

PAPERTRADE · SECURITY REVIEW

Papertrade Protocol Review.

A security review of Protocol Review.

CLIENT	PROJECT	AUDITORS	AUDIT WINDOW	FINAL REPORT
Papertrade	Protocol Review	Minato7namikazi	June 5 – October 9, 2026	October 9, 2026

§ 00

Contents.

83 pages: overview, confidence ranking, scope, 39 findings, disclaimer, and about Guardian.

§ 01	Overview	P. 03
§ 02	Confidence Ranking	P. 04
§ 03	Scope & Classifications	P. 05
	Files in Scope	P. 06
§ 04	Findings	P. 08
	Main Review · 39 findings	P. 10
§ 05	Disclaimer	P. 82
§ 06	About Guardian	P. 83

§ 01

Overview.

Guardian reviewed Protocol Review across 2 review rounds. The final report includes 2 High, 5 Medium, 10 Low, 22 Informational findings.

AUDIT FIRM	REPOSITORY
Guardian	https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445
CLIENT	METHODOLOGY
Papertrade	Smart contract · Manual · Main Review · Remediation Review
PROJECT	MAIN REVIEW COMMIT
Protocol Review	48e552b25032
PREPARED BY	REMEDIATION REVIEW COMMIT
Minato7namikazi	a467d52a9a2d
AUDIT WINDOW	
June 5 – October 9, 2026	
FINAL REPORT	
October 9, 2026	

1.1 Confidence Ranking

4 / 5

High Confidence

Guardian assigns a Confidence Ranking of 4 out of 5, combining finding severity, remediation outcome, and the reviewer’s assessment of latent risk. The full scale is on the next page.

1	2	3	4	5
VERY LOW CONFIDENCE	LOW CONFIDENCE	MODERATE CONFIDENCE	HIGH CONFIDENCE	VERY HIGH CONFIDENCE

1.2 Findings at a Glance

SEVERITY	TOTAL	PENDING	ACK	PARTIAL	RESOLVED
● Critical	0	0	0	0	0
● High	2	0	0	0	2
● Medium	5	0	0	0	5
● Low	10	0	7	0	3
● Informational	22	0	10	0	12
All severities	39	0	17	0	22

§ 02

Confidence Ranking.

Guardian assigns a project-level confidence score using the scale below. The ranking combines finding severity, remediation outcome, and the reviewer's assessment of latent risk.

CONFIDENCE RANKING	DEFINITION AND RECOMMENDATION	RISK PROFILE
5 Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. Recommendation: code is highly secure at time of audit.	0 High/Critical findings and few Low/Medium severity findings.
4 High Confidence ASSIGNED	Code is clean, well-structured, and adheres to best practices. Design patterns are sound and test coverage is strong. Recommendation: suitable for deployment after remediations; consider periodic review with changes.	0-1 High/Critical findings per engagement week and little to no Medium severity findings.
3 Moderate Confidence	Code is functional but shows gaps in defensive programming or test coverage. Recommendation: remediate the findings and re-review the changes before deployment.	Some High/Critical findings, with various Medium severity findings.
2 Low Confidence	Recurring issues suggest systemic gaps rather than isolated mistakes. Recommendation: remediate, then commission a follow-up review at a frozen commit.	Over 6 High and Critical findings across the engagement.
1 Very Low Confidence	The volume and severity of findings indicate the codebase is not ready for production. Recommendation: substantial rework, then a full re-review.	Over 12 High and Critical findings across the engagement.

§ 03

Scope & Classifications.

Guardian reviewed Protocol Review in

<https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445>. The scope includes 7,667 source lines across 29 files.

3.1 Severity Matrix

	IMP · HIGH	IMP · MED	IMP · LOW
LIK · HIGH	Critical	High	Medium
LIK · MED	High	Medium	Low
LIK · LOW	Medium	Low	Low

IMPACT · HIGH

Significant loss of assets, harm to a group of users, or core protocol functionality is disrupted.

IMPACT · MEDIUM

Small amount of funds can be lost, or ancillary functionality is affected. Users may experience reduced or delayed receipt.

IMPACT · LOW

Unexpected behavior in some functionality that is notable but does not meet the criteria for higher severity.

LIKELIHOOD · HIGH

Possible with reasonable assumptions that mimic on-chain conditions; cost is low compared to amount gained.

LIKELIHOOD · MEDIUM

Possible only in uncommon cases or requires a large amount of capital relative to disruption caused.

LIKELIHOOD · LOW

Unlikely to ever occur in production.

Files in Scope 1/2.

FILE	NSLOC	TOTAL
MAIN REVIEW		
contracts/script/DeployConfig.sol	30	77
contracts/script/DeployStaged.s.sol	510	627
contracts/script/GovernanceDeployer.sol	63	79
contracts/script/UpgradeableDeployer.sol	20	30
contracts/script/UpgradeContract.s.sol	123	136
contracts/src/auxiliary/BatchReader.sol	25	38
contracts/src/BatchExecutor.sol	115	143
contracts/src/BatchExecutorRelayerOnly.sol	66	90
contracts/src/Create2DepositProxy.sol	42	63
contracts/src/DepositProxyFactory.sol	33	48
contracts/src/Exchange.sol	995	1,639
contracts/src/ExchangeRelayerOnly.sol	43	65
contracts/src/ExchangeSessionFee.sol	57	72
contracts/src/governance/GovernanceTimelock.sol	133	162
contracts/src/governance/OneOfTwoMultisig.sol	96	123
contracts/src/PaperMathLib.sol	29	75
contracts/src/PaperStaking.sol	238	292
contracts/src/PaperStakingRelayerOnly.sol	89	122
contracts/src/PaperToken.sol	88	110
contracts/src/PaperTokenNonTransferable.sol	19	30
contracts/src/PaperTokenomics.sol	183	254
contracts/src/PriceOracle.sol	81	136
contracts/src/proxy/AdminUpgradeableProxy.sol	81	97
contracts/src/proxy/Initializable.sol	26	32
contracts/src/proxy/ProxyAdmin.sol	31	42
contracts/src/QueueLib.sol	436	702
contracts/src/SafeTokenLib.sol	26	34
contracts/src/SessionKeyManager.sol	152	238
contracts/src/SessionKeyManagerRelayerOnly.sol	19	30
REMEDIATION REVIEW		
contracts/script/DeployConfig.sol	30	77
contracts/script/DeployStaged.s.sol	523	642
contracts/script/GovernanceDeployer.sol	75	92
contracts/script/UpgradeableDeployer.sol	20	30
contracts/script/UpgradeContract.s.sol	123	136

Files in Scope 2/2.

FILE	NSLOC	TOTAL
contracts/src/auxiliary/BatchReader.sol	25	38
contracts/src/BatchExecutor.sol	98	126
contracts/src/BatchExecutorRelayerOnly.sol	66	90
contracts/src/Create2DepositProxy.sol	38	58
contracts/src/DepositProxyFactory.sol	40	57
contracts/src/Exchange.sol	1,020	1,726
contracts/src/ExchangeRelayerOnly.sol	43	65
contracts/src/ExchangeSessionFee.sol	45	57
contracts/src/governance/GovernanceTimelock.sol	133	165
contracts/src/governance/OneOfTwoMultisig.sol	89	116
contracts/src/PaperMathLib.sol	27	74
contracts/src/PaperStaking.sol	238	293
contracts/src/PaperStakingRelayerOnly.sol	89	122
contracts/src/PaperToken.sol	88	110
contracts/src/PaperTokenNonTransferable.sol	19	30
contracts/src/PaperTokenomics.sol	164	249
contracts/src/PriceOracle.sol	81	136
contracts/src/proxy/AdminUpgradeableProxy.sol	79	102
contracts/src/proxy/Initializable.sol	26	32
contracts/src/proxy/ProxyAdmin.sol	31	42
contracts/src/QueueLib.sol	437	719
contracts/src/SessionKeyManager.sol	152	243
contracts/src/SessionKeyManagerRelayerOnly.sol	19	30

§ 04

Findings.

Grouped by round. Visible IDs reset by severity within each round, so a finding is named the same way in the report as it was in the channel it was written in. Status is where each finding ended up; its own page says how.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
MAIN REVIEW					
H-01	ExchangeSessionFee lets queue drain fees	Logical Error	HIGH	RESOLVED	p. 10
H-02	Enqueued Wins Never Debit trackedLpUsd	Logical Error	HIGH	RESOLVED	p. 12
M-01	Solvent liquidation over-credits trackedLpUsd	Math	MEDIUM	RESOLVED	p. 14
M-02	Liquidation sideBucket masks real LP gains	Logical Error	MEDIUM	RESOLVED	p. 15
M-03	Rounded debts create unclaimable staking rewards	DoS	MEDIUM	RESOLVED	p. 18
M-04	Tokenomics under debits gross winning PnL	Logical Error	MEDIUM	RESOLVED	p. 20
M-05	Insolvent sideBucket losses Inflate paper	Logical Error	MEDIUM	RESOLVED	p. 24
L-01	Queue can create an unbounded dead node prefix	DoS	LOW	RESOLVED	p. 27
L-02	Paper mint basis under counts solvent USD losses	Math	LOW	ACKNOWLEDGED	p. 29
L-03	Debt close poisons same batch USD settlements	Logical Error	LOW	ACKNOWLEDGED	p. 30
L-04	pauseMarket Freezes an Unvalidated BBO Snapshot	Oracle	LOW	ACKNOWLEDGED	p. 33
L-05	No Recovery Path for Failed Core Withdrawals	Unexpected Behavior	LOW	ACKNOWLEDGED	p. 35
L-06	Stale Fee-Address Cache Diverts Dev Fees	Logical Error	LOW	ACKNOWLEDGED	p. 36
L-07	Liquidation Dev Fee Routed to Liquidated User	Logical Error	LOW	ACKNOWLEDGED	p. 38
L-08	Single Signer Can Evict Other via updateSigners	Access Control	LOW	ACKNOWLEDGED	p. 40
L-09	Proxy Admin Functions Lack Transparent Routing	Logical Error	LOW	RESOLVED	p. 42
L-10	bustBuffer Is Not Validated in Instrument Config	Validation	LOW	RESOLVED	p. 44
I-01	Debt queue hides drains LP excess to stakers	Validation	INFORMATIONAL	ACKNOWLEDGED	p. 46
I-02	Activation Fee Hardcoded to One Quote Token	Trust Assumptions	INFORMATIONAL	ACKNOWLEDGED	p. 48
I-03	Timelock Delay Can Be Less Than 7 Days	Trust Assumptions	INFORMATIONAL	ACKNOWLEDGED	p. 49
I-04	Factory Initialization Checks Not Enforced	Warning	INFORMATIONAL	RESOLVED	p. 51
I-05	Unused Modifier and Initializer Disable Helper	Superfluous Code	INFORMATIONAL	ACKNOWLEDGED	p. 53

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
I-06	Unused sweepToExchange function	Superfluous Code	INFORMATIONAL	RESOLVED	p. 54
I-07	Liquidations Pay No Incentive to the Caller	Best Practices	INFORMATIONAL	ACKNOWLEDGED	p. 55
I-08	AlreadySwept Revert Name is Misleading	Error	INFORMATIONAL	RESOLVED	p. 56
I-09	Spot Balance Guard Shorter Than Decoded Width	Validation	INFORMATIONAL	RESOLVED	p. 57
I-10	Unused SafeTokenLib	Superfluous Code	INFORMATIONAL	RESOLVED	p. 59
I-11	Closing a Liquidatable Position Reclaims Buffer	Logical Error	INFORMATIONAL	ACKNOWLEDGED	p. 60
I-12	sideBucket Flush Does Not Fire After settleUsd	Superfluous Code	INFORMATIONAL	RESOLVED	p. 62
I-13	Owner Can Grant Infinite Spend Authority	Trust Assumptions	INFORMATIONAL	ACKNOWLEDGED	p. 63
I-14	Perp To Spot Transfer May Falsely Finalize Sweep	Validation	INFORMATIONAL	RESOLVED	p. 65
I-15	Stale and Inaccurate Code Comments	Documentation	INFORMATIONAL	RESOLVED	p. 67
I-16	Global Trading Pause Only Blocks Position Opens	Documentation	INFORMATIONAL	RESOLVED	p. 69
I-17	Redundant State Write Before Revert in _execute	Superfluous Code	INFORMATIONAL	RESOLVED	p. 71
I-18	CallExecuted Event Fires for Skipped Calls	Events	INFORMATIONAL	RESOLVED	p. 72
I-19	Exchange Change Has No Proxy Migration Path	Trust Assumptions	INFORMATIONAL	ACKNOWLEDGED	p. 74
I-20	Staker Fee Recipient Change Redirects Fees	Documentation	INFORMATIONAL	RESOLVED	p. 76
I-21	Relayer only mode bricks scw exits	Logical Error	INFORMATIONAL	ACKNOWLEDGED	p. 78
I-22	Phantom deposit credits before exchange activated	Logical Error	INFORMATIONAL	ACKNOWLEDGED	p. 80

HIGH **H-01**

RESOLVED

ExchangeSessionFee lets queue drain fees

CATEGORY	Logical Error
LOCATION	queueHarvestAvailable()
REVIEW	Main Review

— DESCRIPTION

In ExchangeSessionFee, the profile adds a new reserved liability sessionFeeAccumulator, and correctly subtracts it in lpBankrollAvailable()

But it does not override queueHarvestAvailable(), so harvest() inherits the base Exchange calculation that ignores sessionFeeAccumulator

That makes unclaimed session-key fees look like queue backing cash

Base \ Exchange.queueHarvestAvailable()\ reserves only

```
``plain text
totalBalances + totalLockedMargin + stakerFeeAccumulator + devFeeAccumulator
``
```

It omits the session-fee reserve. meanwhile, lpBankrollAvailable() correctly adds + sessionFeeAccumulator, proving the session fee profile intended those funds to be reserved.

harvest() then uses \ queueHarvestAvailable()\ and sends that available amount into \ queue.harvest()\, which credits queue users internal balances and increases totalBalances

Assume this state in ExchangeSessionFee

```
```plain text
treasury = 100

sessionFeeAccumulator = 100

sessionKeyFees[feeAddr] = 100

totalBalances = 0

totalLockedMargin = 0

stakerFeeAccumulator = 0

devFeeAccumulator = 0

queue.totalQueued = 100 // attacker is queue head
```

**H-01** ExchangeSessionFee lets queue drain fees

CONTINUED

```
sideBucket = 0
```

```
...
```

Correct behavior: `queueHarvestAvailable()` should return 0, because the only 100 in treasury is reserved for `sessionKeyFees`

Actual behavior: inherited `queueHarvestAvailable()` returns 100, because it ignores `sessionFeeAccumulator`

The attacker or anyone calls `harvest(1)`

The queue pays the attacker 100 into `balances[attacker]`, but `sessionFeeAccumulator` and `sessionKeyFees[feeAddr]` remain unchanged

The same 100 is now double-counted, once as the attacker's withdrawable balance, and once as an unclaimed session fee liability

The attacker can then withdraw the harvested balance via `withdrawToCore`, which debits their internal balance and treasury

**RECOMMENDATION**

Override `queueHarvestAvailable()` in `ExchangeSessionFee`

**RESOLUTION**

Status: Resolved.

**Papertrade Team:** We confirm H-01 and accept High. The session-fee profile now includes `sessionFeeAccumulator` in the reserve calculation used by both LP availability and inherited queue harvest. The base calculation delegates to a virtual reserve function, and `ExchangeSessionFee` supplies the complete liability set. This prevents harvest from consuming cash backing an outstanding session-fee liability. The selected relayer-only profile remains unaffected.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

HIGH **H-02**

RESOLVED

# Enqueued Wins Never Debit trackedLpUsd

CATEGORY	Logical Error
LOCATION	<a href="https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L1364">https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L1364</a> , <a href="https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/PaperTokenomics.sol#L258-L259">https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/PaperTokenomics.sol#L258-L259</a>
REVIEW	Main Review

## DESCRIPTION

When a USD-funded winning close is insolvent, `QueueLib.settleUsd()` can split the winner's PnL into `directProfit` and `enqueuedAmount`. Both values are LP liabilities: `directProfit` is paid immediately, while `enqueuedAmount` is owed through the queue.

However, `_applyBatchCloseAccumulator()` only passes `acc.directProfitCredit` into `PaperTokenomics.onCloseBatch()`, omitting the queued winning amount. Note that this also contradicts the comments that (correctly) [states](<https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L1290-L1292>) the LP debit should be `directProfit + enqueuedAmt`.

As a result, enqueued winning PnL never debits `trackedLpUsd`, and the LP accounting overstates real LP equity by exactly the omitted queued win. This phantom LP surplus later flows through `stakerLpExcessToDistribute()` / `pushStakerFees()` and distributes as real USD to stakers, draining funds that actually back users and LP liabilities.

An attacker can trigger this by creating an insolvent winning close that enqueues profit, then temporarily draining the queue by opening a debt-funded position against their own queued claim. Once later losses settle as solvent, `trackedLpUsd` is credited without ever having been debited for the earlier queued win.

The resulting phantom LP surplus flows into `stakerLpExcessToDistribute()` and is paid out through `pushStakerFees()`.

In the PoC attached, `trackedLpUsd - realLpEquity` equals the omitted enqueued win exactly. The final `pushStakerFees()` call distributes more USD to the staker than the LP's real equity, draining backing from user/LP funds rather than true LP profit.

## RECOMMENDATION

Track queued winning PnL as an LP liability at enqueue time. Accumulate `enqueuedAmount` from insolvent winning closes alongside `directProfitCredit`, and pass `directProfitCredit + enqueuedProfitCredit` to `PaperTokenomics.onCloseBatch()` as LP-paid/owed profit. Also ensure queue harvest does not leave this debit unaccounted.

**PROOF OF CONCEPT** <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/tree/poc/trackedlpusd-enqueued-win-inflation>

---

**H-02** Enqueued Wins Never Debit trackedLpUsd

---

CONTINUED

---

**— RESOLUTION**

Status: Resolved.

**Papertrade Team:** We confirm H-02 and accept High. USD-funded wins now debit tracked LP equity by the complete economic liability: direct profit, queued profit, and applicable winner-fee liabilities. A memory-only accumulator keeps this tokenomics debit separate from user balance credit and from debt-funded reporting. Harvest does not debit the same liability again. This is a prospective accounting correction; production must use a fresh or provably pre-activation deployment unless a separately audited replay reconstructs cursor, PAPER supply, and tail progress together.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

MEDIUM **M-01**

RESOLVED

# Solvent liquidation over-credits trackedLpUsd

CATEGORY	Math
LOCATION	Exchange._liquidatePosition
REVIEW	Main Review

## — DESCRIPTION

On a solvent liquidation the fee is carved into the staker / dev accumulators ( reserved ), so LP's actual net gain is size – fee,

The path reports lpCreditedUsd = size to tokenomics, over-crediting `trackedLpUsd` by the fee on every liquidation

Since stakerLpDistributableExcessUsd() is derived from trackedLpUsd, this inflates the amount stakers can extract via the permissionless pushStakerFees(),

Paying stakers out of LP principal they never earned, the close path computes this correctly

lpCreditedFromLoss = rawLoss – fee, only liquidation omits the " – fee "

## — RECOMMENDATION

Report lpCredited = size – fee in the solvent liquidation branch and 0 when insolvent, as today

## — RESOLUTION

Status: Resolved.

**Papertrade Team:** We confirm M-01 and accept Medium. A solvent USD-funded liquidation now credits tracked LP equity by isolated margin minus the actual liquidation fee. A queue-live liquidation credits the full backed margin because fees are waived and sideBucket delays availability rather than creating another liability. In both branches, the complete forfeited margin remains the independent PAPER mint basis. Liquidation price, fee rates, user loss, and PAPER issuance are unchanged.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

MEDIUM **M-02**

RESOLVED

# Liquidation sideBucket masks real LP gains

CATEGORY	Logical Error
LOCATION	liquidatePosition()
REVIEW	Main Review

## — DESCRIPTION

In liquidate, the insolvent-liquidation branch treats the entire liquidated margin as sideBucket whenever `queue.totalQueued > 0`, even if the queue only needs a tiny amount

The excess is later released to LP cash, but PaperTokenomics is told `lpCreditedUsd = 0`

So dust queue makes large liquidations release untracked LP gains and bypass liquidation fees

In `_liquidatePosition`, for USD funded positions, the code does

```
```plain text
if (queue.totalQueued == 0) {
  // charge liquidation fee
} else {
  sideBucket += size;
}

uint256 lpCredited = queue.totalQueued == 0 ? uint256(size) : 0;
paperTokenomics.onLoss(user, positionId, true, uint256(size), lpCredited);
```
```

So a non empty queue, even a dust queue, forces the whole liquidated margin into sideBucket, skips the entire liquidation fee, and reports zero LP credit to tokenomics

The problem is that `harvest()` later pays only the outstanding queue amount. If the queue clears and sideBucket still has a residual, the residual is simply zeroed, which releases it back into ordinary LP cash

But `PaperTokenomics.onLoss()` only advances `trackedLpUsd` when `lpCreditedUsd > 0`; with `lpCreditedUsd = 0`, the real LP gain is not tracked

**M-02** Liquidation sideBucket masks real LP gains

CONTINUED

Assume

```
``plain text
queue.totalQueued = 1 USDC
sideBucket = 0
liquidated margin = 1,000,000 USDC
``
```

1. A liquidatable USD funded position with size = 1,000,000e18 is liquidated while queue.totalQueued > 0
2. \_liquidatePosition executes the insolvent branch

```
```plain text
sideBucket += 1,000,000e18

no liquidation fee is charged

paperTokenomics.onLoss(, mintBasisUsd = 1,000,000e18, lpCreditedUsd = 0)
```
```

3. Anyone calls harvest()
  4. harvest() pays the 1 USDC queue claim, then sees the queue is cleared and zeroes the remaining sideBucket
  5. The remaining ~999,999 USDC becomes ordinary LP cash, but trackedLpUsd remains unchanged
- Correct behavior would only route the queue shortfall to sideBucket, the rest should be treated as LP credited liquidation gain and reported to tokenomics

The Impact that this lets a tiny live queue distort the accounting of arbitrarily large liquidations

Actual result that will happen  
 LP receives almost all liquidated margin after harvest  
 trackedLpUsd receives 0  
 liquidation fee receives 0  
 PAPER minting uses an artificially low lpStartUsd  
 staker LP excess distribution is suppressed

Because stakerLpDistributableExcessUsd() is derived from trackedLpUsd, any released but untracked LP gain will not become distributable to stakers. Also, since the mint curve uses lpStartUsd = trackedLpUsd, keeping trackedLpUsd artificially low can preserve favorable PAPER minting conditions across repeated liquidations

**M-02** Liquidation sideBucket masks real LP gains

CONTINUED

**— RECOMMENDATION**

Before adding the liquidated margin to sideBucket, compute the actual uncovered queue shortfall, not just whether `queue.totalQueued > 0`

**— RESOLUTION**

Status: Resolved.

**Papertrade Team:** We confirm M-02 and accept Medium. Every backed USD loss now credits tracked LP equity at settlement, net only of actual fee liabilities, regardless of temporary sideBucket routing. A queue-live liquidation credits the full forfeited margin; a solvent liquidation credits margin minus its fee. Harvest, debt consumption, and residual bucket release do not move the cursor again. The documented queue-live fee waiver remains unchanged. Because this is a prospective multi-ledger correction, production must use a fresh or provably pre-activation deployment unless a separately audited replay reconstructs tracked LP cursor, PAPER supply, and tail progress together.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

MEDIUM **M-03**

RESOLVED

# Rounded debts create unclaimable staking rewards

|          |                          |
|----------|--------------------------|
| CATEGORY | DoS                      |
| LOCATION | PaperStaking._claimFor() |
| REVIEW   | Main Review              |

## — DESCRIPTION

PaperStaking can promise users slightly more rewards than it actually holds. even a 1 wei shortage is enough to make an honest user whole reward claim revert

This happens because rewards are divided using integer math, which rounds down

After users stake, unstake, and rewards are distributed, these tiny rounded values can add up incorrectly

Reward per share is rounded down in \_distribute()

User reward debt is also rounded down in \_stakeFor() and \_unstakeFor()

Claiming subtracts the full user amount from rewardReserve in \_claimFor()

creditStakingReward() requires enough staking balance in Exchange.sol

The contract assumes

```
``plain text
total user rewards <= rewardReserve
``
```

But because of rounding, this can become false by 1 wei

Then \_claimFor() does this

```
``plain text
rewardReserve -= amount;
exchange.creditStakingReward(user, amount);
``
```

If amount is even 1 wei greater than rewardReserve, the subtraction reverts. the user receives nothing

Example, an attacker uses tiny stake/reward amounts to create a 1 wei rounding deficit. After they claim

first, the remaining rewardReserve is 1 wei too small for the victim claim, so the victim entire claim reverts.

The attacker only gains dust, but the victim much larger reward claim can be blocked,

**M-03** Rounded debts create unclaimable staking rewards

CONTINUED

An honest user may be unable to claim rewards because the contract is short by only 1 wei, the rewards are stuck until new rewards are added

**— RECOMMENDATION**

make `_claimFor` reserve capped and leave unpaid dust pending  
so the user claims everything available and only the 1 wei shortfall remains pending

**PROOF OF CONCEPT** <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/main/contracts/test/PaperStakingRoundingInsolvencyPoC.t.sol>

**— RESOLUTION**

Status: Resolved.

**Papertrade Team:** We confirm M-03 and accept Medium. `_claimFor` now pays the lesser of the user's calculated reward and `rewardReserve`, retains only the unpaid remainder in `pendingRewards`, updates reward debt normally, and credits/emits/returns only the amount actually paid. The reserve cannot be overdrawn, and a one-wei rounding shortfall cannot block the backed portion of a claim. Any unfunded residual remains pending and can clear when later funding creates sufficient rounding slack. Staking weights, allocation, principal, and queue seniority are unchanged.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

MEDIUM **M-04**

RESOLVED

# Tokenomics under debits gross winning PnL

CATEGORY Logical Error  
 LOCATION `applyBatchCloseAccumulator()`  
 REVIEW Main Review

## DESCRIPTION

Solvent winning closes under debit `trackedLpUsd` by the win fee, enabling fake LP excess distribution  
`QueueLib.settleUsd()` handles a solvent win as

```
``plain text
rawProfit = adjustedPnl
fee = rawProfit * winFeeRate
directProfit = rawProfit - fee
stakerFee + devFee = fee
``
```

The code says the solvent win path pays `rawProfit - fee` to the user and accrues the fee to staker / dev accumulators

But the actual LP bankroll delta is `-rawProfit`, not `-(rawProfit - fee)`

The Exchange code own LP budget derivation confirms this, for a solvent win, the LP free delta is documented as `-rawProfit`, because both the user net profit and the fee reserve are liabilities against LP free cash

However, `_applyBatchCloseAccumulator()` sends only `acc.directProfitCredit` to `onCloseBatch()` as `lpPaidProfitUsd`,

`directProfitCredit` is post fee user profit, while `feesFromLoss` only corrects the loss branch; there is no equivalent `feesFromWin` or gross-win debit

`onCloseBatch()` then debits `trackedLpUsd` only by `lpPaidProfitUsd`, and `stakerLpDistributableExcessUsd()` uses `trackedLpUsd - stakerLpRewardCapUsd` as the amount available for LP excess distribution

Assume

```
```plain text
size = 1,000
adjustedPnl / rawProfit = 100
winFeeRate = 2%
fee = 2
directProfit = 98
```

M-04 Tokenomics under debits gross winning PnL

CONTINUED

```
queue.totalQueued = 0
```

```
...
```

Actual Exchange accounting on close

```
``plain text
```

```
locked margin released: +1,000 to LP-free calculation
```

```
user balance credited: -1,098 (margin + directProfit)
```

```
fee accumulators credited: -2
```

```
-----
```

```
actual LP-free delta: -100
```

```
``
```

But tokenomics receives

```
``plain text
```

```
lpPaidProfitUsd = directProfitCredit = 98
```

```
trackedLpUsd delta = -98
```

```
``
```

So after this single close

```
``plain text
```

```
trackedLpUsd is overstated by 2 = the win fee
```

```
``
```

With default fees, the drift is 2% of every solvent winning PnL. If governance sets winFeeRate to the allowed maximum, the drift can be up to 50% of every solvent winning PnL

pushStakerFees() first claims real staker fees, then asks PaperTokenomics for LP excess, calls distributeLpToStakers(lpExcess), and pays or queues that amount to the staking recipient through _enqueueOrPay()

Because trackedLpUsd is too high by cumulative solvent win fees, stakerLpDistributableExcessUsd() can report surplus that does not exist as LP profit. _enqueueOrPay() then turns that fake surplus into a real staker balance if LP cash is available, or a queue claim if it is not

This is effectively a double allocation of winner side fees

1. The win fee is already reserved in stakerFeeAccumulator / devFeeAccumulator
2. The same fee amount is not debited from trackedLpUsd
3. Later, inflated trackedLpUsd can cause the protocol to distribute that amount again as LP excess

M-04 Tokenomics under debits gross winning PnL

CONTINUED

RECOMMENDATION

we need to not reuse directProfitCredit as LP PnL, keep it only for user balance credit and add a separate accumulator for the tokenomics LP debit, for example

```
``plain text
uint256 lpPaidProfitForTokenomics;
``
```

Then in the USD funded win branch

```
```solidity
if (isWin) {
// Preserves the current design of excluding queued wins if desired.
// For solvent wins, sfU + dfU is the win fee, so this debits rawProfit
acc.lpPaidProfitForTokenomics += directProfit + sfU + dfU;

// Existing event / user facing accounting can remain unchanged
acc.totalProfitAfterFee += directProfit + enqueuedAmt;
}
```
```

And In _applyBatchCloseAccumulator()

```
``javascript
paperTokenomics.onCloseBatch(
user,
acc.lpPaidProfitForTokenomics,
mintBasis,
lpCreditedFromLoss
);
``
```

RESOLUTION

Status: Resolved.

Papertrade Team: We confirm M-04 and accept Medium. USD-funded winners now debit tracked LP equity by direct profit, queued profit, and all actual winner-fee liabilities. User balance credit remains separate, and debt-funded closes retain their distinct incremental-profit semantics. Harvest does not debit the same liability again. The shared H-02/M-04 correction adds no storage and changes no user payout. Production must use a fresh or provably unaffected/pre-activation deployment unless a separately audited replay reconstructs tracked LP cursor, PAPER supply, and tail progress together.

M-04 Tokenomics under debits gross winning PnL

CONTINUED

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

MEDIUM **M-05**

RESOLVED

Insolvent sideBucket losses Inflate paper

CATEGORY Logical Error

LOCATION settleUsd()

REVIEW Main Review

— DESCRIPTION

Related with M-02, L-10

The intended logic in the system, If user loses real money, the system gives them paper as compensation / reward

This vulnerability could allow an external attacker to mint a huge amount of paper while losing almost no real USDC

The vulnerable line is the insolvent loss branch in QueueLib.settleUsd()

```
``plain text
uint256 deduction = remainingLoss < size256 ? remainingLoss : size256;
sideBucketAdd = deduction;
marginOut = size256 - deduction;
``
```

When queue.totalQueued > 0, a USD funded losing close does not credit the loss to LP and does not burn the user queued claim

It parks the deducted margin in sideBucketAdd, returns the remaining margin, and later _applyBatchCloseAccumulator() mints paper on the full realized loss while setting lpCreditedFromLoss to zero for the sideBucketed slice

That is already dangerous, but the critical path is that the parked loss can be atomically converted back into attacker available LP budget by consuming the tiny queue that triggered the insolvent branch, exploit

1. Attacker has or creates a small queued claim Q, for example \$1, so queue.totalQueued > 0
2. Attacker closes a large USD funded losing position with loss L, because the queue is non empty, the snippet executes

```
``plain text
sideBucketAdd = L;
marginOut = size - L;
``
```

The attacker loss is treated as an insolvent loss and mints paper, but it does not advance

M-05 Insolvent sideBucket losses Inflate paper

CONTINUED

LP credit accounting

1. Attacker opens a debt funded position of size Q

_openPosition(useDebt=true) consumes the attacker queue claim. If this drains queue.totalQueued to zero, exchange executes

```
``plain text
if (queue.totalQueued == 0 && sideBucket > 0) sideBucket = 0;
``
```

The large sideBucket amount L is now released into ordinary LP bankroll

1. Attacker closes the debt funded position and a winning USD funded position in the same close batch, ordering the debt funded close first

The debt close re enqueues only the tiny claim Q, while the batch accumulator still has the released LP budget. The later USD-funded win sees queue.totalQueued > 0, so the win fee branch is bypassed, but it can still receive direct LP payment from the stale acc.lpBudgetRemaining

1. Net result is that the attacker pays only about the tiny catalyst queue amount Q, preserves / recreates the queue claim, and receives paper minted on the large loss L

Assume

```
``plain text
Q = 1 USDC queued claim
L = 1,000,000 USDC losing close
W = 999,999 USDC winning close
`
```

```
``plain text
```

close loser:

sideBucketAdd = 1,000,000

PAPER minted on 1,000,000 loss

debt open with Q:

queue.totalQueued becomes 0

sideBucket is zeroed/released into LP bankroll

batch close:

debt close re-enqueues Q

winning close direct-pays ~999,999 with no win fee

```
``
```

The attacker USDC cost is approximately Q, while paper is minted as if they suffered a real L loss. This is effectively near free paper inflation, and because paper participates in staking / reward distribution, it can dilute honest stakers and route future usdc rewards to the attacker

M-05 Insolvent sideBucket losses Inflate paper

CONTINUED

In other words, this vulnerability lets the attacker do

1. Create a tiny unpaid queue claim, like \\$.1.
2. Close a huge losing trade, like \\$.1,000,000, so the system mints PAPER as if they truly lost \\$.1,000,000
3. Use the tiny queue claim to unlock the sideBucket accounting
4. Close another winning trade and get almost the same \\$.1,000,000 back from the released LP budget.

So the attacker net USDC cost can be close to the tiny queue amount, while they receive PAPER as if they took a huge real loss

A note to include is that for the winning leg we do `adjustedPnl = applyMarketImpact()` In QueueLib, which slightly reduces the profit since the rollbit curve takes a haircut, but the attack is still profitable

RECOMMENDATION

prevent sideBucketed losses from being released into same transaction or same episode LP budget

```
`plain text`  
// we need to not blindly zero sideBucket when debt consumption drains the queue  
if (queue.totalQueued == 0 && sideBucket > 0) {  
  // Either keep it reserved until harvest/accounting settles,  
  // or explicitly account it as LP credited before release.  
}  
`
```

RESOLUTION

Status: Resolved.

Papertrade Team: We partially confirm M-05 and request cross-reference to M-02, L-10, and L-07 rather than a second independent accounting issue. The remediation ensures every backed USD loss credits tracked LP equity at settlement, net only of actual fee liabilities, regardless of temporary sideBucket routing; bucket clearance does not credit it again. The composite regression reproduces the reported sequence and proves immediate backing plus terminal queue, bucket, margin, and tracked-LP reconciliation. Gross PAPER issuance on a losing isolated leg while another leg wins remains the independent-settlement product policy acknowledged under L-07, and same-batch backed self-netting remains the acknowledged L-10 policy. The accounting defect is resolved for fresh or provably pre-activation deployment.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

LOW **L-01**

RESOLVED

Queue can create an unbounded dead node prefix

| | |
|----------|--------------|
| CATEGORY | DoS |
| LOCATION | QueueLib.sol |
| REVIEW | Main Review |

— DESCRIPTION

In the Martingale queue and debt-funded trading flow

The queue uses monotonically increasing node IDs, enqueue() only initializes head when head == 0, and debt consumption lazy-deletes consumed nodes by setting amount = 0 without advancing the global head. A user with a queued claim can repeatedly

1. Open a debt funded position using the queued claim as margin, _openPosition(useDebt=true) consumes from queue.userQueueTail[user\], and if the queue drains it only clears sideBucket; it does not reset or advance queue.head
2. Close the position at roughly zero PnL. settleDebt() treats zero PnL as surviving margin and enqueues the same amount again as a fresh tail claim.
3. Repeat. Each cycle preserves the attacker queued value but leaves another dead global node before the live tail node

After many cycles, the global queue shape becomes

```
``plain text
head -> dead -> dead -> dead -> ... -> dead -> live attacker claim
``
```

The DoS trigger is harvest(maxEntries), it scans from self.head, and when it sees a dead node (amount == 0) it increments both current and processed

Dead-node skips therefore consume the caller's maxEntries budget, even though no user is paid

Because lpBankrollAvailable() returns 0 whenever queue.totalQueued > 0, a single live claim behind enough dead nodes can keep the whole system in queue live / LP unavailable mode while harvest calls waste gas skipping attacker created dead nodes

Queue payouts can be delayed indefinitely unless keepers overpay for enough skip work

Direct LP payouts remain disabled while queue.totalQueued > 0

Staker / dev / LP excess flows that depend on _enqueueOrPay or LP availability are forced into the degraded queue path

L-01 Queue can create an unbounded dead node prefix

CONTINUED

— RECOMMENDATION

Replace numeric tombstone scanning with a global live node FIFO (next / prev) and unlink any node when debt consume / harvest zeros it, resetting head / tail on empty

— RESOLUTION

Status: Resolved.

Papertrade Team: We partially confirm L-08 at Low severity. A fixed tombstone prefix was finite because every bounded harvest persisted its advanced cursor, and sustained delay required continued paid churn. The remediation nevertheless prevents dead prefixes from carrying into a new economic queue episode: whenever debt consumption or harvest reduces totalQueued to zero, the global head advances directly to nextNodeid in $O(1)$. A later enqueue is immediately harvestable. Tombstones created while another claim keeps the episode live remain bounded work because each skipped ID advances the persistent cursor exactly once.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

LOW L-02

ACKNOWLEDGED

Paper mint basis under counts solvent USD losses

| | |
|----------|--------------------------------------|
| CATEGORY | Math |
| LOCATION | Exchange._applyBatchCloseAccumulator |
| REVIEW | Main Review |

— DESCRIPTION

In a solvent USD loss the fee is LP-side settleUsd carves it from LP's gain, the user loses the full margin haircut

The user realised loss is rawLoss, but the mint basis is computed as $\text{totalLossUsd} - \text{feesFromLoss} = \text{rawLoss} - \text{fee}$

Per IPaperTokenomics, mintBasisUsd should equal the user realised loss, so losing traders are minted slightly less PAPER than the documented basis

That's the same fee-handling inconsistency issue exist in M-01, but in different place

— RECOMMENDATION

Mint on the user's full realised loss rawLoss, keeping lpCreditedFromLoss as the fee excluded LP figure

— RESOLUTION

Status: Acknowledged.

Papertrade Team: We partially confirm L-09 at Low severity. On a normal queue-empty USD loss, the user's balance falls by gross capped loss L; fee allocation F is carved from LP's gain and reserved separately; PAPER eligibility is $L - F$. At the default 2% rate and within the flat emission region, that is a 98% mint basis subject to rounding. It is not an additional fee charged to the losing user. Queue-live losses, debt-claim destruction, and liquidation use separately documented bases. The settlement formula is retained and the interface now describes branch-specific PAPER-eligible loss rather than universally gross realized loss.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

LOW **L-03**

ACKNOWLEDGED

Debt close poisons same batch USD settlements

| | |
|----------|--|
| CATEGORY | Logical Error |
| LOCATION | Exchange._settleFundingToAccumulator() |
| REVIEW | Main Review |

— DESCRIPTION

batch-created queue state lets a trader bypass win fees while still receiving direct LP payment

The path is the close accumulator flow around _applyBatchCloseAccumulator, but the root cause starts earlier in _settleFundingToAccumulator

a debt-funded close can create a new queue claim, yet the batch accumulator only subtracts that claim from acc.lpBudgetRemaining instead of zeroing LP availability once `queue.totalQueued > 0`,

The same batch can then close a normal USD-funded winning position. Because `settleUsd` sees `totalQueued != 0`, it takes the insolvent branch, which charges no win fee, but it still direct-pays `min(lpBudget, rawProfit)` from the stale positive `acc.lpBudgetRemaining`

The final `_applyBatchCloseAccumulator` then credits that full direct profit to the trader's balance

The issue is that a tiny debt-funded close can be used as a queue toggle to make later USD funded profitable closes in the same batch settle as insolvent / no fee while still being paid directly

A trader can repeatedly avoid `winFeeRate` on otherwise solvent winning closes. With the default 2% win fee, every profitable close can be overpaid by 2% of adjusted profit

The fee should be split into `stakerFee` and `devFee` on solvent wins, but the transient queue makes `settleUsd` skip both while still assigning `directProfit = rawProfit`

Assume

```
```javascript
```

```
queue.totalQueued = 0
```

```
lpBankrollAvailable = 1,000
```

**L-03** Debt close poisons same batch USD settlements

CONTINUED

```
winFeeRate = 2%
```

```
...
```

Attacker has

A) one tiny debt-funded position, size = 1, adjustedPnl = 0

B) one USD-funded winning position, raw/adjusted profit = 100

And call close(debtFundedPositionId, usdFundedWinningPositionId)

Execution

\_close snapshots acc.lpBudgetRemaining = lpBankrollAvailable() once at batch start

The debt-funded flat close calls queue.settleDebt(), enqueues 1, and only reduces

acc.lpBudgetRemaining from 1,000 to 999. It does not set the accumulator budget to zero even though queue.totalQueued is now nonzero

The USD-funded winning close now enters settleUsd with queue.totalQueued > 0, so solvent = false. In the insolvent branch, no fee is charged, but directProfit = min(999, 100) = 100

\_applyBatchCloseAccumulator commits directProfitCredit = 100 to the attacker's balance. The correct solvent result would have been 98 to the user and 2 to fee accumulators

The attacker can recycle the small debt funded marker by reopening it from the newly queued claim after the batch, making the fee bypass repeatable

Root cause that the accumulator tries to preserve sequential close semantics, but it treats debt-funded enqueue growth as merely consuming part of the batch LP budget

```
``plain text
if (enqueuedAmount >= acc.lpBudgetRemaining) {
 acc.lpBudgetRemaining = 0;
} else {
 acc.lpBudgetRemaining -= enqueuedAmount;
}
``
```

That is inconsistent with the contract's own global solvency rule, lpBankrollAvailable() returns zero whenever queue.totalQueued > 0

**— RECOMMENDATION**

When any close path creates a queue entry inside a batch, force the remaining batch LP budget to zero, or make settleUsd ignore lpBudget whenever self.totalQueued > 0

---

L-03 Debt close poisons same batch USD settlements

---

CONTINUED

---

— RESOLUTION

Status: Acknowledged.

**Papertrade Team:** We confirm the transition and acknowledge L-10 as Low-severity design behavior. A batch begins with positive LP availability only when the global queue is empty. If a debt-funded close recreates the caller's claim, that full claim is deducted from the running LP budget. A later USD-funded win by the same caller may use only the remaining backed budget while receiving the documented queue-live fee waiver; any excess is queued. It cannot jump another live creditor, spend the recreated claim's backing, or create unbacked balance. We accept this order-dependent atomic self-netting and the associated fee under-collection because it produces no persistent accounting drift or drain.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

LOW **L-04**

ACKNOWLEDGED

# pauseMarket Freezes an Unvalidated BBO Snapshot

CATEGORY	Oracle
LOCATION	<a href="https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L538">https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L538</a>
REVIEW	Main Review

## — DESCRIPTION

When a market is paused, `pauseMarket` reads `oracle.getBBO(assetIndex)` and stores the bid/ask as the frozen price used to settle every close and liquidation for the duration of the pause. The problem is that it reads through `getBBO`, which does no validation. Unlike `_getBboMidPrice`, the `getBBO` path does not check that bid and ask are non-zero or that the book isn't crossed (`ask >= bid`), so whatever the book looks like at the exact pause block gets stored in as the settlement price.

Since a pause may be triggered when a market looks wrong, which is also when the book is most likely to be distorted. If an attacker has manipulated the order book at that moment, or front-runs the pause transaction in the same block, the distorted price is captured and used for all settlement points until the market is unpaused.

## — RECOMMENDATION

Validate the snapshot using the same checks `_getBboMidPrice` already applies, rejecting zero or crossed books. Alternatively, consider freezing the mark or oracle price instead of the raw BBO, since it is far harder to distort at a single instant. Optionally compare the captured BBO against the mark price and refuse to freeze if they diverge beyond a threshold (with an extra force freeze function to bypass this in extreme cases).

## — RESOLUTION

Status: Acknowledged.

**Papertrade Team:** We partially confirm L-11 at Low severity. The pause path's raw BBO snapshot is structurally less defensive than normal execution, but a well-formed HyperCore BBO is intentionally authoritative and is not compared with another mark or oracle. Market retirement is now irreversible: once paused, the old instrument ID never returns to live pricing and all remaining positions settle against its captured midpoint. Relisting requires a fresh sequential ID after both OI sides on the retired ID reach zero. This eliminates reversible frozen/live cycling but does not validate the snapshot. If the proposed snapshot

---

**L-04** pauseMarket Freezes an Unvalidated BBO Snapshot**CONTINUED**

---

is suspect, operators first use the owner-restorable global opens gate while positions continue to settle at live price. Calling pauseMarket is therefore explicit acceptance of a final settlement snapshot.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

LOW **L-05**

ACKNOWLEDGED

# No Recovery Path for Failed Core Withdrawals

CATEGORY Unexpected Behavior

LOCATION <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L740-L772>

REVIEW Main Review

## DESCRIPTION

The deposit side has an owner recovery path for funds that reach Core but don't get credited:

`clearPendingSweep` plus `reconcileSweepCredit` let the owner clear a stuck sweep and manually credit the user. The withdrawal side has no equivalent. `_withdrawToCore` debits the user's internal balance and executes the `spotSend` with no confirmation step, so if the Core transfer fails on landing the user is left debited with nothing received and no on-chain way to re-credit them.

## RECOMMENDATION

Add an owner reconcile path for failed withdrawals similar to `reconcileSweepCredit`, so a user debited without delivery can be re-credited.

## RESOLUTION

Status: Acknowledged.

**Papertrade Team:** We partially confirm L-01 and accept Low. The withdrawal path debits internal accounting before the CoreWriter action settles and does not store a withdrawal-specific completion record. `reconcileSweepCredit` provides an owner-only re-credit path when authoritative evidence proves that delivery failed and Exchange retained the corresponding Core funds. We retain this exceptional manual path rather than add withdrawal-specific storage and gas to every withdrawal. The runbook requires the pinned CoreWithdrawal event, authoritative non-delivery and retained-funds evidence after the defined finality window, exact reconstruction of the full debit including activation headroom, a timelocked governance operation hash, pre/post ledger checks, and an external exactly-once incident record. No automatic or permissionless recovery is introduced.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

LOW **L-06**

ACKNOWLEDGED

## Stale Fee-Address Cache Diverts Dev Fees

CATEGORY	Logical Error
LOCATION	<a href="https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/ExchangeSessionFee.sol#L86-L91">https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/ExchangeSessionFee.sol#L86-L91</a> ,
REVIEW	Main Review

### — DESCRIPTION

On `ExchangeSessionFee`, `_creditDevFees` routes the dev-fee portion of a trade to `userFeeAddress[user]` whenever one is set, falling back to the protocol's `devFeeAccumulator` only when it is zero. That cache is populated by `_cacheSessionFeeAddress`, which is called only on the signature paths (openPositionBySig, closeBySig). The direct paths (close, openPosition) are inherited from the base Exchange and never refresh the cache.

The profile's stated purpose is to route the dev fee to the active session key's fee address (i.e. for session-key-facilitated trades). But because the cache remains and the direct paths never update it, a single `*BySig` trade that caches `userFeeAddress[user] = self` causes every subsequent direct close to divert the dev fee to that cached address, even though those direct calls involve no session key at all. So the diversion leaks outside the session-key context it was designed for: a user registers a session key with their own fee address, makes one signed trade to seed the cache, and from then on the dev-fee portion of all their ordinary direct trading is routed to themselves and claimed via `claimSessionKeyFees`.

The PoC (DevFeeRoutingPoC.t.sol) confirms a direct (non-signature) close after a single prior `*BySig` trade diverts the dev fee to the user's own address (0.0444 to self, 0 to the protocol) and claims it out. The net effect is a permanent diversion of the protocol's entire dev-fee revenue stream on this profile for any user, triggered by a one-time registration, and applying to trade paths the session-fee routing was never intended to cover.

### — RECOMMENDATION

Refresh the fee-address routing on the path that actually executes the trade rather than relying on the cache: either key dev-fee routing off the per-transaction signer (so a direct trade with no session key falls back to `devFeeAccumulator`), or clear/recompute `userFeeAddress[user]` on the direct paths.

**PROOF OF CONCEPT** <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/tree/poc/dev-fee-routing>

---

**L-06** Stale Fee-Address Cache Diverts Dev Fees

---

CONTINUED

**— RESOLUTION**

Status: Acknowledged.

**Papertrade Team:** We acknowledge L-02 as an optional-profile policy. A caller cannot attach a recipient to another user arbitrarily: the user's wallet authorizes registration, chooses feeAddress, and the session key must execute a valid signed action before Exchange caches it. The most recently used signed-session recipient persists for later closes and liquidations until another valid signed action replaces or clears it. Self-referral is an allowed rebate choice in this profile and changes only the existing developer-fee destination; it creates no value and cannot access another user's balance. The selected relayer-only deployment does not use ExchangeSessionFee.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

LOW **L-07**

ACKNOWLEDGED

# Liquidation Dev Fee Routed to Liquidated User

CATEGORY	Logical Error
LOCATION	<a href="https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/ExchangeSessionFee.sol#L75-L84">https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/ExchangeSessionFee.sol#L75-L84</a> , <a href="https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L1060">https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L1060</a>
REVIEW	Main Review

## DESCRIPTION

On `ExchangeSessionFee`, the dev portion of the liquidation penalty is routed to the liquidated user's own fee address. `_liquidatePosition` charges the liquidation fee and credits the dev half via `_creditDevFees(pos.user, devPortion)`, and the overridden `_creditDevFees` sends that to `userFeeAddress[pos.user]` (the position owner's configured fee address) rather than to the protocol's `devFeeAccumulator`:

```
``plain text
sessionKeyFees[feeAddress] += devPortion;
sessionFeeAccumulator += devPortion;
`
```

So when a user is liquidated, half of their own liquidation penalty is credited back to a fee address they control, claimable via `claimSessionKeyFees`. Because liquidate is permissionless, a user can also self-liquidate to deliberately recapture it. The PoC (`DevFeeRoutingPoC.t.sol`) confirms a liquidated user recaptures 0.1 of their own penalty while the protocol receives 0.

Routing a penalty's revenue back to the penalized party undermines the penalty. The liquidation fee is meant to be protocol/staker revenue that offsets liquidation handling and discourages reaching liquidation; refunding half of it to the liquidated user both removes that protocol revenue and weakens the economic deterrent. The fee address is user-chosen at session-key registration with no validation, so any user can configure this recapture in advance.

## RECOMMENDATION

Consider keeping the liquidation dev fee in `devFeeAccumulator` (or route it to a keeper/liquidator identity), so the penalty's revenue cannot be recaptured by the party being penalized.

**PROOF OF CONCEPT** <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/tree/poc/dev-fee-routing>

---

**L-07** Liquidation Dev Fee Routed to Liquidated User

---

CONTINUED

---

**— RESOLUTION**

Status: Acknowledged.

**Papertrade Team:** We acknowledge L-03 and request consolidation with L-02. ExchangeSessionFee intentionally applies the wallet-authorized persistent recipient to the existing developer share on both closes and solvent liquidations. The user may control that address, so the share can act as a self-referral rebate. It does not cancel liquidation, return forfeited margin, change the staker share, create a liability, or affect another user. Queue-live liquidations remain fee-free under the separate insolvency policy. The selected relayer-only deployment does not use this profile.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

LOW **L-08**

ACKNOWLEDGED

# Single Signer Can Evict Other via updateSigners

CATEGORY Access Control

LOCATION <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/governance/OneOfTwoMultisig.sol#L87>

REVIEW Main Review

## DESCRIPTION

`OneOfTwoMultisig` serves as the protocol's governance: it owns `GovernanceTimelock`, which in turn owns every Ownable contract and the ProxyAdmin. The multisig is also the direct pauseGuardian on the Exchange. The timelock is single-owner with no separate proposer, executor, or canceller roles, so whoever controls the multisig controls the entire system.

`updateSigners` is gated by `onlySelf (msg.sender == address(this))`, which appears to require consensus:

```
``plain text
function updateSigners(address newSignerOne, address newSignerTwo) external {
 if (msg.sender != address(this)) revert NotSigner();
 _setSigners(newSignerOne, newSignerTwo);
}
```

But a single signer defeats this gate. `requiredConfirmations()` is hardcoded to 1, and `submitTransaction(..., executeImmediately=true)` runs `_confirm` and `_execute` in one call. So one signer can submit a self-call that rotates both signers in a single transaction:

```
`plain text
submitTransaction(
 address(this),
 0,
 abi.encodeCall(OneOfTwoMultisig.updateSigners, (attacker1, attacker2)),
 true
);
```

This allows one signer to permanently evict the co-signer and seize exclusive control of the governance.

The eviction also defeats the timelock's delay protection. `cancel`` on the timelock is onlyOwner (the multisig), so the only party that can cancel a malicious scheduled operation is the multisig itself. Once a signer has evicted the co-signer, that co-signer can no

**L-08** Single Signer Can Evict Other via updateSigners

CONTINUED

longer cancel anything, and the timelock has no independent canceller or guardian role to fall back on.

**— RECOMMENDATION**

Require a multi-signer authorization for `updateSigners` specifically, either a true 2-of-2 confirmation for that operation, or route signer changes through the timelock so the change is subject to the delay and visible before it takes effect. Consider adding an independent timelock canceller or guardian role, separate from the multisig executor, so one signer who takes over the multisig is not also the only on-chain party able to cancel pending malicious operations.

**— RESOLUTION**

Status: Acknowledged.

**Papertrade Team:** We acknowledge L-04 and accept Low as a governance trust assumption. Either signer is independently authorized to execute multisig self-calls and schedule owner actions, so one signer can replace both signers and exclude the other party's monitoring/recovery role. Deployments requiring two-key consent must use a true threshold wallet. Separately, emergency authority is now pause-only: a guardian may engage the global opens gate or permanently retire a market ID, but only owner/timelock may restore global opens and no caller can revive a retired ID. Remaining old-ID positions settle against the frozen snapshot. Relisting requires both long and short OI on the retired ID to be zero, then a fresh sequential instrument ID. The release gate enforces this zero-OI/fresh-ID sequence. This removes recursive frozen/live settlement cycling without changing the acknowledged 1-of-2 signer model.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

LOW **L-09**

RESOLVED

# Proxy Admin Functions Lack Transparent Routing

CATEGORY	Logical Error
LOCATION	<a href="https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/proxy/AdminUpgradeableProxy.sol#L51">https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/proxy/AdminUpgradeableProxy.sol#L51</a>
REVIEW	Main Review

## — DESCRIPTION

`AdminUpgradeableProxy` follows the transparent-proxy pattern, it defines admin-only functions on the proxy itself ( `admin()`, `implementation()`, `changeAdmin`, `upgradeToAndCall` ) — but omits the caller-based fallback routing that makes a transparent proxy safe. The fallback delegates to the implementation unconditionally:

```
``plain text
fallback() external payable {
 _delegate(_implementation());
}
```

In a standard transparent proxy, the fallback routes based on the caller: the admin's calls reach the proxy's own admin functions, and all non-admin calls are delegated to the implementation regardless of selector. Here, the admin functions live on the proxy's dispatcher unconditionally, so they are matched for every caller before the fallback is reached. This creates a selector-clash surface. If the implementation defines a function whose 4-byte selector collides with `admin()`, `implementation()`, `changeAdmin(address)`, or `upgradeToAndCall(address,bytes)`, the proxy's own function shadows it, and that implementation function becomes unreachable through the proxy. Because those functions are `onlyAdmin`, a non-admin caller attempting to reach the shadowed implementation function reverts with `NotAdmin` rather than reaching the implementation at all. The collision surface is limited to these four selectors, so it is unlikely, but if it occurs it silently renders an implementation function unreachable.

## — RECOMMENDATION

Adopt transparent-proxy routing: gate the admin functions in the fallback on `msg.sender == admin` so that non-admin callers always delegate to the implementation (including for selectors that match the admin function names), and only the admin reaches the proxy's own admin functions. Alternatively, if the direct-function approach is kept, document the four reserved selectors so implementations avoid colliding with them.

---

**L-09** Proxy Admin Functions Lack Transparent Routing

---

CONTINUED

---

**— RESOLUTION**

Status: Resolved.

**Papertrade Team:** We acknowledge L-05 as a Low future-upgrade compatibility risk. The direct-function proxy design reserves selectors for `admin()`, `implementation()`, `changeAdmin(address)`, and `upgradeToAndCall(address,bytes)`. The remediation derives those selectors from the proxy ABI and checks the complete inherited ABI of every manifest-listed proxy implementation. All reviewed implementations are collision-free. The check runs in both upgrade and freeze gates, and implementation overrides must be listed and bytecode-matched to reviewed artifacts before deployment.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

LOW **L-10**

RESOLVED

# bustBuffer Is Not Validated in Instrument Config

CATEGORY Validation

LOCATION <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L1790>

REVIEW Main Review

## DESCRIPTION

`_validateInstrumentConfig` bounds-checks every numeric field of an instrument's configuration except `bustBuffer`. It validates `maxMultiplier`, `baseRate` (required `< 1e18`), `rateMultiplier`, `referenceNotional`, and `maxUserPosition`, but `bustBuffer` is accepted with no bound. The field is set once at `addInstrument` and has no setter, so whatever value is supplied is permanent for the life of the instrument.

`bustBuffer` feeds the liquidation-trigger math in `_getBustPriceCached`. A value of `1e18` or greater produces two distinct severe outcomes.

For longs, the bust trigger collapses onto (or above) the entry price, so a brand-new long with no adverse price move is immediately liquidatable, so the position can be liquidated at entry for a full margin wipe the instant it opens.

For shorts, the bust price computation `closePrice - (closePrice * bustBuffer) / 1e18` underflows and reverts when `bustBuffer >= 1e18`, so every liquidation of a short on that instrument reverts. Positions that should be liquidated cannot be, leaving the LP to carry uncollateralizable bad debt.

Since the owner configures instruments, this is an admin-configuration footgun, but it is a missing-validation gap in a place where every sibling field is validated.

## RECOMMENDATION

Add a `bustBuffer < 1e18` check to `_validateInstrumentConfig`, mirroring the existing `baseRate` validation, so the long trigger cannot collapse onto entry and the short computation cannot underflow.

Since the field has no setter, validating it at `addInstrument` is the only point of enforcement; consider also adding a setter (validated identically) so a misconfigured instrument can be corrected without redeployment.

## RESOLUTION

Status: Resolved.

**L-10** bustBuffer Is Not Validated in Instrument Config

CONTINUED

**Papertrade Team:** We confirm L-06 as Low. Instrument creation now rejects bustBuffer  $\geq 1e18$ . At exactly  $1e18$ , the long trigger can be at or above entry and the short trigger reaches zero; larger values can underflow the short calculation. Boundary tests accept  $1e18 - 1$  and reject  $1e18$  and larger values with `InvalidInstrumentConfig`. No setter is added because changing liquidation thresholds after positions exist would create a separate governance and user-expectation risk. The check adds no storage and no open/close/liquidation runtime gas.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-01

ACKNOWLEDGED

# Debt queue hides drains LP excess to stakers

CATEGORY	Validation
LOCATION	openPosition()
REVIEW	Main Review

## — DESCRIPTION

Debt funded positions can temporarily hide queue liabilities from `lpBankrollAvailable()`, allowing LP excess to be paid to stakers while the same queue claim can later be re created

`lpBankrollAvailable()` only blocks LP payouts when `queue.totalQueued > 0`, Once the queue is zero, it reserves only balances, locked margin, staker fees, and dev fees

The code comment says debt-funded open positions are not reserved, because consuming queue debt removes the live senior queue claim and converts it into contingent position collateral

The problem is that a user can open a position with `useDebt = true`, consuming their queued claim with `queue.consumeDebtFromBack()`, If that drains the queue, `queue.totalQueued` becomes zero,

And the code zeroes `sideBucket` when the queue is drained. no balances or `totalLockedMargin` reserve is created for this debt funded position

At that point, `pushStakerFees()` can distribute LP excess. It asks `tokenomics` for `stakerLpExcessToDistribute()`, calls `paperTokenomics.distributeLpToStakers(lpExcess)`,

Then pays the staking contract through `_enqueueOrPay()`,  
`_enqueueOrPay()` direct credits the recipient when `lpBankrollAvailable()` says enough bankroll exists, otherwise it only enqueues the unpaid remainder

Since the queue has been temporarily hidden, this direct-credit path can fire using funds that should still be backing the contingent queue liability

Then `PaperStaking.distribute()` treats the newly credited Exchange balance as newly received rewards and allocates it pro rata to current stakers

A staker can later claim, and `creditStakingReward()` moves internal balance from the staking contract to the claiming user

Finally, the attacker can close the debt-funded position. `settleDebt()` re enqueues the surviving margin, and winning closes enqueue margin plus profit.

**I-01** Debt queue hides drains LP excess to stakers

CONTINUED

So the attacker can restore the queue claim after the LP/staker payout has already been credited

**— RECOMMENDATION**

Track debt-funded open liabilities as a reserved amount, increment it on useDebt open, decrement it on debt funded close or liquidation, and include it in lpBankrollAvailable()

**— RESOLUTION**

Status: Acknowledged.

**Papertrade Team:** We confirm the described transition and request Informational treatment. Under the submitted realized-state model, `useDebt` consumes the live FIFO claim and converts it into contingent position collateral. The contract does not track live user PnL as a present liability. A normal close creates a fresh tail claim, while liquidation destroys the contingent claim. Cash may therefore be distributable while the position remains open, and a later close may recreate a claim after an LP-excess distribution. This is the disclosed seniority policy rather than an omitted ledger entry, and terminal lifecycle reconciliation is preserved.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-02

ACKNOWLEDGED

# Activation Fee Hardcoded to One Quote Token

CATEGORY	Trust Assumptions
LOCATION	<a href="https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L752">https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L752</a>
REVIEW	Main Review

## DESCRIPTION

In `_withdrawToCore`, when the destination account isn't yet activated on HyperCore, the contract reserves an activation fee computed as `10 ** coreCollateralDecimals`, which is exactly 1 USDC in Core units. This assumes that HyperCore's activation fee will remain exactly one quote token, but the fee is a HyperCore protocol parameter and HyperLiquid adjusts protocol parameters through governance. If the fee quantity ever changes (although unlikely), the `spotSend` lands short or fails on the Core side, and the user is debited internally without receiving the full amount. The value can only be corrected by upgrading the contract.

## RECOMMENDATION

Make the activation fee an owner-settable parameter, so it can be updated without a contract upgrade if HyperCore changes it.

## RESOLUTION

Status: Acknowledged.

**Papertrade Team:** We acknowledge the integration assumption and request Informational treatment. The implementation reserves one quote token only when HyperCore reports that the destination is not activated, matching the rule assumed by the packet. Existing active destinations are unaffected. Because Exchange is upgradeable, an announced HyperCore parameter change can be incorporated, tested, and governed before withdrawals under changed semantics are supported. A mutable owner-set activation fee would add configuration error surface without removing the need for protocol monitoring and compatibility review. Unexpected rejected delivery remains the L-01 recovery case.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-03

ACKNOWLEDGED

# Timelock Delay Can Be Less Than 7 Days

CATEGORY Trust Assumptions

LOCATION <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/governance/GovernanceTimelock.sol#L79-L82>

REVIEW Main Review

## DESCRIPTION

GovernanceTimelock::updateMinDelay enforces no minimum, so minDelay can be set to 0:

```
``plain text
```

```
function updateMinDelay(uint256 newMinDelay) external onlySelf {
 minDelay = newMinDelay;
 emit MinDelayUpdated(newMinDelay);
}
```

The owner can schedule updateMinDelay(0), wait the current delay, execute it, and from then on schedule and execute operations with no delay, potentially in the same block. The update is itself timelocked once, so observers get one full delay period to notice the reduction, but after that the delay guarantee is gone permanently.

This contradicts the documentation which states that every owner action, including upgrades, is queued for 7 days so users have a week to exit ahead of any change. The accurate guarantee is "every owner action is queued for the current minDelay, unless governance first queues and executes a reduction of that delay."

Note this is also reachable through the constructor, which accepts any initialMinDelay including 0 with no validation.

## RECOMMENDATION

Enforce a minimum delay validation in both the setter and the constructor:

```
``plain text
```

```
uint256 public constant MIN_DELAY_FLOOR = 7 days;
```

## I-03 Timelock Delay Can Be Less Than 7 Days

CONTINUED

```
function updateMinDelay(uint256 newMinDelay) external onlySelf {
 if (newMinDelay < MIN_DELAY_FLOOR) revert InvalidDelay();
 minDelay = newMinDelay;
 emit MinDelayUpdated(newMinDelay);
}
...
```

Either keep the floor in line with the documented 7-day guarantee, or update the docs to state that the delay is itself governable.

### — RESOLUTION

Status: Acknowledged.

**Papertrade Team:** We acknowledge I-20 as a governance-configuration risk and request Informational treatment. The resubmission release configuration requires `TIMELOCK_DELAY_SECONDS=604800` (seven days). Its release gate checks the canonical resubmission manifest and deployed on-chain `minDelay` for exact equality before activation, and its deployer rejects a different value. The seven-day setting remains governable rather than an immutable Solidity floor. Because `updateMinDelay` is callable only by the timelock, any later reduction must itself be scheduled and wait the delay active at scheduling time. User-facing governance material for that release will disclose both the seven-day delay and its governable nature.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-04

RESOLVED

# Factory Initialization Checks Not Enforced

CATEGORY Warning

LOCATION <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/DepositProxyFactory.sol#L20>

REVIEW Main Review

## DESCRIPTION

`DepositProxyFactory.initialize` lacks access control and `deployProxy` is callable while exchange is still `address(0)`. The `Guardian Findings Triage` mentioned the access control case, however it's also important to note that `deployProxy` is callable while exchange is still `address(0)`. This allows front-running initialization and permanently poisoning a user's `proxyOf` entry with a proxy set to a zero address or attacker-chosen exchange, since `deployProxy` early-returns on any existing entry. In the actual deployment this is not exploitable since every deploy path puts the factory behind `AdminUpgradeableProxy`, whose constructor delegatecalls `initialize(exchangeProxy)` in the same transaction, so the factory is never externally callable with `exchange == address(0)`. However, if the implementation and initializing are ever done in separate transactions, this can become exploitable, since the contract relies on the deploy script rather than enforcing it itself.

## RECOMMENDATION

Consider reverting in `deployProxy` when `exchange == address(0)`, and adding access control in `initialize`. Document atomic deployment and initialize as a requirement.

## RESOLUTION

Status: Resolved.

**Papertrade Team:** We partially confirm I-01 as Informational defense in depth. All supported deployments initialize `DepositProxyFactory` atomically through `AdminUpgradeableProxy` constructor delegatecall. The remediation additionally rejects a zero Exchange address in both factory profiles and requires a nonzero initialized Exchange before `deployProxy` can write or create a user proxy. Tests cover uninitialized use, zero-address initialization, and immediately usable atomic constructor initialization. Caller-based initializer authorization is not added because fresh proxy storage has no pre-existing owner to authenticate; atomic initialization remains the primary invariant.

**I-04** Factory Initialization Checks Not Enforced

CONTINUED

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-05

ACKNOWLEDGED

# Unused Modifier and Initializer Disable Helper

CATEGORY	Superfluous Code
LOCATION	<a href="https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/proxy/Initializable.sol#L24-L32">https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/proxy/Initializable.sol#L24-L32</a>
REVIEW	Main Review

## DESCRIPTION

`Initializable.sol` defines `onlyInitializing` and `_disableInitializers`, but neither is used by any contract in scope. If used, `_disableInitializers` would lock the implementation contracts against direct initialization, preventing initialization outside the proxy.

## RECOMMENDATION

Remove `onlyInitializing` if it isn't needed, and call `_disableInitializers()` in the constructors of the upgradeable implementations, or remove it if that's not intended.

## RESOLUTION

Status: Acknowledged.

**Papertrade Team:** We acknowledge I-02 as Informational source hygiene and make no Solidity change. Both helpers are internal and expose no callable surface. Initializing an implementation affects only implementation storage and grants no authority over proxy state. Canonical proxy storage is initialized atomically through constructor `delegatecall`. Adding implementation-locking constructors would not harden proxy storage and would require a repository-wide direct-deployment test migration, so the current lifecycle is retained.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/commit/48e552b2503236f6e2c17c2a2bed1c54504da409>

## INFORMATIONAL I-06

RESOLVED

# Unused sweepToExchange function

CATEGORY	Superfluous Code
LOCATION	<a href="https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Create2DepositProxy.sol#L45">https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Create2DepositProxy.sol#L45</a>
REVIEW	Main Review

## DESCRIPTION

`Create2DepositProxy` exposes two sweep functions, `sweepToExchange` (single amount) and `sweepAllToExchange` (spot plus perp). The Exchange's sweep flow only ever calls `sweepAllToExchange`, from `_requestSweep`. Nothing in the contracts calls `sweepToExchange`, so it's unreachable in the deployed system and currently serves as dead code.

## RECOMMENDATION

Remove `sweepToExchange` if the all-balance sweep is the only intended path, or use it in a single-amount sweep if it is meant to be used.

## RESOLUTION

Status: Resolved.

**Papertrade Team:** We confirm I-03. `Create2DepositProxy.sweepToExchange` is removed; Exchange uses only `sweepAllToExchange`, which covers spot-only and combined perp/spot sweeps. Existing Exchange-only authorization and sweep-all behavior remain covered, and a negative regression proves the legacy selector is absent. Removing the endpoint changes proxy creation code and every deterministic raw deposit address. The production deployment will therefore start a fresh deployment epoch, publish addresses only from the exact reviewed creation-code hash, and not treat old testnet or derived-but-undeployed addresses as compatible. Existing already-deployed proxies remain usable only through their recorded `proxyOf` mapping.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-07

ACKNOWLEDGED

# Liquidations Pay No Incentive to the Caller

CATEGORY Best Practices

LOCATION <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L977>

REVIEW Main Review

## DESCRIPTION

`liquidate` and `liquidateBatch` are permissionless, but the caller receives nothing for liquidating. The liquidation fee (when enabled) is split between the `staker` and `dev` accumulators, and the seized margin goes to the `LP` or `sideBucket`. The liquidator gets no reward while paying for gas.

In practice, with no reward, the liquidation reliability depends almost entirely on the protocol keeper. If that keeper is delayed or down during a fast move, positions that should be liquidated can sit open past their bust price, which works against the LP, since the bust buffer is sized assuming prompt liquidation.

## RECOMMENDATION

Consider adding a small caller incentive (a bounty or gas reimbursement out of the seized margin or fee) so the permissionless path is self-sustaining. If liquidation is intended to rely on the protocol keeper, make sure keeper monitoring and uptime are treated as prioritized.

## RESOLUTION

Status: Acknowledged.

**Papertrade Team:** We acknowledge I-04 as Informational. `liquidate` and `liquidateBatch` are permissionless, but the contract pays no caller bounty and therefore creates no economic guarantee that an independent third party will submit liquidations. Adding a caller bounty would change LP, staker, and developer economics and introduce liquidation-race and MEV incentives requiring separate review. We retain the current contract behavior and request Informational treatment.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-08

RESOLVED

# AlreadySwept Revert Name is Misleading

CATEGORY Error

LOCATION <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L1698>

REVIEW Main Review

## DESCRIPTION

In `_requestSweep`, the per-block guard reverts with `AlreadySwept : if (lastSweepBlock[proxy] >= block.number) revert AlreadySwept();`

At this point nothing has actually been swept. The check only blocks a second sweep request for the same proxy in the same block, and the sweep itself isn't completed until `_finalizeSweep` runs in a later block. The name implies the funds already moved, when the real condition is that a request was already made this block.

This can mislead an integrator or anyone debugging a reverted transaction into thinking the sweep completed, when they actually just need to wait a block to request again.

## RECOMMENDATION

Rename `AlreadySwept` to something that reflects the actual condition, such as `SweepAlreadyRequested`.

## RESOLUTION

Status: Resolved.

**Papertrade Team:** We confirm I-05. The custom error is renamed to `SweepAlreadyRequestedThisBlock`, accurately describing the per-proxy, per-block request guard without implying that HyperCore movement or later internal credit completed. A focused regression requests a sweep, clears its outstanding record, retries in the same block, and asserts the new selector. All tracked Exchange-profile ABIs use the same reviewed selector. This intentional ABI change targets the fresh production deployment and changes no storage or control flow.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-09

RESOLVED

# Spot Balance Guard Shorter Than Decoded Width

CATEGORY Validation

LOCATION <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L1736-L1742>

REVIEW Main Review

## DESCRIPTION

`_readSpotBalance` decodes three `uint64` fields but only guards that the returned data is at least 32 bytes:

```
plain text
```

```
if (!ok || result.length < 32) return 0;
(uint64 total,,) = abi.decode(result, (uint64, uint64, uint64));
`
```

The `SpotBalance` struct is three `uint64`s, which ABI-encode as three 32-byte words, so the `decode` reads 96 bytes. The guard only confirms one word is present. If the precompile ever returned a result between 32 and 95 bytes, the length check would pass and the `abi.decode` would then revert on the short data, rather than the function cleanly returning 0 as the guard intends.

In practice the precompile returns the full 96 bytes, so this should not trigger, but is the guard width doesn't match what's actually decoded, so the early-return safety net doesn't cover the case it looks like it's meant to.

## RECOMMENDATION

Change the guard to `result.length < 96` so it matches the three-word decode.

## RESOLUTION

Status: Resolved.

**Papertrade Team:** We confirm I-06. `_readSpotBalance` now requires at least 96 return bytes before decoding `(uint64,uint64,uint64)`. Failed or shorter Core reads fail closed with `NoCoreBalance`; a valid 96-byte response is unchanged. Regressions cover 0-, 31-, 32-, and 95-byte malformed responses plus a complete tuple. The change adds no storage or external production function.

**I-09** Spot Balance Guard Shorter Than Decoded Width

CONTINUED

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-10

RESOLVED

# Unused SafeTokenLib

CATEGORY	Superfluous Code
LOCATION	<a href="https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/SafeTokenLib.sol">https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/SafeTokenLib.sol</a>
REVIEW	Main Review

## DESCRIPTION

SafeTokenLib provides `safeTransfer`, `safeTransferFrom`, and `safeBalanceOf` for non-standard ERC-20s, but no contract in scope calls it. The most relevant case is `Create2DepositProxy.withdrawStuckToken`, which has its own transfer success check instead of using `SafeTokenLib.safeTransfer`. The library is dead code as written, and the one place that would benefit from it doesn't use it. Note that `SafeTokenLib._didSucceed` does not include a no-code (`code.length > 0`) check, so it isn't equivalent to OpenZeppelin's SafeERC20.

## RECOMMENDATION

Either use `SafeTokenLib` where token transfers happen (i.e., `withdrawStuckToken`) and add a `code.length > 0` check to `_didSucceed`, or remove the library if it's not intended to be used.

## RESOLUTION

Status: Resolved.

**Papertrade Team:** We confirm I-07. `SafeTokenLib.sol` is removed because no in-scope contract imported or called it. The stuck-token recovery path retains its explicit transfer return-data validation; no new dependency is introduced merely to consume dead code. Deleting an unlinked internal library changes no deployed ABI, storage, runtime behavior, or gas path.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-11

ACKNOWLEDGED

# Closing a Liquidatable Position Reclaims Buffer

CATEGORY	Logical Error
LOCATION	Exchange._liquidatePosition, QueueLib.settleUsd
REVIEW	Main Review

## — DESCRIPTION

A position that already meets the liquidation condition can still be closed. Both paths are valid at the same price, but they settle differently: the `close path` computes realized `PnL` and returns residual equity to the user, while liquidation wipes the full margin. So the owner of a liquidatable position holds a settlement option, close to reclaim the residual, or self-liquidate (the function is permissionless) to mint PAPER on the full size.

The residual is leverage-amplified, so at the bust boundary it scales as roughly  $\text{size} * \text{bustBuffer} * (\text{leverage} - 1)$ . At 10x on a 10 USD margin position the close path returns about 0.0428 USD plus 975.8 PAPER, while liquidation returns 0 USD plus 1000 PAPER. The fraction grows with leverage: about 0.04% of margin at 2x, 4.72% at 100x, and roughly 47.5% at the protocol's 1000x maximum.

The consequence is that the bust buffer does not provide reliable safety for the LP. The buffer exists to reserve a liquidation safety margin so the LP captures the position before equity reaches zero, but because closing a liquidatable position refunds exactly that margin to the user, the buffer is only captured by the LP if a keeper or liquidator executes before the user closes. Since liquidators receive no reward, there is no profit motive to win that race, so the outcome at the bust boundary is decided by transaction ordering rather than by a single liquidation rule. A searcher can also front-run a user's close with a liquidation to deny them the residual (though that is unprofitable grieving).

This also lets the owner choose the PAPER mint basis. `Close` mints on realized loss, while self-liquidation mints on full margin, so a user wanting emissions can self-liquidate a position to maximize the basis.

## — RECOMMENDATION

Consider the following options: return residual equity minus an explicit liquidation penalty so liquidation and close paths settle to the same value, block `close` once the liquidation condition is met (forcing the liquidation path), or standardize the PAPER mint basis across both.

---

**I-11** Closing a Liquidatable Position Reclaims Buffer**CONTINUED**

---

**— RESOLUTION**

Status: Acknowledged.

**Papertrade Team:** I-16 should be cross-referenced to I-23 rather than counted as a second issue. A voluntary close landing first settles realized PnL and may return residual isolated margin; a liquidation or self-liquidation landing first treats the position as fully busted and forfeits the complete isolated margin. PAPER follows the terminal branch actually selected and settled: the branch-specific PAPER-eligible realized-loss basis, net of the documented LP-side fee allocation where applicable, for voluntary close; full forfeited margin for liquidation. A user may voluntarily surrender residual USD for the larger hard-bust PAPER basis, consistent with the disclosed loss-farming design. The interface now warns: "Liquidation is a hard bust: crossing the displayed bust price forfeits the full isolated margin, even if a voluntary close immediately beforehand could return some equity." We request Informational treatment.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/blob/7a4917b5d3ad3628dab1bfac8d92ab297f46f381/reviewer-evidence/runtime-patches/runtime-controls.patch>

## INFORMATIONAL I-12

RESOLVED

# sideBucket Flush Does Not Fire After settleUsd

CATEGORY Superfluous Code

LOCATION <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L1242-L1244>

REVIEW Main Review

## DESCRIPTION

`_settleFundingToAccumulator` guard `if (queue.totalQueued == 0 && sideBucket > 0) sideBucket = 0;` appears after the `settleUsd` call in the close flow, with a comment indicating it flushes orphaned `sideBucket` if `settleUsd` settlement drained the queue to zero. In the current implementation `settleUsd` never decrements `totalQueued`, rather it only enqueues and its loss branch leaves the user's queue debt intact. So the condition can only hold if the queue was already empty before the call, in which case `sideBucket` should already be zero, so the guard does not fire.

## RECOMMENDATION

Remove the guard or correct the comment to note that `settleUsd` no longer drains the queue so the flush is defensive only.

## RESOLUTION

Status: Resolved.

**Papertrade Team:** We confirm I-08. The impossible `sideBucket` cleanup branch and inaccurate comment immediately after USD `settleUsd` are removed. `settleUsd` can add a winning claim or settle a loss, but it does not decrement `queue.totalQueued`, so it cannot own a nonzero-to-zero queue transition. Cleanup remains at the two reachable sites: after debt-claim consumption and after harvest drains the queue. Under the queue invariant, removal changes no accounting result and removes an unnecessary close-path branch/read.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-13

ACKNOWLEDGED

# Owner Can Grant Infinite Spend Authority

CATEGORY Trust Assumptions

LOCATION <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/PaperToken.sol#L98>

REVIEW Main Review

## DESCRIPTION

`PaperToken.allowance` returns `type(uint256).max` for the `autoApprovedSpender`, and `transferFrom` skips the allowance check entirely when the caller is that address:

```
``plain text
function allowance(...) { if (spender == autoApprovedSpender) return type(uint256).max; }
function transferFrom(address from, ...) {
if (msg.sender != autoApprovedSpender) { /* allowance check */ }
_transfer(from, to, amount);
}
`
```

So whoever is set as `autoApprovedSpender` can move any holder's PAPER, in any amount, without that holder's approval. `setAutoApprovedSpender` is onlyOwner, so the owner can point this capability at an arbitrary address at any time, conferring standing, infinite, consent-free transfer rights over every holder's balance until it is changed.

Due to this, all holders' PAPER is ultimately at the owner's control, and holders have no opt-out. In addition, the owner is the timelock contract, which is owned by the 1-of-2 multisig where control of either single signer key is sufficient – so a single compromised governance key can set `autoApprovedSpender` to an attacker and drain every holder's PAPER.

The auto-approved-spender mechanism itself is presumably intended (to let protocol contracts move PAPER without per-user approval), but its unbounded scope (any address, infinite, all holders, owner-settable at any time) is the main concern.

## RECOMMENDATION

Constrain the capability rather than leaving it as an open owner setter. Pin `autoApprovedSpender` to a known protocol contract at initialization and make it immutable, or restrict it to an allowlisted set. At minimum, route changes through the timelock so a change is observable on-chain before it takes effect, and document that PAPER holders' balances are spendable by the auto-approved spender.

---

**I-13** Owner Can Grant Infinite Spend Authority

---

**CONTINUED****— RESOLUTION**

Status: Acknowledged.

**Papertrade Team:** We acknowledge I-22 and request Informational treatment. PAPER intentionally grants the configured staking proxy automatic transferFrom authority so staking needs no separate approval. Final governance places the PAPER owner and ProxyAdmin under the same timelock, which can change the spender, upgrade the staking implementation at the pinned proxy, or upgrade PAPER itself. Pinning only the address would reduce configuration error but would not create a security boundary against that governance authority. The relayer-only nontransferable profile constrains ordinary movement, not upgrades. We retain the behavior and disclose the complete governance trust boundary.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-14

RESOLVED

# Perp To Spot Transfer May Falsely Finalize Sweep

CATEGORY Validation

LOCATION <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Create2DepositProxy.sol#L58-L68>, <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L1725>

REVIEW Main Review

## DESCRIPTION

`_finalizeSweep` decides a sweep has settled by checking whether the deposit proxy's Core balances changed since the sweep was requested:

```
``plain text
```

```
if (currentSpot == spotAmount && currentPerp == perpAmount) revert SweepStillPending();
// else: credit spotAmount + perpAmount to balances[user], totalBalances, and treasury
```

It then credits the user's internal balance, `totalBalances`, and `treasury` by the snapshotted `spotAmount + perpAmount`. It never reads the exchange's own Core balance, so it confirms settlement from the proxy's balances moving, not from the exchange actually receiving funds. This signal is unsound because `sweepAllToExchange` performs two distinct `CoreWriter` actions. When `perpAmount > 0`, it first issues a `usdClassTransfer` (perp → spot), which is a transfer within the proxy's own Core account: it moves no funds out of the proxy, yet it changes both reads (spot rises from 0 to P, withdrawable perp falls from P to 0). Only the second action, the outbound `spotSend`, actually moves funds to the exchange.

Because the perp-spot transfer alone makes `currentSpot != spotAmount` and `currentPerp != perpAmount`, the "still pending" check passes and `_finalizeSweep` credits the full snapshot, even if the outbound `spotSend` has not settled or has failed. The settlement signal that is supposed to prove the exchange received the funds is still satisfied.

## RECOMMENDATION

Consider splitting perp and spot sweep settlement into separate phases. A pending sweep with `perpAmount > 0` should not be creditable after only `usdClassTransfer`, the contract must first observe the proxy's perp-to-spot conversion, then require an actual outbound proxy spot decrease and an attributable Exchange-side receipt before internal credit is minted.

## I-14 Perp To Spot Transfer May Falsely Finalize Sweep

CONTINUED

### — RESOLUTION

Status: Resolved.

**Papertrade Team:** We confirm I-09, with a correction to the report's arithmetic: normalized proxy value is  $\text{spot} + 100 * \text{perp}$ , because spot token-0 uses 8 decimals and perp withdrawable notional uses 6. The contract sends and credits that same 8-decimal amount, rejects `uint64` overflow before `CoreWriter` actions, requires a strict decrease in normalized proxy value before finalization, and fails closed on failed or malformed Core reads. Perp-to-spot conversion alone preserves normalized value and cannot finalize. The contract-side remediation is in PR #1 at [<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>](<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>). The formally linked reviewer-accessible, full-index settlement-worker patch uses the identical tested predicate, rejects failed, short, noncanonical, or stale reads, isolates per-proxy read failures, and never automatically clears or retries a stale sweep without evidence that the earlier action cannot land. It exposes stale count and age, read failures, cycle failures, and telemetry health. The remaining integration assumption is explicit: source-proxy decrease is not cryptographic receiver attribution, so exact/all-or-nothing send, no other proxy outflow, and no overlapping stale action are release invariants; exceptional recovery remains privileged and evidence-gated.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/blob/7a4917b5d3ad3628dab1bfac8d92ab297f46f381/reviewer-evidence/runtime-patches/i09-settlement-worker.patch>

## INFORMATIONAL I-15

RESOLVED

## Stale and Inaccurate Code Comments

CATEGORY Documentation

LOCATION `QueueLib.settleUsd`, `Exchange.lpBankrollAvailable`, `QueueLib.compactDustRange`, `Exchange.IPaperTokenomics`

REVIEW Main Review

### DESCRIPTION

Several comments describe behavior the code does not implement. All are comment-only divergences; the code behaves correctly in each case.

`QueueLib.settleUsd` `NatSpec` describes a "debt-first burn" loss branch, solvent "iff `totalQueued == 0` after the burn." The loss branch actually does naive settlement with no burn, as the body comment admits.

`Exchange.lpBankrollAvailable` comment states it "subtracts Q, S, F." It does not arithmetically subtract `sideBucket`; the queue reservation is enforced by the `totalQueued > 0` → return 0 short-circuit instead.

`Exchange._settleFundingToAccumulator` post-`settleUsd` flush guard (if `(queue.totalQueued == 0 && sideBucket > 0) sideBucket = 0;`) carries a comment about `settleUsd` draining the queue. `settleUsd` never decrements `totalQueued`, so the guard is inert at this site (it remains live at the `_openPosition` and harvest sites).

`QueueLib.compactDustRange` does not enforce the aggregate `dustThreshold` ceiling that `compactDust` enforces, so its result can exceed the dust size its "dust compaction" doc implies. Totals are preserved.

`IPaperTokenomics` asserts the invariant `lpCreditedUsd <= mintBasisUsd` but it is unenforced, and the `mintBasisUsd == 0` early return silently drops any `lpCreditedUsd` passed alongside it.

### RECOMMENDATION

Align each comment with the implemented behavior, or update the code to match the documented intent. For the `mintBasisUsd == 0` early return, confirm it is never reached with a non-zero `lpCreditedUsd`; if it can be, handle the credit rather than dropping it.

### RESOLUTION

Status: Resolved.

**Papertrade Team:** We confirm I-10 and accept Informational. Source and packet documentation now describe the implemented settlement, bankroll, `sideBucket`, and per-source

**I-15** Stale and Inaccurate Code Comments**CONTINUED**

compaction rules accurately. I-08 removes the impossible post-`settleUsd` bucket branch. Both tokenomics hooks now reject `lpCreditedUsd > mintBasisUsd` before any state change with `InvalidLpCredit`. Valid Exchange call paths already satisfy that invariant, so the guard changes no valid settlement. The tracked ABI is regenerated and compiler-equality checked.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-16

RESOLVED

# Global Trading Pause Only Blocks Position Opens

CATEGORY Documentation

LOCATION <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L519>

REVIEW Main Review

## DESCRIPTION

`setTradingPaused(true)` sets `tradingPaused`, which is only checked by `_requireTradingActive()` in the open path. Closes, liquidations, and withdrawals do not call `_requireTradingActive()`, so while global trading is "paused" those paths continue to run at the live oracle price. The global pause therefore prevents new positions from being opened but does not halt settlement or fund movement.

This may be intended (`pauseMarket` exists to freeze a specific market's settlement price, while `setTradingPaused` blocks new risk), but an operator reaching for a global pause as an emergency stop may be mistaken, as trades can still be closed and liquidated, and users can still withdraw, during the pause.

## RECOMMENDATION

Document that `setTradingPaused` is opens-only and is not a full emergency stop. If a true emergency halt is desired, add a separate switch that also gates close, liquidate, and withdraw (or have those paths honor the global pause), so operators have an explicit way to freeze all activity.

## RESOLUTION

Status: Resolved.

**Papertrade Team:** We confirm I-11 and accept Informational. `setTradingPaused` remains an opens-only gate: it blocks new exposure while allowing closes, liquidations, real-wallet withdrawals, and maintenance at live prices. The pause guardian may engage it, but only owner/timelock may restore global opens. `pauseMarket` is now a separate one-way kill switch: the market ID is permanently retired, remaining positions settle against one captured midpoint, and `unpauseMarket` always reverts. Restoring global opens cannot revive a retired ID. Relisting occurs only after both old-ID OI sides reach zero and uses a fresh sequential instrument ID. We do not make the global gate trap exits or disable risk-reducing settlement.

**I-16** Global Trading Pause Only Blocks Position Opens

CONTINUED

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-17

RESOLVED

# Redundant State Write Before Revert in `_execute`

CATEGORY Superfluous Code

LOCATION <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/governance/OneOfTwoMultisig.sol#L111-L112>

REVIEW Main Review

## DESCRIPTION

In `OneOfTwoMultisig._execute`, the failure branch sets `txn.executed = false` immediately before reverting:

```
``plain text
txn.executed = true;
(bool ok, bytes memory returnData) = txn.to.call{value: txn.value}(txn.data);
if (!ok) {
 txn.executed = false;
 revert ExecutionFailed(returnData);
}
```

The `txn.executed = false` write has no effect since the subsequent revert rolls back all state changes made during the transaction.

## RECOMMENDATION

Remove the `txn.executed = false` line, since the revert already restores the prior state.

## RESOLUTION

Status: Resolved.

**Papertrade Team:** We confirm I-12 and accept Informational. The redundant `txn.executed = false` assignment is removed from the branch that immediately reverts with `ExecutionFailed`. The pre-call `txn.executed = true` transition remains as the checks-effects-interactions/reentrancy guard. A regression proves that failed execution leaves the transaction retryable and that a later successful retry marks it executed. Storage, selectors, and successful behavior are unchanged.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-18

RESOLVED

# CallExecuted Event Fires for Skipped Calls

CATEGORY Events

LOCATION <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/BatchExecutor.sol#L84-L94>

REVIEW Main Review

## DESCRIPTION

In `BatchExecutor.execute`, the `CallExecuted` event is emitted in three cases, including one where the call is never actually executed. When a call fails preflight (e.g. the target/selector is not whitelisted), the function emits `CallExecuted` with `success = false` and returns before ever performing the call:

```
``plain text
bytes memory failure = _preflightFailure(i, c.target, c.data);
if (failure.length != 0) {
 if (!c.allowFailure) revert BatchCallFailed(i, failure);
 failureCount++;
 if (c.groupKey != bytes32(0)) failedGroups[failedGroupCount++] = c.groupKey;
 emit CallExecuted(batchId, i, c.intentId, false, failure); // call never ran
 continue;
}
```

Here no `c.target.call(c.data)` ever occurs, yet the emitted event is named `CallExecuted`. The `success = false` flag does signal the failure, but the event name implies the call was executed (and failed at runtime), whereas in the preflight-rejection branch the call was never dispatched at all. This conflates two distinct outcomes — "executed and reverted" versus "rejected before execution" under the same `CallExecuted(..., false, ...)` event, which can mislead off-chain consumers indexing these events (for example, distinguishing a whitelist rejection from an on-chain revert).

## RECOMMENDATION

Use a distinct event (or a distinct flag/reason field) for preflight-rejected calls so consumers can tell "executed and failed" apart from "rejected before execution."

## RESOLUTION

Status: Resolved.

**I-18** CallExecuted Event Fires for Skipped Calls**CONTINUED**

**Papertrade Team:** We confirm I-13 and accept Informational. `CallRejected(bytes32 indexed batchId,uint256 indexed index,bytes32 indexed intentId,bytes reason)` now represents whitelist or calldata preflight failure before dispatch. `CallExecuted` is reserved for dispatched calls, with its success flag distinguishing success from target revert, and `CallSkipped` remains reserved for dependency/group skips. The contract-side remediation, ABI updates, and event regressions are in PR #1 at [\[https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1\]](https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1) (<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>). The formally linked reviewer-accessible, full-index runtime patch updates the production watcher, PostgreSQL projector, stale-evidence topic set, and receipt tests to recognize and decode the exact four-field event. This observability change does not alter ordering, failure propagation, groups, storage, authorization, or accounting.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/blob/7a4917b5d3ad3628dab1bfac8d92ab297f46f381/reviewer-evidence/runtime-patches/runtime-controls.patch>

## INFORMATIONAL I-19

ACKNOWLEDGED

# Exchange Change Has No Proxy Migration Path

CATEGORY Trust Assumptions

LOCATION <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/DepositProxyFactory.sol#L11-L22>

REVIEW Main Review

## DESCRIPTION

`DepositProxyFactory` is upgradeable (Initializable), so its `exchange` can change, but `Create2DepositProxy.exchange` is `immutable` and the proxy address is CREATE2-derived from it (the constructor args feed the init-code hash used in `computeProxyAddress`).

If exchange ever changes, the two desync. Existing proxies stay permanently bound to the old exchange with no migration path, and `computeProxyAddress(user)` begins returning a new address that does not match the user's already-deployed proxy. Because `deployProxy` is idempotent on `proxyOf\[user\]`, it will not deploy a proxy at the new address for an existing user, so a user directed to the recomputed address could deposit to a location with no controlling proxy and strand funds.

The immutability itself is protective (it prevents an existing proxy from being repointed) but it also means the system cannot migrate existing users to a new exchange.

## RECOMMENDATION

If exchange is meant to be permanent, make the factory's exchange immutable (or the factory non-upgradeable) so it cannot diverge from the deployed proxies. If exchange migration is intended, add a per-user migration path that re-derives and redeploys proxies against the new exchange.

## RESOLUTION

Status: Acknowledged.

**Papertrade Team:** We acknowledge I-14 as Informational. The submitted factory has no supported Exchange-address setter, and implementation upgrades preserve the same Exchange proxy address. A hard Exchange-address change therefore creates a new factory and deployment epoch, updates manifests and every deposit consumer, provisions new user proxies, and retains the old factory/Exchange long enough to sweep legacy or stray deposits. Existing deposit proxies remain intentionally non-retargetable. We do not add a mutable target that would change authorization for already published addresses.

**I-19** Exchange Change Has No Proxy Migration Path

CONTINUED

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-20

RESOLVED

# Staker Fee Recipient Change Redirects Fees

CATEGORY Documentation

LOCATION <https://github.com/GuardianOrg/exchange-contract-audit-guardian-team1-1780258247445/blob/48e552b2503236f6e2c17c2a2bed1c54504da409/contracts/src/Exchange.sol#L504>

REVIEW Main Review

## DESCRIPTION

`setStakerFeeRecipient` updates `stakerFeeRecipient` without first flushing the accrued `stakerFeeAccumulator`:

```
``plain text
function setStakerFeeRecipient(address _recipient) external onlyOwner {
 stakerFeeRecipient = _recipient;
 emit StakerFeeRecipientUpdated(_recipient);
}
```

`stakerFeeAccumulator` is a single global balance paid to whoever is the current `stakerFeeRecipient` at the time `pushStakerFees` is called. Because the setter does not push first, any fees accrued while the previous recipient was configured are paid to the new recipient on the next push, rather than to the recipient that was in place when they were earned. The funds are not lost, but they are redirected from the outgoing recipient to the incoming one. Since the recipient is the staking contract, this hands the previous staking contract's pending distribution to the new one, affecting rewards owed to existing stakers.

## RECOMMENDATION

Call `pushStakerFees` (flushing the accumulator to the current recipient) before updating `stakerFeeRecipient`, or have `setStakerFeeRecipient` push automatically before reassigning. Document that the recipient must be flushed prior to any change so accrued fees settle to the recipient that earned them.

## RESOLUTION

Status: Resolved.

**Papertrade Team:** We partially confirm I-15 and accept Informational. Under cash-basis accounting, an unpushed accumulator is not yet attributed to a historical cohort, so following the active recipient is not itself misappropriation. Recipient replacement nevertheless could strand the old staking proxy's allocated reserve, pending user rewards, Exchange

**I-20** Staker Fee Recipient Change Redirects Fees**CONTINUED**

balance, later-harvested queue claims, and creditStakingReward authority. stakerFeeRecipient is now stable initial wiring: the first nonzero assignment succeeds, assigning the same address is idempotent, and any different address reverts. Supported PaperStaking upgrades occur in place through the existing proxy. A new staking address requires a separately audited migration or fresh deployment epoch.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-21

ACKNOWLEDGED

# Relayer only mode bricks scw exits

CATEGORY	Logical Error
LOCATION	SessionKeyManager.sol
REVIEW	Main Review

## — DESCRIPTION

In ExchangeRelayerOnly, direct user trading is disabled. direct close() always reverts with DirectEconomicCallDisabled, and the only close route is closeBySig(), gated behind onlyAuthorizedExecutor

That signed close route then falls back to base Exchange.closeBySig, which consumes a nonce and requires sessionKeyManager.requireValidSessionSignature() before calling \_close(), so even an honest authorized executor cannot close unless the user already has a currently valid session key signature

The problem is that SessionKeyManagerRelayerOnly disables direct session key registration, while registerSessionKeyBySig ultimately uses the base manager's ECDSA only verification

\_recoverSigner() / ecrecover must recover exactly the user address. there is no EIP-1271 isValidSignature fallback for contract wallets, So if

1. A Gnosis Safe, account abstraction wallet, or other contract wallet has an open USD funded position
2. The system is migrated to the relayer only profile
3. The contract wallet has no currently valid EOA session key, or its previous session key expires/revokes
4. The contract wallet cannot call ExchangeRelayerOnly.close() because direct close is disabled
5. It cannot register/renew a session key directly because SessionKeyManagerRelayerOnly.registerSessionKeyDirect() reverts
6. It cannot register/renew via registerSessionKeyBySig() because the real wallet is a contract and cannot satisfy ecrecover() = user
7. Therefore no valid closeBySig can be produced, and even the authorized executor cannot close the user position

For affected smart contract wallet users, open positions can become unclosable. Their margin remains locked in totalLockedMargin, their position remains exposed to market movement, and the only ordinary terminal path may be third party liquidation

**I-21** Relayer only mode bricks scw exits

CONTINUED

This can only happen if relayer only migration can occur while contract wallet users have open positions

**— RECOMMENDATION**

Add EIP-1271 support for wallet session key registration

**— RESOLUTION**

Status: Acknowledged.

**Papertrade Team:** We acknowledge I-18 only as a conditional migration concern and request Informational treatment. The selected launch is a fresh relayer-only stack with no inherited positions and supports EOA EIP-712 signatures rather than generic EIP-1271 wallets. For any exceptional open-to-relayer-only transition, governance first engages the temporary global opens gate, preserves live-price exits, and enumerates every ID below nextPositionId at a pinned block to prove all positions are closed. `totalLockedMargin == 0` is insufficient because debt-funded positions are excluded, and `terminal pauseMarket` is not used because it permanently retires IDs and freezes settlement. If exhaustive zero-position proof fails, the transition uses a fresh deployment.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/pull/1>

## INFORMATIONAL I-22

ACKNOWLEDGED

# Phantom deposit credits before exchange activated

CATEGORY Logical Error

LOCATION .

REVIEW Main Review

## DESCRIPTION

Hyperliquid charges a 1 quote token activation fee for the first transfer to a new HyperCore Exploit

1. Exchange is deployed, but its HyperCore account is not activated yet
2. Attacker funds their deposit proxy with X Core USDC
3. requestSweep(attacker) snapshots X and queues proxy spotSend(proxy -\> Exchange, X)
4. HyperCore later rejects the spotSend because the Exchange is a fresh destination and the proxy is trying to send its entire balance ( exactly the swept amount ), leaving no extra USDC to pay the activation fee
5. Proxy still has X, Exchange received 0
6. Attacker dusts proxy by 1 raw unit
7. finalizeSweep(attacker) sees proxy balance changed and credits X internally

The key here is that this does not trigger against an already activated / funded Exchange, so this attack can only work in the time between deployment and activation of the Exchange

If an attacker exploited it, he could credit himself any usdc amount in the exchange accounting and withdraw it when that amount becomes available in the system later. If attacker were fast after deployment, he could stay hidden with millions of USDC in the accounting and drains the TVL later

## RECOMMENDATION

we need to not allow requestSweep until the Exchange HyperCore account is activated

```
``plain text
if (!_coreUserExists(address(this))) revert CoreDestinationNotActivated();
``
```

---

**I-22** Phantom deposit credits before exchange activated**CONTINUED**

---

**— RESOLUTION**

Status: Acknowledged.

**Papertrade Team:** We confirm the deployment-window failure mode and request Informational treatment. Deployment stops after the Exchange is created, activates that exact address through the known one-time \[extract\_itex]1 HyperCore transfer, and proves at pinned blocks that chain and address match, `proxyFactory` remains unset, Core User Exists is true, and the receive sentinel landed. The same check runs again immediately before factory wiring. Proxy exposure, manifest publication, deposit settlement, and deposit intake remain blocked until both checks pass. This is equivalent to a mandatory initialization step: adding an activation read to every sweep would impose permanent gas cost to address a one-time state. I-09 separately addresses asynchronous sweep finality.

<https://github.com/the1xinoe/exchange-contract-audit-guardian/blob/7a4917b5d3ad3628dab1bfac8d92ab297f46f381/reviewer-evidence/runtime-patches/c01-deployment-gate.patch>

## § 05

# 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.

## § 06

# About Guardian.

Guardian is a smart contract security firm. Every engagement is worked by two independent teams competing on the same scope, each with its own tooling and methods, so no single reviewer's blind spot survives.

**\$45B+**DIGITAL ASSETS UNDER  
REVIEW**200+**

ENGAGEMENTS COMPLETED

**500+**CRITICAL & HIGH FINDINGS  
REPORTED

Reach us at [audits@guardianaudits.com](mailto:audits@guardianaudits.com).