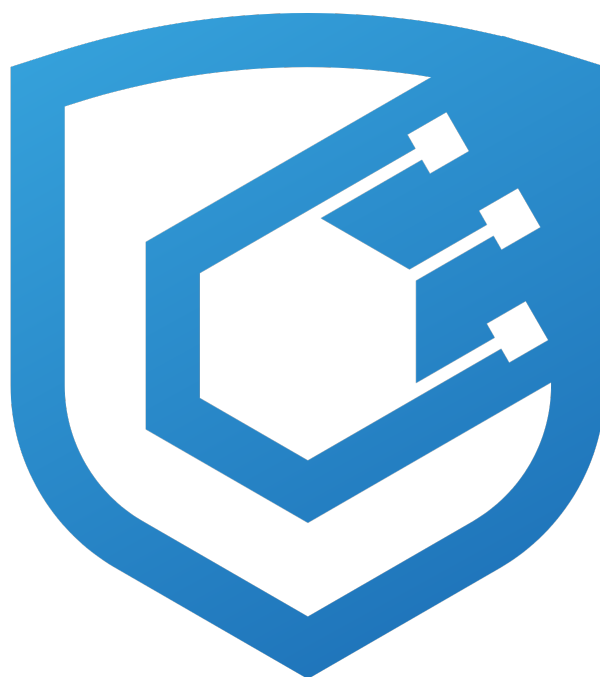




GlazeCorp Audit Report

Prepared by [Cyfrin](#)

Version 2.0

Lead Auditors

[Alix40](#)

[Arnie](#)

October 27, 2026

Contents

1	About Cyfrin	2
2	Disclaimer	2
3	Risk Classification	2
4	Protocol Summary	2
5	Audit Scope	2
6	Executive Summary	3
6.1	Centralization Risks	3
6.2	Defensive Improvements	4
6.3	Protocol Invariants	4
6.3.1	FullRange.sol	4
6.3.2	FullRangeERC20.sol	5
6.3.3	Launchpad.sol	5
6.3.4	Mine.sol	6
6.3.5	Fund.sol	7
6.3.6	Resonance.sol	7
6.3.7	Bribe.sol	8
6.3.8	SignalCoin.sol	9
6.3.9	Auction.sol	9
6.3.10	Coin.sol	9
6.3.11	ResonanceRouter.sol and BribeRouter.sol	10
6.3.12	FundController.sol	10
6.3.13	Periphery.sol and OperationsPeriphery.sol	10
7	Findings	13
7.1	Low Risk	13
7.1.1	Decimals-blind 1e6 raw-unit Auction price floor with linear decay sells low-value lots of low-decimal payment tokens only at zero price	13
7.1.2	FullRangeDeployment::verify does not authenticate constructor state	16
7.1.3	FullRange::initializeAndSeed token minima do not protect seed share output	18
7.2	Informational	20
7.2.1	Launchpad::setProtocolFeeBps lets the owner raise the protocol fee to 200 bps, four times the maximum disclosed in SECURITY_NOTES.md	20
7.2.2	Economics and architecture docs describe the old V2 launch flow, an old launch signature and a 10% Bribe default	20
7.2.3	Mine::PREVIOUS_MINER_BPS NatSpec suggests the revenue destination receives everything after the miner share	21
7.2.4	Launchpad::CoinLaunchedFullRange omits the Mine, Resonance, ResonanceRouter and SignalCoin addresses	21
7.2.5	Auction constructor reports InitialPriceOutOfRange when minimumPrice is the invalid value	22
7.2.6	Launchpad::_seedPool reverts with FullRange.Slippage when its own minimumGenesis-Shares check fails	22
7.2.7	creators-launching.md misstates Mine::_allocatePayment's disabled creator fee routing	22
7.3	Gas Optimization	24
7.3.1	FullRange looks up the same wrapper several times per liquidity operation	24

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the in-scope code being audited. Unless the scope explicitly notes otherwise, the review is limited to the application layer and does not extend to the correctness of API/SDK/execution environment implementations, compilers, standard and third-party libraries, underlying operating systems and hardware.

3 Risk Classification

	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

GlazeCorp is a token launchpad where each launched Coin gets its own self-funding on-chain treasury. In one transaction, `Launchpad::launch` deploys a Coin, an ownerless Fund, a Mine, a Resonance signal market with its `ResonanceRouter`, a `SignalCoin` and a creator-owned `FundController`. It then seeds a Coin/base full-range Uniswap V4 position through the `FullRange` hook and permanently locks the initial LP shares.

Mine runs a single-slot Dutch auction paid in USDG. The winning miner earns Coin at a rate fixed for its tenure until it is replaced. The outgoing miner gets a claim for 80% of the replacement payment, optional creator and protocol fees are credited as claims, and the remainder is sent to the revenue destination.

Resonance streams that revenue to strategies according to the Coin signaled through `SignalCoin`. Each strategy is an Auction that sells its accumulated USDG for a payment token. The payment goes to the Fund, except for an optional share sent to the strategy's `BribeRouter`, which forwards it to a Bribe that rewards the strategy's signalers. Coin holders can redeem a selected in-kind basket of Fund assets pro rata to their share of `Mine::effectiveTotalSupply`.

`FullRange` is a shared V4 hook that manages one full-range position per ERC-20 wrapper (`FullRangeERC20`), with one raw share per unit of native liquidity. Wrappers for the same currency pair and fee share a `PoolId` but keep independent positions and fee checkpoints. Realized swap fees are held as ERC-6909 claims. They are split 70/30 between the wrapper's creator recipients and the protocol recipient, or go entirely to the creator recipients while no protocol recipient is set.

All contracts are non-upgradeable. Each launch's Coin, Mine, Resonance and LP wrapper are administered by the creator through `FundController`. The Launchpad and `FullRange` owners only control global protocol fee settings.

5 Audit Scope

The audit scope was limited to the following files:

```
packages/contracts/src/core/Auction.sol
packages/contracts/src/core/AuctionFactory.sol
packages/contracts/src/core/Bribe.sol
packages/contracts/src/core/BribeFactory.sol
packages/contracts/src/core/BribeRouter.sol
packages/contracts/src/core/Coin.sol
packages/contracts/src/core/Fund.sol
packages/contracts/src/core/FundController.sol
packages/contracts/src/core/Mine.sol
packages/contracts/src/core/Resonance.sol
packages/contracts/src/core/ResonanceRouter.sol
packages/contracts/src/core/SignalCoin.sol
packages/contracts/src/fullrange/FullRange.sol
packages/contracts/src/fullrange/FullRangeERC20.sol
packages/contracts/src/fullrange/FullRangeERC20Deployer.sol
packages/contracts/src/fullrange/FullRangeMath.sol
packages/contracts/src/launch/CoinDeployer.sol
packages/contracts/src/launch/ControllerDeployer.sol
packages/contracts/src/launch/FundDeployer.sol
packages/contracts/src/launch/Launchpad.sol
packages/contracts/src/launch/MineDeployer.sol
packages/contracts/src/launch/ResonanceDeployer.sol
packages/contracts/src/launch/RouterDeployer.sol
packages/contracts/src/launch/SignalDeployer.sol
packages/contracts/src/periphery/OperationsPeriphery.sol
packages/contracts/src/periphery/Periphery.sol
```

6 Executive Summary

Over the course of 10 days, the Cyfrin team conducted an audit on the [GlazeCorp](#) code provided by [GlazeCorp](#). The findings consist of 3 Low severity issues with the remainder being informational and gas optimizations.

6.1 Centralization Risks

The Launchpad owner sets the global mining protocol fee (`Launchpad::setProtocolFeeBps`, 1-200 bps) and its recipient. Every existing Mine reads the current rate on each paid replacement, so a change applies to all launches at once and comes out of the revenue destination's share.

The FullRange owner sets the recipient of the 30% protocol share of swap fees (`FullRange::setProtocolFeeRecipient`). Fees are paid to the recipient set at claim time, so a change also redirects unclaimed balances. Setting it to zero makes existing protocol balances payable to each wrapper's creator recipients, and future fees then go entirely to them.

Each launch's creator owns its `FundController` and through it can:

- change the Mine creator fee (1-200 bps) and its recipient
- replace the Mine revenue destination with any contract, since `Mine::setRevenueRouter` only checks for deployed code. This can redirect all future mining revenue away from the launch's Resonance and Fund.
- add strategies paid in any deployed token other than `SignalCoin`, within the `Auction` parameter bounds, and kill any strategy except the last live one. In managed mode the creator also controls who may signal, so it can steer revenue to a strategy paid in a token it can mint and buy that strategy's USDG with it.
- set the Bribe share (0-2,000 bps), revoke signal managers and force-remove their signal, and permanently open signaling to the community
- change the LP wrapper's creator fee recipients, which also redirects unclaimed creator balances, including those payable to the Fund

- transfer ownership of the Coin, Mine, Resonance and LP wrapper, change the Coin's metadata URI and add reward tokens to a strategy's Bribe

The Fund is ownerless. Its non-Coin assets can only leave through pro-rata redemption by Coin holders, and any Coin it holds can only be burned. No owner can mint Coin outside Mine, change a Mine's emission schedule, move LP principal or locked genesis shares, pause the system or upgrade any contract.

6.2 Defensive Improvements

- `Launchpad::setProtocolFeeBps` and `FullRange::setProtocolFeeRecipient` take effect immediately for every existing launch. A timelock would give users time to react to fee changes.
- `Mine::setRevenueRouter` accepts any contract. Restricting it to the launch's `ResonanceRouter` or `Fund` would remove the revenue diversion described above.
- `Launchpad::launch` accepts a nonzero `swapFeeRecipient` when `receiveSwapFees` is false. It ignores that address but still emits it in `CoinLaunchedFullRange`. Rejecting this combination would keep the event consistent with the wrapper's recipients.

6.3 Protocol Invariants

The following properties must hold for the in-scope contracts. They are derived from the supplied documentation, the known-issue register and the audited source. Each property holds at the boundaries of completed external calls; intermediate state that a token callback can observe during a call is not promised.

6.3.1 FullRange.sol

ID	Property
FR-01	For every currency <code>c</code> , the hook's <code>PoolManager</code> ERC-6909 claim balance of <code>c</code> is at least the sum over all registered wrappers <code>w</code> of <code>creatorClaimable[w][c]</code> plus <code>protocolClaimable[w][c]</code>
FR-02	For every registered wrapper <code>w</code> , <code>FullRangeERC20::totalSupply</code> of <code>w</code> equals the liquidity of the native position owned by the hook at <code>(TICK_LOWER, TICK_UPPER, FullRangeERC20::positionSalt)</code> in the <code>PoolId</code> of <code>w</code>
FR-03	Every <code>unlockCallback</code> leaves the hook with zero open currency deltas: principal paid in equals the principal owed, principal taken equals the principal owed, and ERC-6909 claims minted equal the fees realized in that call
FR-04	Fees realized for wrapper <code>w</code> in currency <code>c</code> by any position update are credited exactly once, and the sum of the creator-role and protocol-role credits equals the realized fee amount
FR-05	For each wrapper and currency, the cumulative creator-role credit from realizations made while <code>protocolFeeRecipient</code> is nonzero equals the floor of 70% of the cumulative fees realized in those realizations, and the protocol role receives exactly the complement; realizations made while it is zero credit the creator role in full
FR-06	<code>claim</code> decreases exactly one role balance of one wrapper and currency by the paid amount, burns the same amount of the hook's claims, and delivers that amount to the role's current destination: the wrapper's creator recipient for that currency, <code>protocolFeeRecipient</code> for the protocol role, or the creator recipient when <code>protocolFeeRecipient</code> is zero
FR-07	No operation on wrapper A changes wrapper B's share supply, position liquidity, role balances or fractional creator remainder
FR-08	<code>removeLiquidity</code> burns exactly the requested shares from the caller, removes exactly that much liquidity from the wrapper's position, and pays the caller's chosen recipient only the principal that liquidity releases, never fees

ID	Property
FR-09	<code>addLiquidity</code> and <code>initializeAndSeed</code> mint shares equal to the liquidity added, less <code>MINIMUM_LIQUIDITY</code> minted to the dead address on a wrapper's first funding. Net shares never fall below the caller's <code>sharesMin</code> , and the amount pulled of each currency is never above the caller's desired or maximum amount nor below the caller's minimum
FR-10	Every V4 pool whose hook is <code>FullRange</code> was initialized by the hook, has an immutable fee within <code>[MIN_LP_FEE, MAX_LP_FEE]</code> and tick spacing <code>TICK_SPACING</code> , and receives liquidity additions only from the hook
FR-11	Only a wrapper's <code>FullRangeERC20::creator</code> can first-fund it, and <code>initialized</code> of a wrapper changes only from false to true
FR-12	<code>create</code> moves no tokens, changes no existing wrapper's state, and registers a wrapper only for sorted, code-bearing currencies whose decimals are at most 18
FR-13	Liquidity and fee operations on a wrapper move only that wrapper's two currencies, and every amount they pay out is debited only from that wrapper's position liquidity or that wrapper's role balances
FR-14	A holder's <code>removeLiquidity</code> never requires a fee payout to any recipient to succeed
FR-15	<code>quoteDeposit</code> , <code>quoteSeed</code> and <code>quoteWithdraw</code> return the amounts and shares that the matching execution produces against the same pool state
FR-16	Changing <code>protocolFeeRecipient</code> or a wrapper's creator recipients changes only the destination of existing role balances, never their amounts
FR-17	For each wrapper and currency, when the same fees are realized under the same protocol-fee regime (<code>protocolFeeRecipient</code> nonzero, or zero, at every realization), splitting their realization across <code>collect</code> calls at different frequencies never shifts value between the creator role and the protocol role beyond the fractional carry

6.3.2 FullRangeERC20.sol

ID	Property
FRE-01	Only the wrapper's hook can change <code>totalSupply</code>
FRE-02	<code>creatorRecipient0</code> and <code>creatorRecipient1</code> are never the zero address
FRE-03	Only the wrapper's current owner can change its creator recipients, and ownership of the wrapper grants no authority over shares or principal

6.3.3 Launchpad.sol

ID	Property
LP-01	After a successful launch, <code>Launchpad</code> holds none of that launch's Coin or LP shares, holds no more base token than before the launch, and has no remaining allowance from that launch to the hook or the <code>SignalCoin</code>
LP-02	After a successful launch, all initial LP shares of the launch wrapper, including the hook's minimum shares, are held by the dead address, and the recorded <code>LaunchRecord.genesisShares</code> equals that total

ID	Property
LP-03	Coin minted during a launch equals the Coin deposited into the position plus the burned seed dust plus the creator grant, and at launch end the Coin's minter is that launch's Mine with <code>Coin::minterLocked true</code>
LP-04	Base pulled from the creator during a launch equals the base deposited into the position plus the base refunded to the creator
LP-05	After a successful launch, Coin, Mine, Resonance and the launch wrapper are owned by that launch's FundController, SignalCoin has no owner, and the FundController's owner is the launching creator
LP-06	At completion of launch, the recorded strategies are registered in that launch's Resonance in the order LP wrapper, then the additional strategies, with no duplicate payment token. The recorded creator signal amounts equal the creator's signal in each strategy at that point. Later signal changes do not alter the record
LP-07	launchCount increases by exactly one per successful launch and launchIdByCoin maps each launched Coin to its own launch record
LP-08	At launch end, both creator recipients of the launch wrapper are the launch's Fund, unless the creator opted to receive swap fees, in which case both are the launch's nonzero swapFeeRecipient. The launch's Mine creator fee recipient is the launch's miningFeeRecipient
LP-09	A launch mints at most MAX_COIN_SEED_AMOUNT Coin for the seed and at most MAX_CREATOR_COIN_AMOUNT Coin for the creator grant
LP-10	A successful launch receives at least the caller's LaunchConfig.minimumGenesisShares net seed shares, excluding the hook's FullRange::MINIMUM_LIQUIDITY, before locking them
LP-11	advanceLaunchNonce succeeds only for the caller's current expected nonce, increases only that caller's creatorLaunchNonce by exactly one, and moves no assets
LP-12	The launch deployers and the shared AuctionFactory and BribeFactory retain no administrative authority over the components they deploy, and each Auction or Bribe a factory deploys is bound to the contract that called the factory, which in a launch is that launch's Resonance
LP-13	After a successful launch, a nonzero creator grant is delivered entirely to the launching creator: without auto-signaling it is minted to the creator's wallet; with auto-signaling it is escrowed in SignalCoin with equal receipts and strategy weights credited to the creator, following LaunchConfig.creatorSignalAmounts, or entirely on the LP strategy when that array is empty
LP-14	A launch succeeds only at or before the caller's LaunchConfig.deadline

6.3.4 Mine.sol

ID	Property
MN-01	Absent issuer confiscation, Mine's USDG balance is at least totalClaimablePayments, with equality absent unsolicited USDG transfers and absent configurations that make the Mine itself the revenue destination, a fee recipient or a miner
MN-02	totalClaimablePayments equals the sum of claimablePayment over all accounts
MN-03	For every paid replacement, the outgoing-miner credit is floor(payment x PREVIOUS_MINER_BPS / BPS) when the slot was occupied and zero otherwise, the creator and protocol credits are each floor(payment x rate / BPS) at the rate and recipient read before the payment's token interactions and zero when that recipient is zero, and the revenue deposit is the payment minus those three credits

ID	Property
MN-04	<code>effectiveTotalSupply</code> equals Coin total supply plus the active tenure's accrued unminted emission, and a replacement leaves it unchanged within the block
MN-05	Each settled tenure mints exactly its locked rate times its duration to its miner, and <code>totalMined</code> equals the sum of settled tenure emissions
MN-06	The prospective rate never increases over time and never falls below <code>TAIL_TPS</code> , and an occupied tenure's rate never changes before its replacement
MN-07	A replacement succeeds only with the current epoch identifier, at a price no greater than the caller's maximum, at or before the deadline, and increments the epoch identifier by exactly one
MN-08	Recorded payment claims are paid only to their recorded beneficiary and only in full, and fee-rate, recipient or revenue-destination changes never alter recorded claims
MN-09	The next starting price equals twice the paid price clamped to <code>[MIN_INITIAL_PRICE, MAX_INITIAL_PRICE]</code> , and the price decays linearly to zero over <code>PRICE_DECAY_PERIOD</code>
MN-10	Every <code>Mine</code> permanently retains its selected initial rate of 0.01-1,024 Coin per second, halving period of 7-1,460 days and tail rate of 0.001-1,024 Coin per second, with the initial rate at least the tail rate. Its prospective rate is $\max(\text{TAIL_TPS}, \text{floor}(\text{INITIAL_TPS} / 2^h))$, where h is the number of completed halving periods since deployment, and both rates and the result are in raw 18-decimal Coin units per second
MN-11	<code>creatorFeeBps</code> and <code>Launchpad::protocolFeeBps</code> always lie within 1-200 basis points, and <code>Resonance::bribeBps</code> never exceeds <code>Resonance::MAX_BRIBE_BPS</code>

6.3.5 Fund.sol

ID	Property
FD-01	A successful redemption of x Coin pays each selected token exactly $\text{floor}(B \times x / S)$ raw units, where B is that token's pre-burn <code>Fund</code> balance and S is the pre-burn <code>Mine::effectiveTotalSupply</code> , and reduces effective supply by exactly x
FD-02	Redeeming x and then y Coin never pays more of any token in total than redeeming $x + y$ in one call against the same balances and supply
FD-03	No in-scope code path moves <code>Fund</code> 's non-Coin assets other than <code>redeem</code> paying the receiver the redeeming holder selects, and <code>Fund</code> -held Coin leaves only by burning
FD-04	A redemption whose deadline has passed or whose computed payout for any selected token is below its minimum burns no Coin and transfers nothing
FD-05	A successful <code>redeem</code> uses a nonempty basket of nonzero, non-Coin token addresses in strictly increasing order, with one minimum per token and a nonzero receiver other than the <code>Fund</code>

6.3.6 Resonance.sol

ID	Property
RS-01	<code>totalSignalWeight</code> equals the sum of <code>Bribe::totalSignalWeight</code> over the paired Bribes of all live strategies
RS-02	Absent issuer confiscation, <code>Resonance</code> 's USDG balance is at least the sum over registered strategies of <code>earnedRevenue</code> plus <code>remainingRevenue</code> plus the saved rate-division remainder

ID	Property
RS-03	The global revenue index never decreases, and no strategy's paid index exceeds the global index
RS-04	A strategy accrues revenue only for time it was live with nonzero weight, never more than its weight's proportional share of the revenue emitted in that time, and a killed strategy keeps the revenue accrued before the kill
RS-05	Every change to a strategy's signal weight is preceded, in the same transaction, by a checkpoint of that strategy's revenue at its prior weight
RS-06	Only <code>SignalCoin</code> changes <code>Bribe::signalWeightOf</code> and <code>Bribe::totalSignalWeight</code> , <code>totalSignalWeight</code> changes otherwise only by removing a killed strategy's weight, only the bound <code>ResonanceRouter</code> notifies revenue, and <code>lifetimeRevenueNotified</code> increases only by fresh notified revenue and never exceeds <code>MAX_LIFETIME_REVENUE_AMOUNT</code>
RS-07	While community signaling is closed only approved managers add signal, opening community signaling is irreversible, and a forced removal applies only to a revoked manager in managed mode and returns that account's <code>Coin</code> to it
RS-08	Once a strategy is registered, at least one strategy stays live
RS-09	No in-scope code path moves <code>Resonance</code> 's <code>USDG</code> other than <code>distributeRevenue</code> , which pays a registered strategy's accrued revenue only to that strategy's own <code>Auction</code> , zeroes only that strategy's accrued revenue, and stays available for killed strategies
RS-10	Each successful revenue notification restarts the seven-day stream with a new rate and saved remainder such that the new rate times <code>REWARD_DURATION</code> plus the new remainder equals the fresh funding plus the previously remaining scheduled revenue plus the previous remainder, and leaves accrued revenue unchanged
RS-11	A killed strategy never becomes live again and accepts no new signal or future revenue
RS-12	<code>resonanceRouter</code> and <code>SignalCoin::resonance</code> are each set at most once, from zero to a nonzero contract, only by the current owner, and never change afterwards

6.3.7 Bribe.sol

ID	Property
BR-01	Absent issuer confiscation, for each reward token, <code>Bribe</code> 's balance is at least the sum over accounts of <code>earned</code> plus <code>remainingReward</code> plus the saved rate-division remainder
BR-02	<code>totalSignalWeight</code> equals the sum of <code>signalWeightOf</code> over all accounts
BR-03	Every change to an account's signal weight is preceded, in the same call, by a checkpoint of every registered reward token for that account at its prior weight
BR-04	Each reward token's index never decreases, and <code>lifetimeRewardNotified</code> never exceeds <code>MAX_LIFETIME_REWARD_AMOUNT</code>
BR-05	Rewards are paid only to the earning account, only when that account or <code>Resonance</code> initiates the claim
BR-06	The reward-token registry is append-only, holds at most <code>MAX_REWARD_TOKENS</code> tokens and has no duplicates
BR-07	For each reward token, an account accrues rewards only for time it held signal weight, never more than its weight's proportional share of the reward streamed in that time
BR-08	<code>claimReward</code> checkpoints and pays only the selected reward token and never requires a transfer of any other reward token

ID	Property
BR-09	Each successful reward notification restarts that token's seven-day stream with a new rate and saved remainder such that the new rate times <code>REWARD_DURATION</code> plus the new remainder equals the fresh funding plus that token's previously remaining scheduled reward plus its previous remainder, and leaves accrued rewards unchanged

6.3.8 SignalCoin.sol

ID	Property
SC-01	SignalCoin's Coin balance is at least its total supply
SC-02	For each account, its SignalCoin balance equals the sum of its <code>Bribe::signalWeightOf</code> over the paired Bribes of all registered strategies
SC-03	SignalCoin balances never move between two nonzero addresses
SC-04	An account's withdrawal of its signal never depends on any Bribe reward payout, or on any reward-token transfer other than SignalCoin returning that account's own escrowed Coin
SC-05	After a canonical launch completes, <code>initializeCreatorSignal</code> cannot be called
SC-06	Every signal addition escrows exactly the added Coin and creates equal receipts and strategy weight for its beneficiary, and every voluntary or forced removal destroys equal weight and receipts and returns exactly that Coin to the holder; killing a strategy, changing manager membership or opening community signaling never prevents an account from removing its own signal

6.3.9 Auction.sol

ID	Property
AU-01	A successful buy transfers to the chosen receiver the Auction's entire USDG balance after pulling its released Resonance revenue, and splits the payment so that the BribeRouter receives $\text{floor}(\text{payment} \times \text{Resonance}::\text{bribeBps read at the start of the call} / \text{BPS})$ and the Fund receives the rest
AU-02	A buy succeeds only with the current epoch identifier, at a price no greater than the caller's maximum, at or before the deadline, and with nonzero USDG for sale, and increments the epoch identifier by exactly one
AU-03	The next starting price equals $\text{floor}(\text{paid price} \times \text{priceMultiplier} / \text{PRICE_SCALE})$, clamped to $[\text{minimumPrice}, \text{ABSOLUTE_MAXIMUM_PRICE}]$

6.3.10 Coin.sol

ID	Property
CN-01	<code>totalSupply</code> equals <code>lifetimeMinted</code> minus <code>lifetimeBurned</code>
CN-02	Once <code>minterLocked</code> is true it stays true and only <code>minter</code> can mint, and Coin ownership grants no minting authority

6.3.11 ResonanceRouter.sol and BribeRouter.sol

ID	Property
RT-01	<code>ResonanceRouter::route</code> forwards its entire USDG balance only when live signal exists and that balance is at least the larger of the remaining scheduled revenue and <code>Resonance::REWARD_DURATION</code> , and otherwise moves nothing
RT-02	<code>BribeRouter::route</code> forwards its entire payment-token balance only when that balance is at least <code>Bribe::REWARD_DURATION</code> and at least the remaining scheduled reward, and otherwise moves nothing

6.3.12 FundController.sol

ID	Property
FC-01	Every <code>FundController</code> action calls only the <code>Coin</code> , <code>Mine</code> and <code>Resonance</code> bound at construction, or reads <code>IFullRangeLiquidityConfiguration::owner</code> of a supplied contract and, only when that returns the controller, calls <code>IFullRangeLiquidityConfiguration::setCreatorRecipients</code> or <code>IFullRangeLiquidityConfiguration::transferOwnership</code> on it, and no action moves LP principal, LP shares or Fund assets
FC-02	Privileged configuration actions and ownership renunciations of <code>FundController</code> , <code>Launchpad</code> , <code>FullRange</code> , <code>Mine</code> and <code>Resonance</code> succeed only for that contract's current owner, and only its current owner may initiate an ownership transfer; for <code>FundController</code> , <code>Launchpad</code> and <code>FullRange</code> , only the nominated pending owner may accept the transfer, and that pending owner gains administrative authority only after acceptance

6.3.13 Periphery.sol and OperationsPeriphery.sol

ID	Property
PH-01	<code>Periphery</code> and <code>OperationsPeriphery</code> never spend token balances they held before the call, and every allowance they grant during a call is zero when it ends
PH-02	A <code>Periphery</code> redemption either increases the receiver's balance of each selected token by at least its minimum or reverts entirely, including the Fund's maintenance burn and fee claims
PH-03	<code>Periphery::buy</code> forwards the caller's reviewed epoch, deadline and maximum unchanged; a failed optional routing never reverts a valid purchase except when the caller supplies insufficient total gas; and a failed purchase reverts all preceding work
PH-04	<code>Periphery::redeem</code> claims only fee roles currently payable to the selected launch's Fund
PH-05	<code>OperationsPeriphery</code> never pulls caller funds, never burns more Fund-held <code>Coin</code> than the reviewed amount, and a failed action does not revert the other actions except when outer gas is insufficient
PH-06	<code>Periphery::redeem</code> , when any fee role of the canonical launch wrapper is currently payable to the Fund, collects that wrapper's fees and claims each Fund-payable role and currency once, subject to <code>FullRange::claim</code> 's per-call cap, before burning Fund-held <code>Coin</code> , including claimed <code>Coin</code> , and then redeems; <code>Periphery::redeemWithoutFeeCollection</code> makes no fee collection or claim but keeps the maintenance burn and every other redemption protection
PH-07	<code>Periphery::mine</code> collects from the caller only the current mining price, never uses existing payment claims to fund it, and a failed outgoing-miner claim leaves the completed replacement and the unpaid claim intact except when outer gas is insufficient

ID	Property
PH-08	Both helpers resolve launch components only from their immutable <code>Launchpad</code> binding, act only on strategies registered with the selected launch's <code>Resonance</code> , and a purchased <code>Auction</code> 's <code>Resonance</code> , <code>Fund</code> and <code>USDG</code> must match that launch
PH-09	A successful <code>OperationsPeriphery::run</code> uses strategy and beneficiary lists that are each strictly increasing by numeric address, and so free of duplicates, with at most <code>OperationsPeriphery::MAX_ITEMS</code> entries each

Summary

Project Name	GlazeCorp
Repository	glazecorp
Commit	8adbf5e6182d...
Fix Commit	bc3d5d7205c6...
Audit Timeline	Oct 12th - Oct 23rd, 2026
Methods	Manual Review

Issues Found

Severity	Found	Resolved	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	3	3	0
Informational	7	7	0
Gas Optimizations	1	1	0
Total	11	11	0

Summary of Findings

[L-1] Decimals-blind 1e6 raw-unit Auction price floor with linear decay sells low-value lots of low-decimal payment tokens only at zero price	Resolved
[L-2] <code>FullRangeDeployment::verify</code> does not authenticate constructor state	Resolved
[L-3] <code>FullRange::initializeAndSeed</code> token minima do not protect seed share output	Resolved
[I-1] <code>Launchpad::setProtocolFeeBps</code> lets the owner raise the protocol fee to 200 bps, four times the maximum disclosed in <code>SECURITY_NOTES.md</code>	Resolved

[I-2] Economics and architecture docs describe the old V2 launch flow, an old launch signature and a 10% Bribe default	Resolved
[I-3] Mine::PREVIOUS_MINER_BPS NatSpec suggests the revenue destination receives everything after the miner share	Resolved
[I-4] Launchpad::CoinLaunchedFullRange omits the Mine, Resonance, ResonanceRouter and SignalCoin addresses	Resolved
[I-5] Auction constructor reports InitialPriceOutOfRange when minimumPrice is the invalid value	Resolved
[I-6] Launchpad::_seedPool reverts with FullRange.Slippage when its own minimumGenesisShares check fails	Resolved
[I-7] creators-launching.md misstates Mine::_allocatePayment's disabled creator fee routing	Resolved
[G-1] FullRange looks up the same wrapper several times per liquidity operation	Resolved

7 Findings

7.1 Low Risk

7.1.1 Decimals-blind 1e6 raw-unit Auction price floor with linear decay sells low-value lots of low-decimal payment tokens only at zero price

Description: `Auction::ABSOLUTE_MINIMUM_PRICE` is 1e6 raw payment-token units "regardless of token decimals". The constructor reverts when `Auction::minimumPrice` is below 1e6, and the initial price and every later epoch's starting price are at least `minimumPrice`, with a zero-price fill restarting exactly at `minimumPrice`. The price then decays linearly in `Auction::currentPrice: price(e) = P0 - floor(P0 * e / epochDuration)` for $e < \text{epochDuration}$, and 0 afterwards. The smallest nonzero price an epoch ever quotes is at $e = \text{epochDuration} - 1$ and equals $\text{ceil}(P0 / \text{epochDuration})$ raw units. Since $P0 \geq 1e6$, that last nonzero price is at least $\text{ceil}(1e6 / \text{epochDuration})$.

Launchpad fixes `AUCTION_EPOCH_DURATION = 1` days and sets `initialPrice = minimumPrice =` the creator's price, so a base-token Auction quotes prices of 24 raw units at $e = 86,398$, 12 raw units at $e = 86,399$ and 0 at $e = 86,400$ when the creator uses the lowest allowed price ($1e6 - \text{floor}(1e6 * 86,399 / 86,400) = 12$). Launchpad::_validate accepts any base token with `decimals() <= 18`, including 0, and `Resonance::addStrategy` adds no decimal check, so the floor is not scaled to the payment token. A lot worth less than the last nonzero price has no nonzero price at which a rational buyer clears it: waiting one more second makes it free. The next epoch restarts at `minimumPrice`, so the same outcome can repeat every epoch. Even when buyers fill at the first price at or below a lot's value, the shortfall against that value can reach one decay step (about $P0 / \text{epochDuration}$ raw units) per epoch, and a creator cannot set a lower floor.

The effect only matters for payment tokens with very few decimals and meaningful unit value (token worth \$20 per whole unit, canonical 1 day epoch, lowest price):

Payment-token decimals	Last nonzero price	Lots that clear only at zero
0	12 tokens (\$240)	under \$240
1	1.2 tokens (\$24)	under \$24
2	0.12 tokens (\$2.40)	under \$2.40
6 or more	at most 0.000012 tokens	negligible

Zero-price fills of fully decayed Auctions are documented as protocol behaviour (`SECURITY_NOTES.md`), and the owner's auction-behaviour confirmation (`docs/audit-todo.md`) accepts that the clock keeps decaying while no revenue is available. The 2026-10-07 owner review of the 1e6 bound stated it "constrains the starting-price floor, not the live fill price" and was reasoned at eight decimals. Through decay granularity the floor does bound the smallest nonzero live fill price, and at 0 to 2 decimals the bound is material, which that review did not cover.

Impact: For a Launchpad base-token Auction or a Resonance strategy Auction whose payment token has 0 to 2 decimals, any epoch whose USDG lot is worth less than $\text{ceil}(P0 / \text{epochDuration})$ raw payment units is sold for zero, even when demand and inventory are continuous. Fund, and the paired Bribe through its share, receive no payment for that strategy's USDG revenue in those epochs, and the buyer takes the lot for free. The loss per epoch is bounded by the lot value, below about 12 raw units of the payment token at the lowest price and the 1 day Launchpad epoch, 278 raw units at the 1 hour minimum epoch a Resonance strategy Auction may use, and about 1 raw unit at the 365 day maximum, and scales with the epoch's starting price $P0$. It is permanent per Auction because the floor and duration are immutable. Payment tokens with 6 or more decimals are unaffected in practice, and none of the token types the client currently expects (USDG, Coin, LP receipts, tokenized stocks) is known to use 0 to 2 decimals, although the supported asset list is not yet fixed.

Proof of Concept: The test registers a zero-decimal payment token through the real `Resonance::addStrategy` on the repository's `ProtocolFixture` and mints the USDG lot directly to the Auction in place of distributed revenue.

Save it as `packages/contracts/test/minimal/glazecorp-pocs/LowDecimalAuctionFloor.t.sol`:

```

// SPDX-License-Identifier: GPL-3.0-or-later
pragma solidity 0.8.26;

import { IERC20 } from "@openzeppelin/contracts/token/ERC20/IERC20.sol";

import { Auction } from "../../src/core/Auction.sol";
import { ProtocolFixture } from "../../utils/ProtocolFixture.sol";
import { MockERC20 } from "../../utils/Tokens.sol";

contract LowDecimalAuctionFloorTest is ProtocolFixture {
    function setUp() external {
        _deployProtocol();
    }

    function test_PoC_LowDecimalAuctionFloor() external {
        MockERC20 payment = new MockERC20("Zero Decimal Token", "ZDT", 0);

        // Register the strategy through the real Resonance with the Launchpad default shape:
        // one day epoch, 1.2x multiplier, lowest allowed floor and starting price
        Auction.Config memory config = Auction.Config(1e6, 1 days, 1.2e18, 1e6);
        (address strategy,, address bribeRouter) = resonance.addStrategy(IERC20(address(payment)),
        ↪ config);
        Auction auction = Auction(strategy);

        // lot of 150 USDG standing in for distributed revenue; a token worth 20 USDG makes 7 tokens the
        ↪ most a rational buyer pays
        uint256 lot = 150e6;
        usdg.mint(strategy, lot);
        uint256 start = auction.epochStartedAt();

        vm.warp(start + 86_398);
        assertEq(auction.currentPrice(), 24);
        vm.warp(start + 86_399);
        assertEq(auction.currentPrice(), 12);

        // the last nonzero price is above what the lot is worth to a buyer
        bytes memory expected = abi.encodeWithSelector(Auction.MaximumPaymentExceeded.selector, 12, 7);
        vm.prank(BOB);
        vm.expectRevert(expected);
        auction.buy(BOB, 0, block.timestamp, 7);

        // one second later the lot is free
        vm.warp(start + 86_400);
        assertEq(auction.currentPrice(), 0);
        vm.prank(BOB);
        uint256 paid = auction.buy(BOB, 0, block.timestamp, 0);

        assertEq(paid, 0);
        assertEq(usdg.balanceOf(BOB), lot);
        assertEq(payment.balanceOf(address(fund)), 0);
        assertEq(payment.balanceOf(bribeRouter), 0);
        assertEq(auction.initialPrice(), 1e6);
        assertEq(auction.epochId(), 1);
    }
}

```

Run from packages/contracts:

```
forge test --match-path test/minimal/glazecorp-pocs/LowDecimalAuctionFloor.t.sol -vv
```

Recommended Mitigation: Derive the lowest allowed `minimumPrice` from the payment token's decimals in the

Auction constructor: below 6 decimals, one whole token but at least 10 raw units, the smallest floor whose next starting price still rises under `Auction::MIN_PRICE_MULTIPLIER` (`floor(10 * 1.1) = 11`); otherwise, and for tokens whose `decimals()` is missing, reverts, returns malformed data or needs more than the 50,000 gas the new `DECIMALS_GAS_LIMIT` forwards, keep 1e6. The `Config` struct, the constructor signature, `AuctionFactory`, `Resonance::addStrategy` and `Launchpad` need no source change, because every `Auction`, including the three `Launchpad` defaults, is built through this constructor. The new floor is never above the current one, and the gas cap also bounds the data copied back, so a token whose `decimals()` burns its gas or returns oversized data adds only a bounded cost to deployment and every configuration accepted today stays accepted:

```
uint256 public constant PRICE_SCALE = 1e18;
+   /// @notice Smallest low-decimal floor whose next start still rises under `MIN_PRICE_MULTIPLIER`
+   uint256 public constant LOW_DECIMAL_MINIMUM_PRICE = 10;
+   /// @notice Gas forwarded to the payment token's optional `decimals()`
+   uint256 public constant DECIMALS_GAS_LIMIT = 50_000;
```

```
-   if (config.minimumPrice < ABSOLUTE_MINIMUM_PRICE || config.minimumPrice >
→  ABSOLUTE_MAXIMUM_PRICE) {
+   if (config.minimumPrice < _minimumPriceFloor(paymentToken_) || config.minimumPrice >
→  ABSOLUTE_MAXIMUM_PRICE) {
        revert MinimumPriceOutOfRange(config.minimumPrice);
    }
```

```
    if (nextPrice < minimumPrice) return minimumPrice;
}
+   /// @dev Lowest allowed `minimumPrice`: one whole token, but at least `LOW_DECIMAL_MINIMUM_PRICE`,
→  below 6 decimals,
+   /// otherwise `ABSOLUTE_MINIMUM_PRICE`
+   function _minimumPriceFloor(IERC20 token) private view returns (uint256) {
+       // A missing, reverting, malformed or gas-heavy `decimals()` keeps the absolute bound; the gas
→  cap also bounds
+       // the returned data copied here
+       (bool success, bytes memory data) =
+         address(token).staticcall{ gas: DECIMALS_GAS_LIMIT
→  }(abi.encodeWithSignature("decimals()"));
+       if (success && data.length >= 32) {
+         uint256 tokenDecimals = abi.decode(data, (uint256));
+         if (tokenDecimals < 6) return Math.max(10 ** tokenDecimals, LOW_DECIMAL_MINIMUM_PRICE);
+       }
+       return ABSOLUTE_MINIMUM_PRICE;
+   }
```

At the lowest allowed price, a 0-decimal Auction started at 10 raw units and a 2-decimal Auction started at 100 raw units both quote 1 raw unit at their last nonzero second, for any epoch duration from 1 hour, so a lot has an affordable nonzero quote unless it is worth less than one raw unit. At a higher starting price the last nonzero price remains `ceil(P0 / epochDuration)`. Zero-price fills of fully decayed Auctions are unchanged. The fix only lowers what a creator may choose: `Launchpad` passes the creator's price as both the initial and the minimum price, so a creator of a launch with a 0-decimal base token can now price the base Auction as low as 10 whole tokens.

Auction is not upgradeable, so the fix ships as a new `AuctionFactory` and a new `Launchpad` with its deployer set. Existing launch graphs and Auctions keep the old floor and decay behaviour, and no pending obligation moves.

Update the following:

- `ABSOLUTE_MINIMUM_PRICE` `NatSpec`, the row in `apps/web/content/docs/reference-constants.md` that says "regardless of token decimals", and the "minimum of 1,000,000 raw units" sentence in `packages/contracts/script/fullrange/README.md`, to describe the decimals-scaled floor
- `packages/contracts/test/minimal/Auction.t.sol` (`test_ConstructorEnforcesTheMinimumPriceWindow` keeps passing with its 18-decimal payment token): add constructor cases for 0, 2 and 6 decimal payment tokens, a token without `decimals()`, a token whose `decimals()` exhausts its gas or returns oversized data, and a case where a fill at the low-decimal floor raises the next starting price

- the web client's creator flows, which reject or raise prices below the same 1e6 floor (apps/web/lib/create/launch-config.ts, apps/web/lib/create/strategy-config.ts and apps/web/lib/strategy-assets/calculation.ts, with their tests), so that creators using the interface can choose the lower floor
- packages/contracts/test/minimal/glazecorp-pocs/LowDecimalAuctionFloor.t.sol: its unchanged configuration still reproduces the behaviour on the new build, because the fix lowers the permitted floor rather than any configured price. Demonstrating the new floor requires changing both configured prices to 10 and asserting that the last nonzero price is 1 and that a lot worth 7 tokens can be bought at a nonzero price no higher than 7 before the epoch ends

GlazeCorp: Fixed in commit [5edced1d](#).

Cyfrin: Verified. ABSOLUTE_MINIMUM_PRICE is now 10 raw units, so a 0-decimal Auction at the lowest price quotes 1 raw unit at its last nonzero second and the next start still rises under MIN_PRICE_MULTIPLIER (10 -> 11).

7.1.2 FullRangeDeployment::verify does not authenticate constructor state

Description: FullRangeDeployment::verify checks matching runtime code and visible configuration but does not authenticate that the reviewed FullRange::constructor created the supplied hook.

```
// FullRangeDeployment::verify
require(address(hook.poolManager()) == address(manager), "hook PoolManager mismatch");
require(hook.owner() == owner, "FullRange owner mismatch");
require(hook.protocolFeeRecipient() == protocolRecipient, "FullRange protocol recipient mismatch");

bytes memory expected = address(reference_).code;
_replaceWord(expected, address(reference_), address(hook));
_replaceWord(expected, address(reference_.wrapperDeployer()), wrapperDeployer);

require(keccak256(expected) == address(hook).codehash, "FullRange bytecode mismatch");
```

Runtime equality does not authenticate the constructor that returned that runtime or any storage written during deployment.

A substituted constructor may:

1. Deploy the genuine FullRangeERC20Deployer::constructor.
2. Store the expected owner and protocol recipient.
3. Preload a malicious wrapper in FullRange::isPool and FullRange::initialized.
4. Return the exact reviewed FullRange runtime with the expected immutables.

The wrapper does not need to appear in FullRange::allPools, because FullRange::_pool checks only isPool.

```
// FullRange::_pool
function _pool(address pool) private view returns (FullRangeERC20 token) {
    if (!isPool[pool]) revert UnknownPool();
    return FullRangeERC20(pool);
}
```

The malicious wrapper can return arbitrary values from FullRangeERC20::balanceOf and FullRangeERC20::totalSupply, return another wrapper's currencies and positionSalt, and make its burn function succeed without changing state.

FullRange::unlockCallback passes the wrapper-supplied salt to PoolManager::modifyLiquidity.

```
// FullRange::unlockCallback
FullRangeERC20 token = _pool(callback.pool);
PoolKey memory key = poolKey(callback.pool);

(BalanceDelta delta, BalanceDelta fees) = poolManager.modifyLiquidity(
```

```

    key,
    ModifyLiquidityParams(
        TICK_LOWER,
        TICK_UPPER,
        callback.liquidityDelta,
        token.positionSalt()
    ),
    ""
);

```

PoolManager identifies each position by the calling FullRange hook, tick range and supplied salt. A forged wrapper can therefore select a legitimate wrapper's native position under the same hook.

FullRange::removeLiquidity trusts the forged wrapper's balance, supply and burn behavior and transfers the selected principal to the attacker-controlled recipient.

```

// FullRange::removeLiquidity
FullRangeERC20 token = _pool(params.pool);

if (params.liquidity > token.balanceOf(msg.sender)) {
    revert InsufficientShares();
}

uint128 liquidityToRemove =
    _liquidityForShares(params.pool, params.liquidity);

OperationResult memory result =
    _modifyLiquidity(
        params.pool,
        -int256(uint256(liquidityToRemove)),
        params.to
    );

token.burn(msg.sender, params.liquidity);

```

VerifyLaunchpad::_verifyPoolFactory inherits the issue because it calls FullRangeDeployment::verify. DeployLaunchpad::run checks only that the supplied FullRange address contains code before permanently binding it.

The canonical DeployFullRange::run path is not affected because it deploys the exact reviewed init code through the reviewed deterministic helper. The issue applies when an independently supplied hook is accepted through VerifyFullRange::run or bound through DeployLaunchpad::run without proving its deployment provenance.

Impact: An attacker-controlled FullRange deployment can pass the current verifier while containing a hidden registered wrapper. That wrapper can remove native positions belonging to legitimate wrappers later created under the same hook, transferring their principal to the attacker. Previously uncollected fees realized by that removal are credited under the forged wrapper. The attacker can claim the creator allocation through attacker-controlled creator/recipient getters. Any enabled protocol allocation still follows the configured protocol recipient, and existing fee credits recorded under legitimate wrappers are not captured by this path.

Recommended Mitigation: Extend FullRangeDeployment::verify with the deterministic helper, its expected immutable owner, the deployment salt and the original constructor owner and protocol recipient as inputs. Authenticate the helper's exact runtime and owner before trusting its deployment records, then authenticate the constructor provenance:

```

+ bytes memory initCode = abi.encodePacked(
+     type(FullRange).creationCode,
+     abi.encode(manager, initialOwner, initialProtocolRecipient)
+ );
+ bytes32 initCodeHash = keccak256(initCode);
+
+ require(

```

```

+     helper.deployedInitCodeHash(salt) == initCodeHash,
+     "FullRange init code hash mismatch"
+ );
+
+ address expectedHook = address(
+     uint160(
+         uint256(
+             keccak256(
+                 abi.encodePacked(
+                     bytes1(0xff),
+                     address(helper),
+                     salt,
+                     initCodeHash
+                 )
+             )
+         )
+     )
+ );
+
+ require(address(hook) == expectedHook, "FullRange CREATE2 address mismatch");

```

Preserve all existing runtime and configuration checks. Keep the original constructor arguments separate from the current owner and protocol-recipient checks, since both settings can change after deployment.

`DeployLaunchpad::run` should perform the complete verification before beginning its broadcast. Where `FullRange` uses the same helper, `VerifyLaunchpad` can reuse its existing helper authentication. Update all verifier callers, environment templates and deployment documentation together.

This deliberately restricts acceptance to the authenticated helper/salt provenance. Otherwise genuine hooks deployed independently through another deployment path will be rejected.

Glazecorp: Fixed in commit [ea85d391](#).

Cyfrin: Verified. Fix follows recommendation.

7.1.3 `FullRange::initializeAndSeed` token minima do not protect seed share output

Description: `FullRange::initializeAndSeed` protects the token amounts consumed but does not let the creator specify a minimum number of net shares.

If another wrapper initializes the shared market first, `FullRange::quoteSeed` adopts that market price. Coarse token rounding can allow the changed price to consume the same exact token amounts while producing substantially less liquidity.

```

// FullRange::_addLiquidity
liquidity = FullRangeMath.liquidity(
    price,
    params.amount0Desired,
    params.amount1Desired
);

uint256 shares = first ? liquidity - MINIMUM_LIQUIDITY : liquidity;

OperationResult memory result =
    _modifyLiquidity(params.pool, int256(liquidity), address(0));

if (
    result.amount0 < params.amount0Min
    || result.amount1 < params.amount1Min
) revert Slippage();

if (first) token.mint(address(0xdead), MINIMUM_LIQUIDITY);
token.mint(params.to, shares);

```

For maxima and exact minima of $(2^{60}, 1)$:

```
At sqrtPriceX96 = 2^66:  
gross liquidity = 1,073,741,824  
token consumption = (2^60, 1)  
net creator shares = 1,073,740,824  
  
At sqrtPriceX96 = 2^47:  
gross liquidity = 2,048  
token consumption = (2^60, 1)  
net creator shares = 1,048
```

A creator reviewing a wrapper whose shared market is still uninitialized receives the first quote because `FullRangeMath::seedPrice(2^60, 1)` returns 2^{66} .

Before `FullRange::initializeAndSeed` executes, another account can create and fund a sibling wrapper for the same currencies. Funding that wrapper with $(2^{98}, 1)$ initializes the shared `PoolId` at 2^{47} .

The victim's exact token minima still pass, but `FullRange::_addLiquidity` permanently mints 1,000 of the resulting 2,048 shares to `0xdead`.

```
locked fraction = 1,000 / 2,048  
                = 48.828125%
```

An immediate withdrawal through `FullRange::removeLiquidity` returns only 589,971,551,185,534,975 raw `token0` to the victim. The difference of 562,949,953,421,312,001 raw `token0` consists of locked principal and native rounding dust.

The same rounding gap exists in `FullRange::addLiquidity`, but fewer shares at a different price do not by themselves prove proportional economic loss because one share represents one native liquidity unit. The concrete impact arises during first funding because the fixed 1,000-share lock becomes a material portion of the position.

`Launchpad::_seedPool` is separately protected by its caller-supplied `minimumGenesisShares`.

Impact: A creator calling `FullRange::initializeAndSeed` directly can have a material portion of its seed principal permanently locked even though execution consumes exactly the reviewed token amounts.

The demonstrated case locks 48.828125% of the position.

Recommended Mitigation: Add a caller-supplied minimum net share output to `FullRange::initializeAndSeed` and `FullRange::addLiquidity`.

```
+ if (shares < sharesMin) revert Slippage();
```

For `FullRange::initializeAndSeed`, compare `sharesMin` against the net creator shares after subtracting `MINIMUM_LIQUIDITY`.

Glazecorp: Fixed in commit [77f61f4e](#).

Cyfrin: Verified. Minimum share validation has been added.

7.2 Informational

7.2.1 Launchpad::setProtocolFeeBps lets the owner raise the protocol fee to 200 bps, four times the maximum disclosed in SECURITY_NOTES.md

Description: The Launchpad owner sets a single global protocol fee that every Mine applies to its mining payments. Launchpad::setProtocolFeeBps accepts any value up to 200 bps:

```
uint256 public constant MAX_PROTOCOL_FEE_BPS = 200;
...
function setProtocolFeeBps(uint256 feeBps) external onlyOwner {
    if (feeBps < MIN_PROTOCOL_FEE_BPS || feeBps > MAX_PROTOCOL_FEE_BPS) revert
    ↪ InvalidProtocolFeeBps(feeBps);
}
```

Each Mine reads the current rate on every paid replacement, so a change applies to all existing Mines at once:

```
protocolBps: launchpad.protocolFeeBps(),
```

SECURITY_NOTES.md, the document users read to judge the protocol's trust assumptions, discloses a much lower ceiling and a different model:

The Launchpad owner can change the global creator and protocol mining fee rate from 1 to 50 basis points of each payment, initially 50.

docs/audit-todo.md repeats this. In the code the creator fee is a separate per-Mine rate, also up to 200 bps, set through Mine::setCreatorFeeBps.

Impact: Users are told the protocol fee is at most 0.5% of each mining payment. The owner can raise it to 2% at any time for every existing Mine, and the extra comes out of the revenue destination's share.

Recommended Mitigation: Update SECURITY_NOTES.md and docs/audit-todo.md to describe the 1–200 bps global protocol fee and the separate 1–200 bps per-Mine creator fee.

Glazecorp: Fixed in commit [8d922b83](#).

Cyfrin: Verified. SECURITY_NOTES.md and docs/audit-todo.md now describe the 1–200 bps global protocol fee and the separate 1–200 bps per-Mine creator fee.

7.2.2 Economics and architecture docs describe the old V2 launch flow, an old launch signature and a 10% Bribe default

Description: Several design docs still describe the removed V2 launch flow and its defaults.

1. docs/economics.md "Genesis liquidity" still describes the V2 launch

Launchpad's owner sets genesisCoinAmount and genesisUsdgAmount together for future launches using setGenesisLiquidity. Defaults remain **1,000 Coin and 1 USDG** ...

The same section says the caller passes LaunchConfig(creatorFeeBps, expectedGenesisCoin, expectedGenesisUsdg), describes a "fresh pair from its immutable V2 factory", and gives an LP Auction price of $\sqrt{\text{product}} * 50$. setGenesisLiquidity, the V2 factory and that price formula no longer exist. Launchpad::launch now takes explicit baseSeedAmount, coinSeedAmount, minimumGenesisShares and lpAuctionPrice and seeds through Full-Range.

2. docs/economics.md and docs/revenue-model.md give a 10% community Bribe default

A launch with communitySignaling = true lets every Coin holder add signal and starts with a 1,000-bps Bribe share.

revenue-model.md works through an example "at defaults" with a 10% bribe share. For launches through Launchpad there is no contract default, because the creator's value is always applied:

```
Resonance(result.resonance).setBribeBps(config.bribeBps);
```

The 1,000 bps DEFAULT_BRIBE_BPS only applies to a standalone Resonance.

3. docs/architecture.md documents an old launch signature

Launchpad.launch(name, symbol, metadataURI, creatorFeeRecipient, communitySignaling, config) creates the graph atomically.

The actual function is launch(LaunchConfig calldata config).

Recommended Mitigation: Update these sections to match the current LaunchConfig and FullRange seeding flow.

Glazecorp: Fixed in commit [775bc9fd](#).

Cyfrin: Verified. economics.md now describes the current LaunchConfig and FullRange seeding with no V2 genesis flow, the Bribe share is the explicit bribeBps in both modes, revenue-model.md labels the 10% example as illustrative, and architecture.md shows Launchpad.launch(config).

7.2.3 Mine::PREVIOUS_MINER_BPS NatSpec suggests the revenue destination receives everything after the miner share

Description: The NatSpec on PREVIOUS_MINER_BPS reads:

```
///@dev The miner share is rounded down; the revenue destination receives the complete remainder.
uint256 public constant PREVIOUS_MINER_BPS = 8_000;
```

Read on its own, this says the revenue destination gets everything the outgoing miner doesn't. But Mine::_allocatePayment also takes out the creator and protocol fees first:

```
revenueAmount = paymentAmount - previousMinerAmount - creatorAmount - protocolAmount;
```

Recommended Mitigation:

```
-///@dev The miner share is rounded down; the revenue destination receives the complete remainder.
+///@dev The miner share is rounded down; the revenue destination receives the remainder after the
+///miner, creator and protocol shares.
```

Glazecorp: Fixed in commit [fdd1af7b](#).

Cyfrin: Verified. Implemented recommended mitigation.

7.2.4 Launchpad::CoinLaunchedFullRange omits the Mine, Resonance, ResonanceRouter and SignalCoin addresses

Description: The launch event carries some of the new graph's addresses, but not all of them:

```
event CoinLaunchedFullRange(
    uint256 indexed launchId,
    address indexed coin,
    address indexed originalCreator,
    address controller,
    address fund,
    address pool,
    address baseToken,
    bytes32 poolId,
    uint24 swapFee,
    LaunchConfig configuration
);
```

mine, resonance, resonanceRouter and signalCoin are missing, and no other Launchpad event carries them. The launch README says the event connects the snapshot "to its component addresses", but the current subgraph has to call getLaunch while indexing to find them.

Recommended Mitigation: Correct the README to say the remaining addresses come from getLaunch, or add them to the event.

Glazecorp: Fixed in commit [d7f886cf](#).

Cyfrin: Verified. The launch README now points to getLaunch for the Mine, Resonance, ResonanceRouter and SignalCoin addresses.

7.2.5 Auction constructor reports InitialPriceOutOfRange when minimumPrice is the invalid value

Description: The Auction constructor compares initialPrice against minimumPrice before it range-checks minimumPrice:

```
if (config.initialPrice < config.minimumPrice || config.initialPrice > ABSOLUTE_MAXIMUM_PRICE) {
    revert InitialPriceOutOfRange(config.initialPrice);
}
...
if (config.minimumPrice < ABSOLUTE_MINIMUM_PRICE || config.minimumPrice > ABSOLUTE_MAXIMUM_PRICE) {
    revert MinimumPriceOutOfRange(config.minimumPrice);
}
```

If initialPrice is fine but minimumPrice is above ABSOLUTE_MAXIMUM_PRICE, the first check fires and the revert blames initialPrice even though minimumPrice is the invalid value.

Recommended Mitigation: Range-check minimumPrice before comparing initialPrice against it.

Glazecorp: Fixed in commit [59f34156](#).

Cyfrin: Verified. minimumPrice is now range-checked before initialPrice is compared against it.

7.2.6 Launchpad::_seedPool reverts with FullRange.Slippage when its own minimumGenesisShares check fails

Description: Launchpad::_seedPool reuses FullRange's error for its own check:

```
// Launchpad retains its separate guarantee on the net genesis shares it receives.
if (r.genesisShares < minShares) revert FullRange.Slippage();
```

FullRange raises the same Slippage() in quoteSeed and in initializeAndSeed, both of which run during the same launch. When a launch reverts with Slippage(), the selector alone doesn't tell the creator which of the three checks failed.

Recommended Mitigation: Declare a Launchpad-specific error and use it for the genesis share check:

```
+error InsufficientGenesisShares(uint256 received, uint256 minimum);
...
-if (r.genesisShares < minShares) revert FullRange.Slippage();
+if (r.genesisShares < minShares) revert InsufficientGenesisShares(r.genesisShares, minShares);
```

Glazecorp: Fixed in commit [3fe68b95](#).

Cyfrin: Verified. Implemented recommended mitigation.

7.2.7 creators-launching.md misstates Mine::_allocatePayment's disabled creator fee routing

Description: The creator guide says:

With the fee off, its share goes to the fund.

Setting `creatorFeeRecipient` to zero disables future creator fee accrual. `_allocatePayment` creates a creator claim only when the snapshotted recipient is nonzero:

```
uint256 creatorAmount;
if (fees.creator != address(0)) {
    creatorAmount = Math.mulDiv(paymentAmount, fees.creatorBps, BPS);
    claimablePayment[fees.creator] += creatorAmount;
}

revenueAmount =
    paymentAmount - previousMinerAmount - creatorAmount - protocolAmount;
```

When the creator recipient is zero, no creator amount is calculated or subtracted. The amount that would otherwise have been the floored creator allocation therefore remains in `revenueAmount`, which `_collectAndDeposit` transfers to the snapshotted revenue destination:

```
usdg.safeTransfer(router, revenueAmount);
```

Canonical launches initially use `ResonanceRouter` as that destination, not `Fund`. The current Mine owner may later select another destination, including `Fund`. Under the default route, the revenue can contribute to `Fund` acquisitions only after separate routing, distribution and Auction purchases.

Impact: The guide misstates the immediate destination of the disabled allocation and omits that the revenue destination is configurable.

Recommended Mitigation: Replace the final sentence with:

With the creator fee off, no new creator claim accrues. That amount remains in the mining payment remainder sent to Mine's current revenue destination, initially `ResonanceRouter`, but configurable by the Mine owner.

Glazecorp: Fixed in commit [6b1e5b58](#).

Cyfrin: Verified, updates to documentation have been made.

7.3 Gas Optimization

7.3.1 FullRange looks up the same wrapper several times per liquidity operation

Description: `_pool` (the `isPool` check) runs several times for the same wrapper in one operation: four times for an add, four for a remove and six for a seed. Part of this is in the `unlock` callback, which resolves the wrapper and then calls `poolKey`, which resolves it again:

```
FullRangeERC20 token = _pool(callback.pool);  
PoolKey memory key = poolKey(callback.pool);
```

```
function poolKey(address pool) public view returns (PoolKey memory) {  
    FullRangeERC20 token = _pool(pool);  
    return _key(token.currency0(), token.currency1(), token.fee());  
}
```

`removeLiquidity` also resolves it again inside `_liquidityForShares`, and `addLiquidity` resolves it once more after `_addLiquidity` to emit `Deposited`.

Recommended Mitigation: Reuse the wrapper that has already been resolved, for example by building the key from `token` in the callback and passing `token` into `_liquidityForShares`:

```
FullRangeERC20 token = _pool(callback.pool);  
-PoolKey memory key = poolKey(callback.pool);  
+PoolKey memory key = _key(token.currency0(), token.currency1(), token.fee());
```

Glazecorp: Fixed in commits [9103d2dd](#) and [c8dbcf59](#).

Cyfrin: Verified. The resolved wrapper is now passed into the internal helpers, so a seed, add or remove does 2 `_pool` lookups instead of 6, 4 and 4.