmonad/AtomicUnstakePool.sol,101,174,40
fastlane-contracts-internal/src/shmonad/Constants.sol,26,40,21
fastlane-contracts-internal/src/shmonad/Errors.sol,52,63,8
fastlane-contracts-internal/src/shmonad/Events.sol,100,109,4
fastlane-contracts-internal/src/shmonad/FLERC20.sol,135,328,157
fastlane-contracts-internal/src/shmonad/FLERC4626.sol,230,448,135
fastlane-contracts-internal/src/shmonad/Holds.sol,51,140,74
fastlane-contracts-internal/src/shmonad/Policies.sol,304,759,338
fastlane-contracts-internal/src/shmonad/PrecompileHelpers.sol,107,152,27
fastlane-contracts-internal/src/shmonad/ShMonad.sol,83,214,98
fastlane-contracts-internal/src/shmonad/StakeTracker.sol,757,1304,383
fastlane-contracts-internal/src/shmonad/Storage.sol,111,171,30
fastlane-contracts-internal/src/shmonad/Types.sol,137,189,59
fastlane-contracts-internal/src/shmonad/ValidatorRegistry.sol,194,266,27
fastlane-contracts-internal/src/shmonad/libraries/AccountingLib.sol,69,240,157
fastlane-contracts-internal/src/shmonad/libraries/FeeLib.sol,158,260,80
fastlane-contracts-internal/src/shmonad/libraries/HoldsLib.sol,40,58,7
fastlane-contracts-internal/src/shmonad/libraries/StakeAllocationLib.sol,57,104,33
fastlane-contracts-internal/src/shmonad/libraries/StorageLib.sol,,,
source count: {
  total: 5019,
  source: 2712,
  comment: 1678,
  single: 984,
  block: 694,
  mixed: 109,
  empty: 778,
  todo: 14,
  blockEmpty: 40,
  commentToSourceRatio: 0.6187315634218289
}
```

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 FastLane, 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
ACCT-01	Asset delta equals Liability+Equity delta (fundamental accounting equation)	✅	✅	✅	10M+
ACCT-02	Deposit and withdrawal in same epoch results in loss (no round-trip profit without donation)	✅	✅	✅	10M+
BALS-01	totalSupply >= totalCommittedAllPolicies + totalUncommittingAllPolicies + sum(uncommitted)	✅	✅	✅	10M+
BALS-02	shMonadTotalSupply == s_supply.total	✅	✅	✅	10M+
BALS-03	committedTotalSupply == s_supply.committedTotal	✅	✅	✅	10M+
BALS-04	balanceOf(user) == s_balances[user].uncommitted	✅	✅	✅	10M+
BALS-05	allocatedAmount >= 0 (pool liquidity non-negative)	✅	✅	✅	10M+
CRANK-01	Goodwill should be zero without donation (no untracked funds)	✅	✅	✅	10M+
CRANK-02	Goodwill during crank should be zero with all active validators	✅	❌	✅	10M+
INVAR-EPC	Balance decreased within expected bounds across epoch; total assets cover liabilities	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
POL-GLOB-01	Sum of committed uncommitted and uncommitting balances <= total supply	✅	✅	✅	10M+
POL-GLOB-02	Policy committed balances match sum of user's committed for all policies	✅	✅	✅	10M+
POL-GLOB-03	Policy uncommitting balances match sum of user's uncommitting for all policies	✅	✅	✅	10M+
POL-GLOB-04	Holds never exceed committed balances	✅	✅	✅	10M+
POL-CMT-01	Policy must be active and have at least the primary agent	✅	✅	✅	10M+
POL-CMT-02	Committed balance of commit recipient should increase by shares	✅	✅	✅	10M+
POL-CMT-03	Uncommitted balance of sender should decrease by shares	✅	✅	✅	10M+
POL-CMT-04	Total committed supply should increase by committed shares	✅	✅	✅	10M+
POL-CMT-05	Total supply should stay the same after commit	✅	✅	✅	10M+
POL-DEP-CMT-01	Total supply should increase by minted shares	✅	✅	✅	10M+
POL-DEP-CMT-02	Uncommitted balance should increase by (minted - committed) shares	✅	✅	✅	10M+
POL-UNCMT-01	Uncommit start block set and uncommitting amount increased	✅	✅	✅	10M+
POL-UNCMT-02	User must have sufficient unheld committed balance	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
POL-UNCMT-03	Requesting uncommit reduces committed balance and totals	✅	✅	✅	10M+
POL-UNCMT-04	Requesting uncommit sets min balance to new value	✅	✅	✅	10M+
POL-UNCMT-05	Requesting uncommit increases total uncommitting	✅	✅	✅	10M+
POL-RUAC-01	Completor is overwritten with supplied address	✅	✅	✅	10M+
POL-RUAC-02	Approval's share allowance increases by shares	✅	✅	✅	10M+
POL-COMP-01	block.number > uncommitStartBlock + escrowDuration	✅	✅	✅	10M+
POL-COMP-02	User must have enough uncommitting balance before completion	✅	✅	✅	10M+
POL-COMP-03	Deducts from uncommitting and credits user's uncommitted	✅	✅	✅	10M+
POL-CUWA-01	Only approved completor may complete uncommit	✅	✅	✅	10M+
POL-CUWA-02	Finite approval shares must cover and decrement by shares completed	✅	✅	✅	10M+
POL-CUWA-03	Infinite approvals (type(uint96).max) are never decremented	✅	✅	✅	10M+
AGT-GLOB-01	Held should be transient no held balances in pre-state	✅	✅	✅	10M+
AGT-01	Policy is active and has an agent	✅	✅	✅	10M+
AGT-02	Current actor is an authorized agent for the policy	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
AG-HOLD-01	Delta held equals requested shares after successful hold	✅	✅	✅	10M+
AG-HOLD-02	Policy/global totals stay aligned after hold	✅	✅	✅	10M+
AG-HOLD-03	Hold is bounded by committed (held <= committed)	✅	✅	✅	10M+
AG-HOLD-04	Hold handler must not change real balances or supply	✅	✅	✅	10M+
AGT-REL-01	Release calls decrease held by requested amount	✅	✅	✅	10M+
AGT-SUP-01	Agent transfers never change total shMON supply	✅	✅	✅	10M+
AGT-UNCOMM-01	Spending from committed never increases uncommitting balance	✅	✅	✅	10M+
AGT-TFC-01	Committed >= held before spending	✅	✅	✅	10M+
AGT-TFC-02	Total committed supply never decreases	✅	✅	✅	10M+
AGT-TFC-FLOW-01	Destination receives exactly what leaves the source (committed path)	✅	✅	✅	10M+
AGT-TFC-DEST-01	Destination committed must increase when receiving committed	✅	✅	✅	10M+
AGT-TTU-FLOW-01	Source committed delta mirrors policy delta (uncommitted path)	✅	✅	✅	10M+
AGT-TTU-DEST-01	Destination uncommitted balance increases when receiving	✅	✅	✅	10M+
AGT-TTU-AGENT-01	Source account must not be a policy agent	✅	✅	✅	10M+

Invariants Assessed

ID	Description	Tested	Passed	Remediation	Run Count
AGT-WITH-DEST-01	Destination holds only ETH (no shMON balance changes)	✅	✅	✅	10M+
AGT-WITH-SUP-01	Total supply must strictly decrease on withdraw when amount > 0	✅	✅	✅	10M+
AGT-WITH-ETH-01	Contract ETH loss equals destination ETH gain	✅	✅	✅	10M+
AGT-WITH-SRC-01	Source uncommitted balance can only decrease	✅	✅	✅	10M+
AGT-BOOST-01	Boost yield burns shMON but retains ETH in contract	✅	✅	✅	10M+
AGT-BOOST-02	Source balances follow spend semantics	✅	✅	✅	10M+
AGT-BOOST-03	Committed totals reflect burned supply net of top-up	✅	✅	✅	10M+
AGT-BOOST-04	Non-source actors remain unaffected during boost	✅	✅	✅	10M+

Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	Atomic Liquidity Fee Bypassed At Max Capacity	Logical Error	High	Resolved
H-02	Withdrawals Exceed Intended Limits	Logical Error	High	Resolved
H-03	Overflow DOS In Crank Locks Funds	Logical Error	High	Resolved
M-01	Auto Top-up Ignores minCommitted Threshold	Unexpected Behavior	Medium	Resolved
M-02	Agent Instant-Uncommit Can Be Bypassed	Trust Assumptions	Medium	Resolved
M-03	Single-Failure Deactivation Of New Validators	Logical Error	Medium	Resolved
M-04	Withdrawal Rewards Misclassified As Surplus	Rewards	Medium	Resolved
M-05	Boosting From Committed Shares	Logical Error	Medium	Resolved
M-06	MEV Through Atomic Withdrawals	MEV	Medium	Partially Resolved
M-07	solveGrossGivenNet Overflow	Math	Medium	Resolved
M-08	Double Liability Deduction In Target Stake	Logical Error	Medium	Resolved
M-09	Inactive Validator's Pending Stake Not Unstaked	Logical Error	Medium	Resolved
L-01	Unused changeCommittedTotalSupply	Best Practices	Low	Resolved
L-02	Lack Of Smart Contract Signatures Support	Best Practices	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-03	Commission Reapplied In Process	Logical Error	Low	Resolved
L-04	Access Control: denullifyEligibilityMap()	Access Control	Low	Resolved
L-05	Dead Code: _queueNetDepositsForStaking()	Best Practices	Low	Resolved
L-06	Preview And Max Withdraw Drift	Rounding	Low	Resolved
L-07	Incorrect Unique Block.coinbase Assumption	Assumptions	Low	Resolved

Discussion

ID	Title	Category	Severity	Status
D-01	Direct Delegation For Risk-Free Rewards	Rewards	Discussion	Acknowledged
D-02	Validators Can Game Stake Allocation	Gaming	Discussion	Acknowledged
D-03	Any Agent Can Disable Policy For All Agents	Access Control	Discussion	Acknowledged
D-04	boostYield() Attributes To Current Validator	Rewards	Discussion	Acknowledged
D-05	Escrow Duration Locks Funds	Validation	Discussion	Acknowledged
D-06	Redundant realTotalSupply() Function	Best Practices	Discussion	Acknowledged
D-07	removePolicyAgent Assigns Last Agent As Prime	Configuration	Discussion	Acknowledged
D-08	yieldOriginator Parameter Can Be Spoofed	Informational	Discussion	Acknowledged
D-09	Revenue Offset Creates Friction For Withdrawals	Informational	Discussion	Acknowledged
D-10	Escrow Period Bypassed	Informational	Discussion	Acknowledged
D-11	depositAndCommit No Deposit Share Return	Best Practices	Discussion	Resolved
D-12	Atomic Pool Init Defaults	Best Practices	Discussion	Acknowledged
D-13	Malicious Coinbase Address Can Lead To DOS	Gas Griefing	Discussion	Resolved

Discussion

ID	Title	Category	Severity	Status
D-14	N(delta) Wraps Arbitrary Offsets	Validation	Discussion	Acknowledged
D-15	sum(balanceOf) = totalSupply - ERC20 Invariant	Informational	Discussion	Acknowledged
D-16	Revenue-Based Stake Allocation Creates Barrier	Informational	Discussion	Acknowledged
D-17	Assumptions On Monad Staking Precompile	Informational	Discussion	Acknowledged
D-18	Dead Code In AccountingLib	Superfluous Code	Discussion	Resolved

H-01 | Atomic Liquidity Fee Bypassed At Max Capacity

Category	Severity	Location	Status
Logical Error	High	/src/shmonad/ShMonad.sol: 196	Resolved

Description PoC

The function agentWithdrawFromCommitted is meant to let a policy agent to pull from a user's committed balance while paying a fee to the atomic pool.

This function has an isUnderlying argument that allows the caller to request the withdrawal to be denominated in MON.

When the isUnderlying argument is set to false it first converts the share amount to gross assets, both gross and fee, the code then explicitly sets _assetsToReceive = gross - fee.|

However, when the isUnderlying argument is set to true the function is meant to take the specified amount as net and calls _getGrossAndFeeFromNetAssets returning both gross and fee, however, when there is not enough liquidity in the pool, the function will cap the returned gross amount while maintaining the original net amount as the amount to transfer.

This not only leads to the agent with being able to bypass the liquidity cap but also to seize the fee meant to buffer the float and capture protocol revenue, but also leaves the pool insolvent relative to its accounting, as the function calls in both cases function _accountForWithdraw(uint256 netAmount, uint256 fee).

Which expects the netAmount to be passed along with the fee, whereas in this case, the gross amount is being sent along with a fee that is never captured producing a mismatch between the protocol accounting and the real protocol state.

Moreover an agent can repeat the operation whenever the pool refills, harvesting the entire liquidity each time while paying zero utilization fee and leaving other users without the ability to instant withdrawal.

Recommendation

After calling _getGrossAndFeeFromNetAssets, calculate the deliverable net by calculating gross - fee and if it doesn't match the requested net amount either revert or return the actual net amount.

Resolution

FastLane Team: The issue was resolved in PR#532.

H-02 | Withdrawals Exceed Intended Limits

Category	Severity	Location	Status
Logical Error	High	StakeTracker.sol: 1486	Resolved

Description PoC

The `_accountForWithdraw` function manages accounting for instant withdrawals from the atomic unstaking pool. It allows withdrawals up to `s_atomicAssets.allocatedAmount` plus `globalRewardsPtr_N(0).earnedRevenue`.

The issue arises when `allocatedAmount` is depleted and a shortfall is covered using `earnedRevenue`. In this case, the function increases `allocatedAmount` by the shortfall amount.

As a result, subsequent checks compare `_distributedAmount + netAmount` against an inflated `_allocatedAmount + _earnedRevenueOffset`, effectively expanding the withdrawal limit.

A user can repeatedly perform instant withdrawals as long as each withdrawal remains below earned revenue, gradually draining the contract and consuming funds reserved for other protocol operations.

Recommendation

Modify `_accountForWithdraw` to properly account for already used `earnedRevenue`, ensuring instant withdrawals cannot exceed the combined allocated and `earnedRevenue` amount.

Resolution

FastLane Team: The issue was resolved in PR#587.

H-03 | Overflow DOS In Crank Locks Funds

Category	Severity	Location	Status
Logical Error	High	StakeTracker.sol	Resolved

Description

In `StakeTracker._crankGlobal()`, in the `_checkSetNewAtomicLiquidityTarget` call, there is the possibility of an overflow, in a specific scenario:

• There has been an atomic unstake
• A large amount of shMON has been `requestUnstaked`

This causes equity to be very small, because most funds are offset by a liability to represent the unstake request.

Then, when we get to the global crank phase after that setup, in `_checkSetNewAtomicLiquidityTarget` we calculate `_newScaledTargetPercent` as `s_atomicAssets.distributedAmount` / `_totalEquity` . This figure can be reach into the thousands of percents, and has in live testnet deployments.

The overflow and revert then occurs when we call `_unscaledTargetLiquidityPercentage()` and pass in this value when it is larger than 655% (the max a uint16 can hold, when represented in basis points)

Thus, the crank cannot complete, and therefore the users who `requestUnstaked` can never reach the epoch where they can call `completeUnstake`, so funds are stuck.

Recommendation

Cap the calculated `_newScaledTargetPercent` at 100%, and do an “emergency” decrease of allocated and distributed amounts in `_afterRequestUnstake` if the unstaked amount + allocated + recent revenue is > total equity.

Resolution

FastLane Team: The issue was resolved in PR#557.

M-01 | Auto Top-up Ignores minCommitted Threshold

Category	Severity	Location	Status
Unexpected Behavior	Medium	src/shmonad/Policies.sol: 660	Resolved

Description PoC

According to the comments in the code documenting the top-up mechanism. When a user's committed balance falls below `minCommitted`, the system attempts to commit more shares from their uncommitted balance. However, this is not always true.

When the `_spendFromCommitted` function processes an agent's spend, it computes `fundsAvailable` as the caller's current committed balance minus any holds and only invokes `_tryTopUp` if the requested shares exceed that pre-spend amount.

Consequently, as long as the agent keeps each transfer less than or equal to the balance that is already committed, the top-up path never runs. The committed balance is decremented immediately without ever triggering a top-up from the user's uncommitted funds.

This defeats the intended guarantee that the auto-top-up keeps at least `minCommitted` shares available.

Recommendation

Trigger the top-up when the updated `fundsAvailable` is below the `shares + minCommited`

Resolution

FastLane Team: The issue was resolved in PR#573.

M-02 | Agent Instant-Uncommit Can Be Bypassed

Category	Severity	Location	Status
Trust Assumptions	Medium	src/shmonad/ShMonad.sol: 131	Resolved

Description

The `agentTransferFromCommited` function prevents agents from uncommitting their own balance from a policy.

However, a malicious agent can shuffle their committed balance to any non-agent function they own using this function then immediately after call `agentTransferToUncommitted` to transfer the funds back to themselves. Bypassing the protection.

Recommendation

Prevent agents from transferring their own committed balances to non-agent addresses without a delay.

Resolution

FastLane Team: The issue was resolved in PR#586.

M-03 | Single-Failure Deactivation Of New Validators

Category	Severity	Location	Status
Logical Error	Medium	StakeTracker.sol: 711	Resolved

Description

The `_rollValidatorEpochForwards` function advances the validator epoch state and updates accounting for each validator. At the end of this process, a validator is queued for deactivation when both `s_validatorData[valId].inActiveSet_Last` and `s_validatorData[valId].inActiveSet_Current` are false.

However, newly added validators start with `s_validatorData[valId].inActiveSet_Current` at its default value of false, which then causes `_advanceActiveSetFlags` to set `inActiveSet_Last` to false.

If the first crank involving that validator experiences a failure that sets `inActiveSet_Current` to false again, the deactivation condition is met immediately.

As a result, the validator can be deactivated after just one failure, bypassing the intended two-strike mechanism that applies once a validator has prior epoch history. This situation can occur when a validator is added on Monad during the delay window and becomes active in epoch n+2.

Recommendation

Ensure newly added validators are initialized with appropriate active set state or implement a grace period before applying standard deactivation logic, preventing unintended single-failure removals.

Resolution

FastLane Team: The issue was resolved in PR#552.

M-04 | Withdrawal Rewards Misclassified As Surplus

Category	Severity	Location	Status
Rewards	Medium	StakeTracker.sol: 1244	Resolved

Description PoC

The `_handleCompleteDecreasedAllocation` function compares the received amount from a withdrawal to the undelegated `expectedAmount` and books any excess as "surplus." On Monad, a withdrawal can include both the returned principal and rewards accumulated during the exit window.

As a result, the excess over `expectedAmount` often represents earned rewards rather than surplus capital. Currently, this excess is redirected and potentially re-staked through `queueToStake`, bypassing the standard reward handling path.

This causes validator rewards to be misclassified as surplus, skipping commission, underreporting `earnedRevenue`, and distorting validator performance metrics used for allocation.

Recommendation

When `amount > expectedAmount`, treat `amount - expectedAmount` as validator rewards and process it through the same reward-handling logic used in `_handleEarnedStakingYield` to ensure proper accounting.

Resolution

FastLane Team: The issue was resolved in PR#549.

M-05 | Boosting From Committed Shares

Category	Severity	Location	Status
Logical Error	Medium	ShMonad.sol: 235	Resolved

Description

`agentBoostYieldFromCommitted` calls `_handleBoostYield`, which books revenue and increases `queueToStake` even though no new MON arrives and the committed shares are already staked.

During `crank()`, this synthetic `queueToStake` can be netted in `_settleGlobalNetMONAgainstAtomicUnstaking` or reduced alongside `queueForUnstake` in `_offsetLiabilitiesWithDeposits` when there are uncovered liabilities and sufficient current assets, pushing genuine withdrawals or other payments into later epochs.

The call also inflates `globalRewardsPtr_N(0).earnedRevenue`. The instant-withdraw path relies on that value as an offset: if `distributed + netAmount > allocated`, it will allow the withdrawal so long as `allocated + earnedRevenue` covers it, then enqueue the shortfall to `queueForUnstake` and increase `allocatedAmount`.

When `earnedRevenue` was boosted without new cash, the contract still pays out immediately by drawing on liquid assets. This overstates near-term liquidity and forces liquidity to come from other areas, such as `reservedAmount`, until the backfilling unstake settles.

Recommendation

Keep `queueForUnstake` and revenue tied to real inflows so staking, withdrawals, and instant-withdraw capacity reflect actual liquidity.

Resolution

FastLane Team: The issue was resolved in PR#580.

M-06 | MEV Through Atomic Withdrawals

Category	Severity	Location	Status
MEV	Medium	src/shmonad/StakeTracker.sol: 1486	Partially Resolved

Description

The atomic unstake paths (withdraw, redeem, `agentWithdrawFromCommitted`) price every withdrawal off the current utilization snapshot returned by `_getLiquidityForAtomicUnstaking`.

Each successful instant unstake immediately increments `s_atomicAssets.distributedAmount` in `_accountForWithdraw` so the next withdraw is repriced at a higher fee rate.

An MEV bot that holds shMON can observe large instant withdrawals in the mempool and front-run them with a tiny redeem/withdraw first to push utilization up and pocket the victim's extra fee via it's remaining shMON balance.

Because the fee curve is affine (fee(u) = c + m·u) with m = 1% by default, even a small utilization bump significantly raises the victim's fee. A holder with just ~1% of the supply already profits from this attack.

Recommendation

Add slippage controls to every instant-unstake path so users aren't forced to execute at a worse fee than they previewed.

Resolution

FastLane Team: The issue was resolved in PR#583.

M-07 | solveGrossGivenNet Overflow

Category	Severity	Location	Status
Math	Medium	src/shmonad/libraries/FeeLib.sol: 311	Resolved

Description PoC

When solveGrossGivenNet (used by withdraw and previewWithdraw) enters the quadratic branch, it computes (2*m*RAY*targetNet)/L inside Math.mulDiv, any withdrawal request with a targetNet greater than (2^256-1)/(2*m*RAY) will result in an overflow.

With DEFAULT_SLOPE_RATE_RAY m = 1% (1e25), any withdrawal request above 5.79M MON causes the intermediate product to exceed 256 bits, so Math.mulDiv reverts with MulDivFailed() before applying liquiditycaps or fees.

Raising the slope rate will proportionally reduce the overflow threshold.

Recommendation

Replace the 256-bit function mulDiv with the 512-bit version fullMulDiv

Resolution

FastLane Team: The issue was resolved in PR#532.

M-08 | Double Liability Deduction In Target Stake

Category	Severity	Location	Status
Logical Error	Medium	AccountingLib.sol: 297	Resolved

Description

The `targetStakedAmount` function determines the protocol's next staking allocation based on available equity after accounting for required float.

It computes `totalEquity` as `_totalStaked + nativeTokenBalance - totalLiabilities`, where `totalLiabilities` includes `liabilities.redemptionsPayable`, `liabilities.rewardsPayable`, and `admin.commissionPayable`.

To derive the staking target, the function subtracts `_targetAtomicAllocatedAmount` and `workingCapital.reservedAmount` from this equity.

The issue arises because some liabilities are already reflected in both components. In `_handleValidatorRewards`, when `liabilities.rewardsPayable` increases, `reservedAmount` also increases by the same value.

Similarly, `_handleCompleteDecreasedAllocation` tops up `currentLiabilities` to cover `liabilities.redemptionsPayable`, which are also reserved until redemption.

As a result, the `targetStakedAmount` calculation deducts these amounts twice, once as liabilities and again through reserved assets leading to an understated staking target and potential underutilization of available equity.

Recommendation

Adjust the `targetStakedAmount` computation to avoid double-counting obligations that are already captured in both `totalLiabilities` and `reservedAmount`.

Resolution

FastLane Team: The issue was resolved in PR#589.

M-09 | Inactive Validator's Pending Stake Not Unstaked

Category	Severity	Location	Status
Logical Error	Medium	StakeTracker.sol: 638-650	Resolved

Description

When cranking an inactive validator at (StakeTracker.sol:638-650) the protocol unstakes only _validatorUnstakableAmount and sets _nextTargetStakeAmount = 0.

However, _validatorUnstakableAmount is calculated as targetStakeAmount - pendingStaking. This means the pendingStaking amount would be excluded from the decrease allocation that should happen for same validator next epoch, leaving pending funds unaccounted.

Example:

Validator has 100 ETH target + 10 ETH pending. When inactive, only 90 ETH is queued for unstaking and nextTarget becomes 0.

The 10 ETH pending stake remains locked indefinitely.

Recommendation

Consider assigning difference between previous targetStakeAmount and _validatorUnstakableAmount as next target stake amount.

Resolution

FastLane Team: The issue was resolved in PR#563.

L-01 | Unused changeCommittedTotalSupply

Category	Severity	Location	Status
Best Practices	Low	Policies.sol: 473-474	Resolved

Description

The changeCommittedTotalSupply boolean parameter in the internal _commitToPolicy() function is always passed as true in all three call sites (lines 55, 78, 185). The conditional check on line 513 serves no purpose as the flag is never set to false.

```
function _commitToPolicy(
uint64 policyID,
address accountFrom,
address sharesRecipient,
uint256 shares,
bool changeCommittedTotalSupply  // Always true
)
```

Recommendation

Consider removing the changeCommittedTotalSupply parameter and the conditional check and always incrementing s_supply.committedTotal when committing shares.

Resolution

FastLane Team: The issue was resolved in PR#555.

L-02 | Lack Of Smart Contract Signatures Support

Category	Severity	Location	Status
Best Practices	Low	FLERC20.sol: 148-149	Resolved

Description

The implementation assumes EOA signers and does not support `ERC-4337` smart contract wallets or other account abstraction methods that are becoming increasingly common.

Recommendation

Consider adding support for `ERC-1271`: Smart contract signatures.

Resolution

FastLane Team: The issue was resolved in PR#578.

L-03 | Commission Reapplied In Process

Category	Severity	Location	Status
Logical Error	Low	Coinbase.sol: 61	Resolved

Description

The process function calculates commission based on the coinbase contract's entire current balance, distributes the commission, and then attempts to send the remaining rewards to delegators. If the payout to delegators fails, those untransferred rewards remain in the contract.

On the next process call, commission is recalculated on the full balance again, including the previously retained rewards. This effectively applies commission again to the same rewards, reducing the amount available to delegators.

Recommendation

Ensure commission is only calculated on newly accrued rewards, excluding any balance carried over from prior failed distribution attempts.

Resolution

FastLane Team: The issue was resolved in PR#551.

L-04 | Access Control: denullifyEligibilityMap()

Category	Severity	Location	Status
Access Control	Low	ValidatorRegistry.sol: 345	Resolved

Description

`denullifyEligibilityMap` is public with no access control, allowing anyone to modify `s_valEligibility` mapping.

The `s_valEligibility` mapping (`Storage.sol:102`) is never read in any protocol logic.

Only used in a view function getter, suggesting it's dead code.

Protocol confirmed in discussions that this is a deadcode.

Recommendation

Consider removing the `s_valEligibility` mapping and associated code.

Resolution

FastLane Team: The issue was resolved in PR#570.

L-05 | Dead Code: _queueNetDepositsForStaking()

Category	Severity	Location	Status
Best Practices	Low	StakeTracker.sol: 1460	Resolved

Description

The internal function `_queueNetDepositsForStaking()` at `StakeTracker.sol: 1460` is defined but has no callers in the codebase.

```
function _queueNetDepositsForStaking(uint256 amount) internal {
// Function body exists but is never called
}
```

Recommendation

Consider removing deadcode.

Resolution

FastLane Team: The issue was resolved in PR#579.

L-06 | Preview And Max Withdraw Drift

Category	Severity	Location	Status
Rounding	Low	src/shmonad/FLERC4626.sol: 139	Resolved

Description

`maxWithdraw` reports a net-asset amount based on `_convertToAssets(..., Floor)` while `previewWithdraw` inverts the path with `_convertToShares(..., Ceil)` to cover fees.

As a result the preview rounds the required shares up, so a user can request assets equal to `maxWithdraw` yet be asked to burn more shares than they actually have in balance.

Recommendation

Align the runtime withdraw flow with the same rounding and liquidity-aware math used in `maxWithdraw`.

Resolution

FastLane Team: Given recent exploits involving rounding and precision issues, we are purposefully making these rounding choices in favor of the security of the protocol.

We round down when users specify shares, to give them a conservative amount of assets they can definitely withdraw (maxWithdraw).

We round up when users specify assets, to give them a conservative number of shares they need to burn to get those assets (previewWithdraw).

In both cases we round in favor of the protocol and not the user. Changing either of these would sometimes round in favor of the user, and even though its fairly inconsequential (1 wei), we think we should always round defensively in favor of the protocol.

L-07 | Incorrect Unique Block.coinbase Assumption

Category	Severity	Location	Status
Unexpected Behavior	Low	StakeTracker.sol: 593-604	Resolved

Description

When cranking the linked list of validators, validators are primarily identified using their `block.coinbase` addresses which are stored in this linked list.

Each of those addresses is passed into `StakeTracker._crankValidator()`, and an associated validator ID is looked up. That validator ID is then passed into any internal functions.

This assumes that validator ID <> coinbase is strictly 1:1 relationship. However that assumption is incorrect - validator IDs are unique but many IDs can map to the same coinbase address.

This means that the same coinbase address could be used multiple times in that linked list, which would always map back to the same validator ID.

This relationship is protected at the stage where the owner adds validators to ShMonad, but in reality we will need to support validators that map to the same coinbase.

Recommendation

Refactor the linked list and any other identifying logic to always use validator ID, and only look up coinbase from ID at the point where it is required in the logic (e.g. to call `process()`)

Resolution

FastLane Team: The issue was resolved in PR#593.

D-01 | Direct Delegation For Risk-Free Rewards

Category	Severity	Location	Status
Rewards	Discussion	StakeTracker.sol: 880-881	Acknowledged

Description

A malicious actor can frontrun MEV reward distribution by directly delegating to validators via the Monad staking precompile immediately after MEV is earned in epoch N.

Since FastLane distributes MEV rewards to validators in epoch N+1 (via `externalReward()`), and direct delegations become active in N+1, the attacker receives a proportional share of these rewards without taking any performance risk.

This creates a `risk-free profit opportunity` that can be exploited repeatedly every epoch.

Example scenario:

Epoch 5: MEV reward is generated for validator V1.

Epoch 6 crank starts:
• Global crank passes.
• Validator crank begins.
• Validator rewards from Epoch 5 are credited to V1.

Since it is deterministic that the reward will be credited in Epoch 6, an attacker can delegate to V1 in Epoch 5, just before the epoch boundary, and receive a share of the reward without having participated in validation.

Attackers can repeatedly capture validator MEV rewards with no stake-performance exposure.

Recommendation

If considered an issue, a straightforward mitigation would be to send validator rewards immediately as they are earned, rather than deferring to the next epoch.

However, this may require a broader architectural refactor of the accounting.

Other solution is enforcing that `sendValidatorRewards` is only called during boundary period of epoch, hence even if someone notices and delegates later, their delegation would be active from n+2.

Resolution

FastLane Team: We do not perceive this to be an issue at this time. It is an unfortunate side effect of the Monad staking design. Full mitigations are impossible, and partial mitigations would only work for MEV earned during the boundary block - a very small percentage of the overall MEV. We do not believe a minor mitigation is worth the extra complexity.

Note also that if we could deposit the validator rewards when they're collected then we would, but MEV transactions are the most gas-sensitive of any transaction type, and the external reward function is quite gas intensive. External rewarding in each MEV transaction is a non-starter. Running a second crank would not fully solve the issue, either.

We're lobbying the monad team for the deposit wait period to be the same as the withdrawal wait period (1 epoch - 2 in boundary) rather than its shorter, current duration (0 - 1). From our POV, that is the only real solution.

D-02 | Validators Can Game Stake Allocation

Category	Severity	Location	Status
Gaming	Discussion	StakeAllocationLib.sol: 67-72 StakeTracker.sol: 444-449	Acknowledged

Description

Validators can manipulate stake allocation by:

Self-delegate large amounts on Monad (not via FastLane)
Call `sendValidatorRewards()` to spike their `earnedRevenue`
Next epoch's allocation uses avg of epochs n-1, n-2
Validator receives outsized share of `queueToStake`

Attack Economics:

Validator with 10 FastLane stake, self-delegates 80 (total 100)
Donates X rewards → loses only 20% (10% to FastLane, 10% to others)
Gains larger stake allocation worth more than 20% cost
Most profitable in early stages when FastLane share is small

Code:

```
// StakeAllocationLib.sol:67-72
targetDelta = queueToStake * validatorRevenue / globalRevenue;
// StakeTracker.sol:444 - uses n-1, n-2 epochs
earnedRevenueLast: globalRewardsPtr_N(0).earnedRevenue    // n-1
earnedRevenueCurrent: globalRewardsPtr_N(-1).earnedRevenue // n-2
```

Recommendation

Use n-2, n-3 epochs instead of n-1, n-2 to add 1-epoch delay, preventing immediate manipulation.

Resolution

FastLane Team:

This finding is a design choice of shMonad. Validators can freely bid for deposits each crank. You can monitor validator bidding here: https://analytics.shmonad.xyz/stake-distribution

Validators must keep in mind two costs:

1. The Fastlane fees taken on rewards

2. The validator bid would have to be sustained otherwise the algorithm will naturally start undelegating from that validator as soon as they stop bidding

This design allows shMonad holders to capture increase yield from validators bidding for deposits

Please note:

- We're not planning to change the smoothing window (n-1/n-2 → n-2/n-3); adding another delay would slow legitimate responsiveness without preventing repeated donations.
- Operationally, we'll monitor bidding strategies from validators and may change the algorithm in the future

D-03 | Any Agent Can Disable Policy For All Agents

Category	Severity	Location	Status
Access Control	Discussion	Holds.sol: 150-151	Acknowledged

Description

disablePolicy() function allows any single policy agent to irreversibly disable the entire policy, affecting all other agents and users.

Recommendation

Consider allowing only primary agents to disable policy

Resolution

FastLane Team: This is a design choice of the agent and policy system. Appointing agents is a permissioned action that we control, so any agent is a trusted entity.

D-04 | boostYield() Attributes To Current Validator

Category	Severity	Location	Status
Rewards	Discussion	StakeTracker.sol: 1168-1169	Acknowledged

Description

When `boostYield()` is called, it attributes the boosted yield to the current active validator's `earnedRevenue`, even though the boost may be unrelated to that validator's performance. This skews validator performance metrics.

Recommendation

Consider not attributing boosts to the current validator.

Resolution

FastLane Team: This is a design choice of the system. The function should always attribute to the current block's producer. If someone wants to attribute earnings to a different validator but still give all earnings to shMON holders, they should call `sendValidatorRewards()` with a 100% fee and specify a validatorID

D-05 | Escrow Duration Locks Funds

Category	Severity	Location	Status
Validation	Discussion	src/shmonad/Policies.sol: 279	Acknowledged

Description

`createPolicy(uint48 escrowDuration)` accepts any 48-bit duration without bounds. A policy owner can set `escrowDuration` near `type(uint48).max` (~9e13 blocks), and anyone who later deposits/commits into that policy will be unable to finish `completeUncommit` in any practical time frame.

Because `escrowDuration` is immutable per policy and users often rely on pre-existing policies, this acts as an effectively permanent lock on their liquidity.

Recommendation

Enforce a sane upper bound when creating policies.

Resolution

FastLane Team: The issue was resolved in PR#554. This is a design choice of the system, specifically to support restaking usecases built on top of ShMonad policies which require explicit approval to release funds, and should not be circumvented by a finite escrow duration.

D-06 | Redundant realTotalSupply() Function

Category	Severity	Location	Status
Best Practices	Discussion	FLERC20.sol: 56-57	Acknowledged

Description

Both `totalSupply()` and `realTotalSupply()` return identical values by calling the same internal function `_realTotalSupply()`.

Unlike previous versions where these functions returned different values, they are now redundant. This adds unnecessary code and potential confusion for integrators.

```
function totalSupply() public view returns (uint256) {
return _realTotalSupply();
}
function realTotalSupply() external view returns (uint256) {
return _realTotalSupply(); // @audit why two supplies only one is fine
}
```

Recommendation

Consider removing the `realTotalSupply()` function entirely and rely solely on the standard `totalSupply()` function for consistency with `ERC20` standards.

Resolution

FastLane Team: This is left in intentionally to support isolated staking (to a specific validator). This version of shMonad has removed the feature, but we intend to add it back in a future upgrade.

D-07 | removePolicyAgent Assigns Last Agent As Prime

Category	Severity	Location	Status
Configuration	Discussion	Policies.sol: 872-873	Acknowledged

Description

In `_removePolicyAgent`, when the primary agent is removed, the function assigns whoever was last in the array as the new primary. This is arbitrary - the last agent may not be qualified or authorized for the primary role.

Recommendation

Consider passing flag for who should be next primary agent.

Resolution

FastLane Team: The design of the agent and policy system does not differentiate between “primary agents” and other agents and there is no intention to differentiate between the two. We added a “primary agent” to save gas as it is included in the initial SLOAD of the Policy struct while other agents would require another SLOAD in the modifier check.

D-08 | yieldOriginator Parameter Can Be Spoofed

Category	Severity	Location	Status
Informational	Discussion	FLERC4626.sol: 55-56	Acknowledged

Description

The `yieldOriginator` address is user-provided input in `boostYield()`, allowing anyone to claim yield came from any address. This could confuse UIs, indexers, or analytics tracking yield sources

Recommendation

Consider validating `yieldOriginator` is `msg.sender` or beware of this risk associated.

Resolution

FastLane Team: This is a design choice of the protocol. `yieldOriginator` is for our own internal tracking via events, and is intended to be the bundler or DAppControl address of the Atlas metacall that generated yield. Atlas is not part of this audit, but you can learn more about Atlas here: https://www.atlasevm.com/

D-09 | Revenue Offset Creates Friction For Withdrawals

Category	Severity	Location	Status
Informational	Discussion	FLERC4626.sol: 321-322	Acknowledged

Description

The `_recentRevenueOffset()` mechanism deducts recent revenue from equity calculations during withdrawals, meaning legitimate users who need to withdraw after revenue events receive less value. While this protects against JIT attacks, it creates UX friction for normal innocent users.

Recommendation

Be aware this is a tradeoff between security and user experience. Consider documenting this behavior clearly for users.

Resolution

FastLane Team: This is intentional design of the system and we will make it clearly documented to users of the protocol.

D-10 | Escrow Period Bypassed

Category	Severity	Location	Status
Informational	Discussion	Policies.sol: 674-675	Acknowledged

Description

When a user requests uncommit (triggering escrow period), agents can still immediately use those uncommitting funds via `_spendFromCommitted()` without checking if escrow has completed.

Potentially confusing users who expect their funds to be locked during escrow and later available for withdrawal.

Recommendation

Consider documenting that escrow only restricts users, not agents

Resolution

FastLane Team: This is intentional design of the system and we will make it clearly documented to users of the protocol.

D-11 | depositAndCommit No Deposit Share Return

Category	Severity	Location	Status
Best Practices	Discussion	/src/shmonad/Policies.sol: 61	Resolved

Description

While the `deposit` function returns the number of shares minted for the amount of MON sent. The `depositAndCommit` function which does the same but committing the shares afterwards doesn't.

Leading to difficulties for other integrators to know the number of minted shares.

Recommendation

Return `sharesMinted` as part of the function.

Resolution

FastLane Team: The issue was resolved in PR#556.

D-12 | Atomic Pool Init Defaults

Category	Severity	Location	Status
Best Practices	Discussion	/src/shmonad/AtomicUnstakePool.sol: 22	Acknowledged

Description

`__AtomicUnstakePool_init()` silently applies the hard-coded fee curve whenever both mRay and cRay are zero.

On a fresh proxy that behavior is fine, but it also means you can't feed custom parameters during initialization, the owner must overwrite the defaults immediately after the upgrade if he was to change it.

Recommendation

Consider accepting initializer inputs or documenting the post-init step.

Resolution

FastLane Team: The protocol has been deployed, so this function is no longer relevant.

D-13 | Malicious Coinbase Address Can Lead To DOS

Category	Severity	Location	Status
Gas Griefing	Discussion	src/shmonad/StakeTracker.sol: 654	Resolved

Description

`addValidator(uint64 validatorId, address coinbase)` lets the owner register any coinbase contract for a validator.

During `_crankValidator`, the system calls that coinbase via `ICoinbase(coinbase).process()` with all remaining gas (`src/shmonad/StakeTracker.sol: 684-699`).

Although wrapped in try/catch, the call isn't gas-limited—if the callee deliberately burns all gas (e.g., endless loop or heavy computation), the outer `_crankValidator` runs out of gas before reaching the catch, causing the entire crank transaction to revert.

Recommendation

Before adding a validator, vet the coinbase contract (or use an audited template) to ensure its hooks can't stall `crank()`. If you must interact with an untrusted implementation, wrap those calls in a bounded-gas/try-catch path so a misbehaving validator can't brick the epoch processing.

Resolution

FastLane Team: The issue was resolved in PR#590.

D-14 | N(delta) Wraps Arbitrary Offsets

Category	Severity	Location	Status
Validation	Discussion	src/shmonad/Storage.sol: 193	Acknowledged

Description

`Storage.N(int256 nDelta)` just adds `nDelta` to `s_admin.internalEpoch` inside an unchecked block and applies % `EPOCHS_TRACKED`. There's no guard that the caller stays within the tracked window.

Passing `nDelta` = 8, -9, or any other value simply wraps around the 8-slot ring and hands back a slot that might be in active use.

The codebase only calls `*_Ptr_N()` with small, hard-coded offsets ([-3 … +2]) today, but because the helper is exposed to the whole codebase and widely used, someone could add a future call with an unbounded or user-supplied delta.

If they overshoot, they'll silently read/write the wrong epoch bucket.

Recommendation

Document this risk properly or add bounds based on the number of epochs tracked.

Resolution

FastLane Team: We have no intention to add a user supplied delta to the system.

D-15 | sum(balanceOf) = totalSupply - ERC20 Invariant

Category	Severity	Location	Status
Informational	Discussion	FLERC20.sol: 19-20	Acknowledged

Description

Since a user's committed balance is excluded from `balanceOf`, it breaks the usual `ERC20` invariant that the sum of all user balances equals the total supply.

This won't cause any issues per se, but it's something to be aware of for analytics teams.

Recommendation

Consider documenting this

Resolution

FastLane Team: This is intentional design of the protocol and we will work with analytic or frontend teams to correctly account for user balances.

D-16 | Revenue-Based Stake Allocation Creates Barrier

Category	Severity	Location	Status
Informational	Discussion	StakeAllocationLib.sol: 61-72	Acknowledged

Description

The crank allocates queued stake to validators proportionally based on their earned/global revenue. This results in cold start problem initially.

When all validators are newly added, both validator and global earned revenue are zero, resulting in no stake allocation. However, this is solvable by sending validator rewards or boosting yield to jumpstart the system.

Now, once a group of validators has established their stake and revenue, new validators face significant barriers to entry:

1. New validator joins with `earnedRevenue = 0`

2. Receives 0 stake allocation: `0 / globalRevenue = 0%`

3. Without stake delegation, cannot earn organic revenue from validation alone.

4. Must get revenue by sending validator rewards or triggering boost yield.

Recommendation

Be aware of this game-theoretic dynamic when operating FastLane.

Resolution

FastLane Team: The system is live, so no longer relevant.

D-17 | Assumptions On Monad Staking Precompile

Category	Severity	Location	Status
Informational	Discussion	Global	Acknowledged

Description

The protocol's `PrecompileHelpers` and related contracts make various assumptions about the Monad staking precompile's behavior.

All testing is currently done against `MockMonadStakingPrecompile`, not the actual production precompile.

If the production Monad staking precompile behaves differently from the mock, it could result in issues,

Recommendation

Beware of the risk associated. Once precompile is in production, consider running test suite with it and reviewing it for compatibility with helpers contract.

Resolution

FastLane Team: The system is live on both mainnet and testnet running against the actual pre-compile.

D-18 | Dead Code In AccountingLib

Category	Severity	Location	Status
Best Practices	Discussion	AccountingLib.sol	Resolved

Description

The targetAtomicAllocatedDelta and targetStakedDelta functions in AccountingLib are not used.

Recommendation

Consider removing deadcode.

Resolution

FastLane Team: The issue was resolved in PR#584.

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	Global Pending Updates Affect Allocations	Logical Error	High	Resolved
M-01	Agent Withdraw Burn Mismatch	Logical Error	Medium	Resolved
D-01	MaxWithdraw Rounding Mismatch	Logical Error	Discussion	Acknowledged

H-01 | Global Pending Updates Affect Allocations

Category	Severity	Location	Status
Logical Error	High	StakeAllocationLib.sol: 88-89	Resolved

Description PoC

During the validator crank loop, validators are processed sequentially.

When for a certain validator a `undelegate()` is called, it immediately increments `s_globalPending.pendingUnstaking`.

This directly reduces `globalUnstakableAmount` (calculated as `stakedAmount - pendingStaking - pendingUnstaking`) for all subsequent validators in the same crank cycle.

Which affects their allocations and can cause underflow and revert.

The issue occurs because `queueForUnstake` is cached at the start of the crank, but `globalUnstakableAmount` is recalculated fresh for each validator using the live `s_globalPending` state.

When validator N undelegates, it increases `pendingUnstaking`, which decreases `globalUnstakableAmount` for validator N+1. If validator N+1's allocation is calculated assuming the new `globalUnstakableAmount`, the formula:

`_stakeAllocationDecrease = queueForUnstake * validatorAmountAvailableToUnstake / globalAmountAvailableToUnstake`

can produce a `_stakeAllocationDecrease` larger than expected, leading to underflow at: `targetValidatorStake = validatorEpoch_Last.targetStakeAmount - netAmount;`

Recommendation

Consider caching `pendingUnstaking` at the start of the crank cycle

Resolution

FastLane Team: The issue was resolved in PR#611.

M-01 | Agent Withdraw Burn Mismatch

Category	Severity	Location	Status
Logical Error	Medium	src/shmonad/ShMonad.sol:221	Resolved

Description

When `agentWithdrawFromCommitted` is called with `amountSpecifiedInUnderlying == false`, it converts the requested shares to gross assets and caps that gross via `_getGrossCappedAndFeeFromGrossAssets`.

If the cap triggers (low liquidity), `_assetsToReceive` reflects only the capped gross minus fee, but `_sharesToDeduct` stays equal to the original amount.

`_spendFromCommitted` then burns the full requested shares even though the pool delivered fewer assets, so the caller loses more shMON than the payout justifies

Recommendation

Recompute `_sharesToDeduct` from `_grossAssetsCapped` before calling `_spendFromCommitted`, or revert when `_grossAssetsCapped < _grossAssetsWanted` to avoid inconsistent burns.

Resolution

FastLane Team: The issue was resolved in PR#612.

D-01 | MaxWithdraw Rounding Mismatch

Category	Severity	Location	Status
Logical Error	Discussion	FLERC4626.sol: 359	Acknowledged

Description PoC

`maxWithdraw` calculates a user's withdrawable amount by converting their share balance through the forward path (shares → assets), which floors the result and then applies fees and caps.

When the user later calls `withdraw`, the contract reverses that value using `_previewWithdraw`, which follows the inverse path (net assets → gross assets → shares) and uses ceiling rounding.

Because the forward and inverse paths round differently, the amount returned by `maxWithdraw` can translate back into more shares than the user holds, causing `_burn` to revert with `ERC20InsufficientBalance`.

Recommendation

Align `maxWithdraw` with the path and rounding used in `_previewWithdraw` so the returned amount never maps back to more shares than the caller owns.

Resolution

FastLane 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