

SYNTHETIX · SECURITY REVIEW

Synthetix Withdrawal Flow.

A security review of Withdrawal Flow.

CLIENT	PROJECT	AUDITORS	AUDIT WINDOW	FINAL REPORT
Synthetix	Withdrawal Flow	Daniel Gelfand	December 1 – December 15, 2025	December 15, 2025

§ 00

Contents.

107 pages: overview, confidence ranking, scope, 83 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. 09
	Main Review · 83 findings	P. 13
§ 05	Disclaimer	P. 106
§ 06	About Guardian	P. 107

§ 01

Overview.

Guardian reviewed Withdrawal Flow across a single review round. The final report includes 17 High, 21 Medium, 25 Low, 20 Informational findings.

AUDIT FIRM	Guardian	REPOSITORY	https://github.com/GuardianOrg/v4-offchain-team1-1764269086710, https://github.com/Synthetixio/v4-offchain
CLIENT	Synthetix	METHODOLOGY	Smart contract · Manual · Main Review
PROJECT	Withdrawal Flow	MAIN REVIEW COMMIT	e6a7d6ceea4c
PREPARED BY	Daniel Gelfand		
AUDIT WINDOW	December 1 – December 15, 2025		
FINAL REPORT	December 15, 2025		

1.1 Confidence Ranking

1 / 5

Very Low Confidence

Guardian assigns a Confidence Ranking of 1 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 Findings at a Glance

SEVERITY	TOTAL	PENDING	ACK	PARTIAL	RESOLVED
Critical	0	0	0	0	0
High	17	17	0	0	0
Medium	21	21	0	0	0
Low	25	25	0	0	0
Informational	20	20	0	0	0
All severities	83	83	0	0	0

§ 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	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 ASSIGNED	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 Withdrawal Flow in <https://github.com/GuardianOrg/v4-offchain-team1-1764269086710>, <https://github.com/Synthetixio/v4-offchain>.

The scope includes 25,915 source lines across 73 files.

ALSO IN SCOPE

SynthetixDepositContract.sol (Onchain Deposit Contract Scope; repository Synthetixio/v4-deposit-contract, no commit stated)

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 1/3.

FILE	NSLOC	TOTAL
services/relayer/internal/api/health.go	76	105
services/relayer/internal/blockchain/client.go	12	16
services/relayer/internal/blockchain/erc20.go	69	90
services/relayer/internal/blockchain/listener.go	489	620
services/relayer/internal/blockchain/simulated_backend.go	83	137
services/relayer/internal/blockchain/synthetic_deposit_contract.go	4,294	6,559
services/relayer/internal/blockchain/test_helpers.go	13	17
services/relayer/internal/config/config.go	142	208
services/relayer/internal/core/auto_approve_watcher.go	289	368
services/relayer/internal/core/cow_order_subscriber.go	90	106
services/relayer/internal/core/transaction_manager.go	331	421
services/relayer/internal/core/watcher.go	132	172
services/relayer/internal/core/withdrawal_subscriber.go	368	486
services/relayer/internal/coworder/service.go	488	562
services/relayer/internal/database/database.go	66	97
services/relayer/internal/database/redis_database_cluster.go	265	381
services/relayer/internal/kms/signer.go	22	33
services/relayer/internal/metrics/metrics.go	61	73
services/relayer/internal/services/types.go	7	13
services/subaccount/internal/config/config.go	122	152
services/subaccount/internal/constants/constants.go	63	102
services/subaccount/internal/constants/tokens.go	52	73
services/subaccount/internal/handlers/grpc_handler.go	1,452	1,916
services/subaccount/internal/middleware/auth_interceptor.go	84	118
services/subaccount/internal/nats/client.go	206	275
services/subaccount/internal/repository/collateral_repository.go	167	221
services/subaccount/internal/repository/funding_rate_history_repository.go	31	43
services/subaccount/internal/repository/leverage_repository.go	110	148
services/subaccount/internal/repository/order_history_repository.go	57	76
services/subaccount/internal/repository/position_repository.go	148	203
services/subaccount/internal/repository/subaccount_repository.go	445	607
services/subaccount/internal/repository/trade_history_repository.go	117	188
services/subaccount/internal/repository/withdraw_repository.go	78	107
services/subaccount/internal/services/subaccount/atomic_position_worker_pool.go	200	246
services/subaccount/internal/services/subaccount/event_handlers.go	1,077	1,389
services/subaccount/internal/services/subaccount/metrics.go	40	47

Files in Scope 2/3.

FILE	NSLOC	TOTAL
services/subaccount/internal/services/subaccount/open_order_worker_pool.go	191	238
services/subaccount/internal/services/subaccount/order_history_worker_pool.go	159	184
services/subaccount/internal/services/subaccount/service.go	2,877	3,798
services/subaccount/internal/services/subaccount/state_event_handler.go	154	190
services/subaccount/internal/services/subaccount/trade_history_worker_pool.go	158	183
services/subaccount/internal/services/subaccount/withdrawal_validation_handler.go	65	81
services/subaccount/internal/services/subaccount/worker_event_handlers.go	179	198
services/subaccount/internal/services/subaccount/worker_pool.go	195	247
services/subaccount/internal/services/subaccount/worker_subscriptions.go	202	242
services/subaccount/internal/types/funding_payments.go	22	29
services/subaccount/internal/types/open_order.go	19	23
services/subaccount/internal/types/subaccount.go	27	37
services/subaccount/internal/types/withdraw.go	24	33
services/subaccount/internal/utils/open_order_converter.go	48	61
services/subaccount/internal/utils/order_history_converter.go	172	219
services/subaccount/internal/utils/trade_history_converter.go	50	57
services/subaccount/internal/utils/validation.go	26	39
services/trading/internal/handlers/withdrawal_handler.go	113	141
services/trading/internal/subaccount/actor_position_order.go	99	146
services/trading/internal/subaccount/actor_slp_handler.go	75	95
services/trading/internal/subaccount/actor_tick.go	334	430
services/trading/internal/subaccount/actor_withdrawal.go	204	280
services/trading/internal/subaccount/actor.go	928	1,265
services/trading/internal/subaccount/constant.go	6	7
services/trading/internal/subaccount/events.go	600	728
services/trading/internal/subaccount/leverage_update.go	120	158
services/trading/internal/subaccount/liquidation_engine_process_state.go	629	899
services/trading/internal/subaccount/liquidation_engine.go	813	1,126
services/trading/internal/subaccount/liquidation_state_machine.go	126	190
services/trading/internal/subaccount/liquidation_types.go	76	111
services/trading/internal/subaccount/manager.go	1,109	1,430
services/trading/internal/subaccount/margin_calculator.go	516	823
services/trading/internal/subaccount/orders.go	2,828	3,677
services/trading/internal/subaccount/response_wrapper.go	42	60
services/trading/internal/subaccount/slp_transfer.go	548	717
services/trading/internal/subaccount/test_helpers_integration.go	220	268

Files in Scope 3/3.

FILE	NSLOC	TOTAL
services/trading/internal/subaccount/test_helpers.go	215	274

§ 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	Watchers Unable To Cast Votes On Set Contract	DoS	HIGH	PENDING	p. 13
H-02	Race Condition In CreateActor()	Frontrunning	HIGH	PENDING	p. 14
H-03	Withdrawal Disputes Not Implemented	Logical Error	HIGH	PENDING	p. 15
H-04	Withdrawal Pause Stalls Entire Event Listener	Logical Error	HIGH	PENDING	p. 16
H-05	Token Specific Pauses Entirely Ignored	DoS	HIGH	PENDING	p. 17
H-06	Lost Withdrawal Requests With Unconditional ACK	Logical Error	HIGH	PENDING	p. 18
H-07	Closed Channel Will Panic Actor	DoS	HIGH	PENDING	p. 19
H-08	No Rate Limiting on Withdrawal Requests	DoS	HIGH	PENDING	p. 20
H-09	Missing Timeouts Enable Slowloris DoS	DoS	HIGH	PENDING	p. 21
H-10	WebSocket Lack Per-IP Quotas	DoS	HIGH	PENDING	p. 22
H-11	Unbounded Request Bodies Lead to OOM	DoS	HIGH	PENDING	p. 23
H-12	Status Update Can Overwrite Balances	Frontrunning	HIGH	PENDING	p. 24
H-13	WS Message Replay Due To Lack of Freshness Check	Logical Error	HIGH	PENDING	p. 25
H-14	Shared Key For Relayer, Teller and Watcher	Logical Error	HIGH	PENDING	p. 26
H-15	Trade Action Signing Omits Payload Fields	Logical Error	HIGH	PENDING	p. 28
H-16	Missing NATS Handlers for Internal Transfers	Logical Error	HIGH	PENDING	p. 29
H-17	SLP Handoff Ignores ImmediateSLPTakeover Breach	Validation	HIGH	PENDING	p. 30
M-01	Panic Due To Missing SLP Actor	DoS	MEDIUM	PENDING	p. 32
M-02	Deposit/Withdraw Responses Lacks Idempotency	Logical Error	MEDIUM	PENDING	p. 33
M-03	Lock Held For 5 Minutes Blocks Other Tx's	DoS	MEDIUM	PENDING	p. 34
M-04	Subaccount ID Overflow in Relayer	Logical Error	MEDIUM	PENDING	p. 35
M-05	Reduce-Only Orders Inflate Margin Calculations	Logical Error	MEDIUM	PENDING	p. 36

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
M-06	Off-chain Withdrawal Minimum Not Validated	Gas Griefing	MEDIUM	PENDING	p. 37
M-07	Gap In ProcessTransfer Relationship Validation	Access Control	MEDIUM	PENDING	p. 38
M-08	Actor Crash Causes Balance Desync	Logical Error	MEDIUM	PENDING	p. 39
M-09	Collateral Jobs Never Processed	Logical Error	MEDIUM	PENDING	p. 40
M-10	One Failed Publish Bricks TP/SL	Error	MEDIUM	PENDING	p. 41
M-11	Rate Limiter Bypass With Multi-Instance	DoS	MEDIUM	PENDING	p. 43
M-12	Matching Engine Responses Dropped	Logical Error	MEDIUM	PENDING	p. 44
M-13	GetCandlesticks Unrestricted Range Enables DoS	DoS	MEDIUM	PENDING	p. 45
M-14	Pending Reduce-Only Orders Not Adjusted	Logical Error	MEDIUM	PENDING	p. 46
M-15	Pending Orders Ignored In RMR Simulation	Logical Error	MEDIUM	PENDING	p. 48
M-16	Deposit/Withdrawal Acked Before Processing	Logical Error	MEDIUM	PENDING	p. 49
M-17	Teller Role Does Not Cancel Stale Withdrawals	Logical Error	MEDIUM	PENDING	p. 50
M-18	Partial Liquidation Forces SLP Takeover	Logical Error	MEDIUM	PENDING	p. 52
M-19	Panic Due To Closed Channel	DoS	MEDIUM	PENDING	p. 53
M-20	Liquidation Attempt State Resets On Crash	Logical Error	MEDIUM	PENDING	p. 55
M-21	Assets Disappear When SLP Transfer Fails	Unexpected Behavior	MEDIUM	PENDING	p. 56
L-01	Deposit Spec and Contract Inconsistencies	Documentation	LOW	PENDING	p. 58
L-02	Symbol Not Validated At API Layer	Validation	LOW	PENDING	p. 59
L-03	Auto Approve Watcher Votes On Already Voted IDs	Gas Griefing	LOW	PENDING	p. 60
L-04	Withdrawal Completion Events Lack NATS Dedupe	Warning	LOW	PENDING	p. 61
L-05	Stale Bindings Present	Warning	LOW	PENDING	p. 62
L-06	Non-USDT Collateral Not Supported	Warning	LOW	PENDING	p. 63
L-07	Zero Address Passes API	Validation	LOW	PENDING	p. 64
L-08	Withdrawals Are Approved Using Stale Margin Data	Logical Error	LOW	PENDING	p. 65
L-09	Silent Failures Return Zero Margin	Warning	LOW	PENDING	p. 66
L-10	Simultaneous Deposits Can Drop One	Logical Error	LOW	PENDING	p. 67
L-11	Wildcard Test Deposit Endpoint	Warning	LOW	PENDING	p. 68
L-12	Cancels Are Not Rate Limited	Warning	LOW	PENDING	p. 69
L-13	Trade Safety Check Uses Optimistic Fill Prices	Logical Error	LOW	PENDING	p. 70

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
L-14	Worker Pool Panic Causes Permanent Message Loss	DoS	LOW	PENDING	p. 71
L-15	Auto-Exchange Double-Deducts Collateral	Logical Error	LOW	PENDING	p. 72
L-16	No Input Length Validation on Decimal Parsing	Validation	LOW	PENDING	p. 74
L-17	Silent Success Response When NATS Unavailable	Warning	LOW	PENDING	p. 75
L-18	Assets Can Disappear During Liquidation	Unexpected Behavior	LOW	PENDING	p. 76
L-19	Collaterals Not Cleared clearLiquidatedAccount	Unexpected Behavior	LOW	PENDING	p. 78
L-20	TP/SL Event Retry	Error	LOW	PENDING	p. 79
L-21	Database Init Error Handling	Suggestion	LOW	PENDING	p. 80
L-22	Unbounded Rebroadcast Loop	Warning	LOW	PENDING	p. 81
L-23	Inconsistent Behaviour In recoveryMode	Validation	LOW	PENDING	p. 82
L-24	Price Feed Limitations During Liquidations	Configuration	LOW	PENDING	p. 83
L-25	calculateReduceOnlyAdjustment Race Condition	Unexpected Behavior	LOW	PENDING	p. 84
I-01	safeAddOnelfGreater Unused Function	Best Practices	INFORMATIONAL	PENDING	p. 85
I-02	Incomplete Response On Withdraw	Best Practices	INFORMATIONAL	PENDING	p. 86
I-03	Certain Transactions Don't Wait For Mining	Warning	INFORMATIONAL	PENDING	p. 87
I-04	ERC-1271 Signatures Unsupported	Logical Error	INFORMATIONAL	PENDING	p. 88
I-05	handleWithdrawal Processes Negative Quantities	Informational	INFORMATIONAL	PENDING	p. 89
I-06	Inconsistent InitialMargin Value	Informational	INFORMATIONAL	PENDING	p. 90
I-07	GetPositionsBySymbol Response Not Populated	Informational	INFORMATIONAL	PENDING	p. 91
I-08	Skipped Transactions Can Undercount Totals	Informational	INFORMATIONAL	PENDING	p. 92
I-09	Inconsistent Rounding Between Deposits and Pnl	Informational	INFORMATIONAL	PENDING	p. 93
I-10	Missing Collateral Due To Unhandled Error	Informational	INFORMATIONAL	PENDING	p. 94
I-11	Bulk Order Cancellation May Lose Results	Informational	INFORMATIONAL	PENDING	p. 95
I-12	Margin Not Cleared On Error	Best Practices	INFORMATIONAL	PENDING	p. 96
I-13	Legacy Code Not Cleaned Up	Informational	INFORMATIONAL	PENDING	p. 97
I-14	Unused Parameters	Informational	INFORMATIONAL	PENDING	p. 98
I-15	SLP Collateral Logs Zero	Informational	INFORMATIONAL	PENDING	p. 99
I-16	created_at Overwritten On Every Event	Informational	INFORMATIONAL	PENDING	p. 100
I-17	Unreachable Code Risk Of Stuck Withdraws	Informational	INFORMATIONAL	PENDING	p. 101

Findings continued.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
I-18	Unrealized PnL Considered Withdrawable	Logical Error	INFORMATIONAL	PENDING	p. 103
I-19	Authentication Can Be Disabled	Warning	INFORMATIONAL	PENDING	p. 104
I-20	Missing End-to-End On-Chain Integration Tests	Warning	INFORMATIONAL	PENDING	p. 105

HIGH **H-01**

PENDING

Watchers Unable To Cast Votes On Set Contract

CATEGORY	DoS
LOCATION	https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/relayer/internal/core/auto_approve_watcher.go#L233
REVIEW	Main Review

— DESCRIPTION

The AutoApproveWatcher calls `getAllWithdrawalRequests(offset, limit)` on the SynthetixDepositContract to fetch pending withdrawals for automatic approval.

However, this function was removed from the SynthetixDepositContract and moved to a separate Lens contract. The relayer only has the SynthetixDepositContract address configured (`contract_address`), with no Lens contract address available. When `getAllWithdrawalRequests` is called on the current contract deployment, the call reverts. This causes `fetchRequestedIds` to return an error, and `processRequested` fails. Ultimately, all automatic watcher approvals will fail and withdrawals may remain in Requested status until expiry — locking user funds.

— RECOMMENDATION

Add the Lens contract address to the relayer configuration and update AutoApproveWatcher to query the Lens contract for withdrawal data.

— RESOLUTION

Status: Pending.

<https://github.com/Synthetixio/v4-offchain/pull/801>

HIGH **H-02**

PENDING

Race Condition In CreateActor()

CATEGORY Frontrunning

LOCATION manager.go

REVIEW Main Review

— DESCRIPTION

`CreateActor()` performs a `Load`, builds a new actor, and then `Stores` without any synchronization. Concurrent invocations (`handleDeposit()` handler and `handleSubaccountCreate()` handler both run in parallel JetStream/NATS goroutines) can each instantiate and start their own actor for the same subaccount before either `Store` executes.

Any initialization that was handled by the losing actor (e.g., deposits applied to the SubAccount before calling `CreateActor()`) is effectively lost which can lead to deposits not being reflected in the live actor.

— RECOMMENDATION

Use an atomic "create-if-absent" guard (mutex or `sync.Map.LoadOrStore`) around `CreateActor()` so only one actor per subaccount can be created.

— RESOLUTION

Status: Pending.

HIGH **H-03**

PENDING

Withdrawal Disputes Not Implemented

CATEGORY	Logical Error
LOCATION	Global
REVIEW	Main Review

DESCRIPTION

The on-chain withdrawal flow supports a dispute mechanism where the Relayer or Watcher can call `disputeWithdrawals()` to flag suspicious withdrawals for Guardian review, and the Guardian can subsequently call `resolveDisputedWithdrawal()` to approve or deny them. However, the off-chain system does not implement this functionality:

1. `disputeWithdrawals()` and `resolveDisputedWithdrawal` - Contract bindings exist but are never called.
2. When a `WithdrawalStatusDisputed` event is observed, the code only logs a warning and takes no action:
`1.logger.Warn("[PROCESSOR] Withdrawal DISPUTED on-chain - no action taken")`

Ultimately, there is no mechanism to dispute suspicious withdrawals and trigger Guardian review, violating the intended withdrawal flow and spec.

RECOMMENDATION

Add logic to support `disputeWithdrawals`, and `resolveDisputedWithdrawal`, and the `Disputed` status.

RESOLUTION

Status: Pending.

<https://github.com/Synthetixio/v4-offchain/pull/814>

HIGH **H-04**

PENDING

Withdrawal Pause Stalls Entire Event Listener

CATEGORY Logical Error
LOCATION listener.go
REVIEW Main Review

DESCRIPTION

When `GlobalWithdrawalsPaused` is emitted, the listener continues to submit disburse transactions for `Validated` withdrawals without checking the pause state. Those on-chain calls revert while paused. Because the listener returns an error on disbursal failure and cannot skip or defer the failing log, it gets stuck retrying the same `Validated` log and never advances the block checkpoint. While stuck, subsequent `AssetDeposited` events are not processed off-chain, so user deposits stop being credited and any other events in later blocks are missed. Although `withdrawalsGloballyPaused = true` should only pause withdrawals, it effectively disabled deposits due to deposits not being credited off-chain, and can lead to failed margin top-ups and liquidation.

RECOMMENDATION

Before submitting disbursals, verify whether withdrawals are paused and queue the disburse to be handled upon unpauses.

RESOLUTION

Status: Pending.

<https://github.com/Synthetixio/v4-deposit-contract/pull/34>

HIGH **H-05**

PENDING

Token Specific Pauses Entirely Ignored

CATEGORY DoS
LOCATION listener.go
REVIEW Main Review

— DESCRIPTION

The listener handles GlobalWithdrawalsPaused events but does not listen for or track TokenSpecificWithdrawalsPaused events. When a guardian pauses withdrawals for a specific token with `pauseWithdrawalsFor()`, the listener remains unaware.

It continues to call SubmitBatchDisbursal for Validated withdrawals containing that token, which reverts on-chain with TokenWithdrawalsPaused. Since the listener returns an error on disbursal failure and cannot skip the failing log, it gets stuck and cannot advance the block checkpoint. This stalls all event processing including deposits for unrelated tokens.

— RECOMMENDATION

Add handlers for TokenSpecificWithdrawalsPaused and TokenSpecificWithdrawalsUnpaused events in `routeEvent` and skip disbursals when paused.

— RESOLUTION

Status: Pending.

<https://github.com/Synthetixio/v4-offchain/pull/905>

HIGH **H-06**

PENDING

Lost Withdrawal Requests With Unconditional ACK

CATEGORY	Logical Error
LOCATION	withdrawal_subscriber.go
REVIEW	Main Review

— DESCRIPTION

In `handleWithdrawalRequest`, when `SubmitWithdrawalRequest` returns an error, the handler returns nil which causes the NATS message to be ACKed and permanently removed from the queue. This can simply happen in a period of RPC connectivity issues, in which case no on-chain transaction is made, no `PendingTransaction` is saved, yet the withdrawal request is ACKed and discarded.

User withdrawal requests can be permanently lost due to transient infrastructure failures. A brief RPC outage will silently drop the withdrawal with no retry path, locking user funds.

Furthermore, `PendingWithdraw` was increased when the user requested a withdrawal but never cleared, hence the withdraw quantity is stuck in `PendingWithdraw` and the user may not re-request the same amount through the API due to the user's reduced available balance.

Note that other consumers are also exposed to the unconditional ACK pattern, such as in `cow_order_subscriber` the `handleCowOrder` function and `handleMatchingMsg` in `subaccount/manager.go`.

— RECOMMENDATION

Return an error if the tx was not broadcast so the request will be retried.

— RESOLUTION

Status: Pending.

<https://github.com/Synthetixio/v4-offchain/pull/836>

HIGH **H-07**

PENDING

Closed Channel Will Panic Actor

CATEGORY	DoS
LOCATION	https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/handlers/withdrawal_handler.go#L107
REVIEW	Main Review

DESCRIPTION

The withdrawal handler creates a buffered response channel and closes it if the handler times out after 5 seconds in `case <-time.After(5 * time.Second)`. However, the Actor's

`_sendWithdrawalResponseIfChannelOpen` function only checks for nil, not closed.

If the Actor processes the withdrawal after the handler timeout, sending on the closed channel causes a panic, crashing the Actor goroutine. Once the Actor is crashed, it is not automatically restarted and will block all subsequent operations on that subaccount affecting trading, withdrawals, etc.

RECOMMENDATION

Don't send a response if the channel is closed or don't close the channel in the handler.

RESOLUTION

Status: Pending.

HIGH **H-08**

PENDING

No Rate Limiting on Withdrawal Requests

CATEGORY	DoS
LOCATION	withdraw_collateral.go
REVIEW	Main Review

— DESCRIPTION

Unlike order placement which enforces rate limits with `ctx.OrderRateLimiter.CheckOrderLimit()`, withdrawal requests have no rate limiting. An attacker can spam withdrawal requests to exhaust the relayer's NATS consumer capacity and block legitimate withdrawals. The relayer's withdrawal consumer is configured with `MaxAckPending: 100` within the `withdrawal_subscriber.go` file. Each message remains pending until `WaitMined()` completes (up to 5 minutes) or the transaction fails. When an attacker submits 100 withdrawal requests:

- NATS delivers all 100 messages
- All 100 AckPending slots are consumed immediately
- NATS stops delivering new messages until slots free up
- Legitimate withdrawal #101 is queued indefinitely

Even though the on-chain contract only allows one active withdrawal per user, the off-chain queue is shared across all users so the off-chain DoS is possible. An attacker with multiple accounts (or one account repeatedly failing at the contract level) can take total control over the queue and block others from withdrawing their assets indefinitely.

— RECOMMENDATION

Add withdrawal rate limiting at the API layer.

— RESOLUTION

Status: Pending.

<https://github.com/Synthetixio/v4-offchain/pull/848>

HIGH **H-09**

PENDING

Missing Timeouts Enable Slowloris DoS

CATEGORY	DoS
LOCATION	https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/api/cmd/server/main.go#L273
REVIEW	Main Review

DESCRIPTION

Any unauthenticated client can open a TCP connection to the public REST API (e.g. /v1/trade or /v1/info) and slowly drip HTTP headers/body bytes without ever finishing the request. In `services/api/internal/config/config.go` the configuration already exposes `ReadTimeout` and `WriteTimeout`, but in `services/api/cmd/server/main.go` the Echo server is created with `echo.New()` and started with `e.Start(cfg.ServerAddress)` without ever assigning those timeout values to the underlying `http.Server`. The Go `net/http` server defaults both timeouts to zero, meaning the connection is kept open indefinitely. An attacker can therefore hold thousands of concurrent goroutines by keeping connections half-open, exhausting file descriptors and worker capacity and preventing legitimate traders from placing orders.

RECOMMENDATION

Before calling `e.Start`, set the underlying server timeouts, e.g. `e.Server.ReadTimeout = cfg.ReadTimeout` and `e.Server.WriteTimeout = cfg.WriteTimeout`, and optionally cap `ReadHeaderTimeout` and `IdleTimeout`. This enforces per-connection limits and prevents Slowloris-style resource exhaustion.

RESOLUTION

Status: Pending.

HIGH **H-10**

PENDING

WebSocket Lack Per-IP Quotas

CATEGORY	DoS
LOCATION	https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/websocket/internal/handlers/websocket_manager_subscriptions.go#L147
REVIEW	Main Review

DESCRIPTION

The WebSocket service lacks checks that would enforce the configured `maxConnectionsPerIP/maxSubscriptionsPerIP` thresholds, and the health monitor is only consulted by the `/health/ready` endpoint rather than the upgrade handler. An attacker can therefore open thousands of `/ws/info` or `/ws/trade` sockets and subscribe to all broadcasters without being rejected, consuming goroutines, Redis/JETStream state, and NATS subscriptions until the service stops accepting new traders. This attack will prevent legitimate trader connections to `/ws/trade` and order execution.

RECOMMENDATION

1. Enable the Redis-backed per-IP connection limit inside `handleWebSocketConnection` (return HTTP 429 when `limitExceeded` is true).
2. Enable the per-IP subscription cap inside `handleSubscribeRequest` (increment the counter atomically and reject when the limit is hit).
3. Gate the WebSocket upgrade path on `healthMonitor.ShouldAcceptNewConnections()/MaxTotalConnections` (return 503 when the health monitor rejects), so the service can shed load before it runs out of goroutines or NATS subscriptions.

RESOLUTION

Status: Pending.

HIGH **H-11**

PENDING

Unbounded Request Bodies Lead to OOM

CATEGORY	DoS
LOCATION	Global
REVIEW	Main Review

DESCRIPTION

Attacker-controlled HTTP POST data reaches every exposed service without bounds: `compose.yml` publishes `rpcproxy-service` on 0.0.0.0:3001, so arbitrary clients can hit `Proxy.ServeHTTP` in `services/rpcproxy/internal/services/rpcproxy/proxy.go`, which immediately calls `io.ReadAll(r.Body)` without wrapping `http.MaxBytesReader`.

Likewise, `services/api/cmd/server/main.go` mounts public POST `/v1/info` and POST `/v1/trade`, and their handlers (`services/api/internal/handlers/info_handler.go` and `services/api/internal/handlers/trade_handler.go`) read the entire `c.Request().Body` with `io.ReadAll` before validation or auth.

Because none of these paths enforce content-length limits or streaming, a single oversized POST to any endpoint forces the Go process to allocate memory proportional to the payload, exhausting RAM and crashing the rpc proxy as well as both trading/info APIs. Ultimately users will not be able to place/cancel orders, adjust margin, withdraw, etc.

RECOMMENDATION

Apply strict per-endpoint request-body limits before any `io.ReadAll` call. Wrap each handler's Body with `http.MaxBytesReader` (configurable sane defaults such as 10 MiB for `rpcproxy`, 5 MiB for `/v1/info`, 1 MiB for `/v1/trade`), return HTTP 413 when exceeded, and optionally enforce matching limits at load balancers.

RESOLUTION

Status: Pending.

HIGH **H-12**

PENDING

Status Update Can Overwrite Balances

CATEGORY Frontrunning
LOCATION subaccount_repository.go
REVIEW Main Review

DESCRIPTION

`UpdateSubaccountStatus()` is supposed to flip a subaccount between ACTIVE/FROZEN, but it loads the entire `sub_accounts` row and then calls `Save(&subAccount)` after only changing the `Status` and `UpdatedAt` fields:

```
```plain text
```

```
subAccount.Status = status
```

```
subAccount.UpdatedAt = snx_lib_utils_time.Now()
```

```
if err := r.db.WithContext(ctx).Save(&subAccount).Error; err != nil {
```

```
return nil, snx_lib_grpc.SubaccountStatus_ACCOUNT_STATUS_UNSPECIFIED, fmt.Errorf("failed to update
subaccount status: %w", err)
```

```
}
```

```
``
```

A concurrency issue can occur because the admin gRPC endpoint `UpdateSubaccountStatus()` runs in its own goroutine, and other service is always mutating the same `sub_accounts`` rows during deposits, transfers, withdrawals, etc.

So if a deposit/withdrawal updates balances just before an admin freeze/unfreeze runs, the status call replays its stale snapshot and undoes those balance changes.

## RECOMMENDATION

Use a field-scoped update to update only the single status field rather than the whole row.

## RESOLUTION

Status: Pending.

HIGH **H-13**

PENDING

# WS Message Replay Due To Lack of Freshness Check

CATEGORY	Logical Error
LOCATION	<a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/websocket/internal/handlers/websocket_manager.go#L515">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/websocket/internal/handlers/websocket_manager.go#L515</a>
REVIEW	Main Review

## — DESCRIPTION

WebSocket trading connections authenticate by submitting an EIP-712 typed data message to `/trade` and the service calls `AccountAuthenticator.ValidateAccountAuth` with `SupportExpiration` explicitly disabled. That function never validates that the provided timestamp is fresh -- it only reserves that timestamp in Redis for 10 minutes as a nonce.

Once the Redis key expires, the same signed payload (and the same signature) may be replayed to reauthenticate the victim's sub-account without requiring a new signature from the wallet.

An attacker who captures a single auth message/signature pair (e.g., through a compromised client or a MITM where TLS is not enforced) can wait 10 minutes, replay the same payload, and bypass the victim's private key to re-open the trading socket, read account streams, or issue trades on their behalf.

Ultimately, replay of a captured auth request after the nonce key expires allows remote attackers to impersonate the victim's wallet address and sub-account, subscribe to sensitive account updates, and send authenticated trade actions without providing a new signature. This gives the attacker full access to the victim's trading WebSocket session.

## — RECOMMENDATION

Call `ValidateTimestamp` (or otherwise enforce `expiresAfter`) before reserving the nonce and store the nonce indefinitely so seen timestamps can never be reused, eliminating the replay window even after the Redis TTL expires.

## — RESOLUTION

Status: Pending.

HIGH **H-14**

PENDING

# Shared Key For Relayer, Teller and Watcher

CATEGORY	Logical Error
LOCATION	Global
REVIEW	Main Review

— DESCRIPTION

## H-14 Shared Key For Relayer, Teller and Watcher

CONTINUED

The off-chain relayer service is currently designed to operate with a single private key for all on-chain operations including:

1. **RELAYER\_ROLE**: Submitting withdrawal requests with `requestWithdrawal()`
2. **WATCHER\_ROLE**: Casting approval votes with `castWatcherVotes()`
3. **TELLER\_ROLE**: Disbursing validated withdrawals with `disburseWithdrawals()`

The `AutoApproveWatcher` component in the relayer service automatically casts watcher votes for pending withdrawals using the same `TransactionManager` and private key that performs relayer and teller functions. This can be observed in `services/relayer/cmd/server/main.go` where a single `txManager` is passed to both the `AutoApproveWatcher` and other components, and in `services/relayer/internal/core/auto_approve_watcher.go` where votes are cast using `w.txManager.CastWatcherVote()`.

The on-chain contract, however, was primarily designed for a multi-party model with N-of-M watcher quorum. When a withdrawal receives enough watcher votes to meet the `watcherQuorum` threshold, it transitions from `Requested` to `Validated` status. The deployment configuration in `.env.example` expects 3 independent watchers with a 2-of-3 quorum, with separate addresses for `WATCHER_1`, `WATCHER_2`, and `WATCHER_3`, and `WATCHER_QUORUM=2`.

This architectural mismatch creates a deployment dilemma:

If deployed with intended security (3 watchers, quorum of 2):

- The single relayer can only cast 1 vote per withdrawal
- Withdrawals never reach the required 2-vote quorum
- All withdrawals remain stuck in `Requested` status until they expire
- The system is non-functional for withdrawals

If deployed with single watcher (quorum of 1):

- Compromise of the relayer key grants complete control over all withdrawals
- The watcher quorum system provides limited additional security/health
- The multi-party security model is negated

If multiple relayer instances are deployed (attempting a workaround):

- Nonce collisions occur as all instances query `PendingNonceAt()` independently
- Duplicate transaction submissions occur for the same withdrawal
- Race conditions arise in NATS message consumption
- The system becomes unreliable and prone to failures

### — RECOMMENDATION

Separate watcher and teller from the relayer so they can run with separate, independent keys.

### — RESOLUTION

Status: Pending.

HIGH **H-15**

PENDING

## Trade Action Signing Omits Payload Fields

CATEGORY	Logical Error
LOCATION	<a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/api/internal/handlers/trade_handler.go#L609">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/api/internal/handlers/trade_handler.go#L609</a>
REVIEW	Main Review

### — DESCRIPTION

CreateTradeTypedData falls back to signing only subAccountId/action/nonce/expiresAfter for every requestAction except the few order APIs. TradeHandler ultimately uses that same typed data to validate signatures, so the EIP-712 verification never covers parameters such as withdraw amount, destination, leverage or delegated signer details. As soon as an attacker can intercept or modify a client request (MITM, compromised relay, etc.) they can rewrite these fields after the client has signed the request and still have the request succeed.

For example `withdrawCollateral` uses the user-supplied amount/destination directly when building the WithdrawCollateralRequest that is sent to the trading service, so a tampered request is processed with whatever values the attacker injected while the signature remains valid. Every other handler that does not have a custom typed data ( `transferCollateral` , `updateLeverage` , add/remove delegated signer, etc.) is affected because their payloads never entered the signed message space.

An attacker who can intercept or modify a trade API request (e.g., an HTTPS MITM, proxy, or compromised client) can increase withdrawal amounts, redirect collateral withdrawals to their address, or change leverage/transfer targets without touching the signature or nonce, effectively stealing funds or corrupting subaccount state while still passing signature verification.

### — RECOMMENDATION

Create an EIP-712 schema for each action and include every user-controlled parameter in the typed data so the signature is bound to the actual payload (amount, destination, symbol, permissions, etc.). For now treat any action without a custom schema as invalid (fail fast) and add typed data builders before allowing the handler to run. This ensures any tampering of the JSON body invalidates the signature and stops unauthorized parameter changes.

### — RESOLUTION

Status: Pending.

HIGH **H-16**

PENDING

## Missing NATS Handlers for Internal Transfers

CATEGORY Logical Error  
LOCATION `account_events.go`  
REVIEW Main Review

### — DESCRIPTION

The subaccount service publishes transfer requests to `subaccount.event.transfer.request`, waits for a response with 30 second timeout, and subscribes to `trading.event.transfer.response` to receive responses.

No consumer exists for `SubaccountEventTransferRequest` and the trading service does not publish `TradingEventTransferResponse`. Every `TransferBetweenAccounts` call will timeout and fail, hence the transfer feature between subaccounts is not functional.

### — RECOMMENDATION

Implement the missing transfer handler in trading service. Subscribe to `subaccount.event.transfer.request`. Validate and execute the transfer and then publish `trading.event.transfer.response`.

### — RESOLUTION

Status: Pending.

HIGH **H-17**

PENDING

# SLP Handoff Ignores ImmediateSLPTakeover Breach

CATEGORY Validation  
 LOCATION liquidation\_engine.go  
 REVIEW Main Review

## DESCRIPTION

Liquidation runs in two phases:

Phase 1 – Position liquidation ( `processMMRBreachState()` ): cancel open orders, liquidate positions. If `checkMarginBreach()` ever reports `ImmediateSLPTakeoverBreachType` here, the engine immediately sets `state.Stage = LiquidationStageSLPTakeover` and skips to phase 2 (correct behavior).

Phase 2 – Debt handoff ( `handleAllPositionsLiquidated()` ): once no positions remain, the engine re-checks margins to decide weather any debt must be transferred to SLP. The code only handles `breach.Type == MMRBreachType` in this step:

```
```plain text
breach := le.checkMarginBreach()

if err := le.handleAutoExchange(context.Background()); err != nil {
    ...
}

if breach.Type == MMRBreachType {
    // escalate to SLP takeover
    ...
    return nil
}

le.logger.Debug("MMR breach resolved through liquidation", ...)
le.state.Reset()
return nil
```
```

If phase 1 started as a normal `MMRBreachType` but, after liquidation, the refreshed check now returns `ImmediateSLPTakeoverBreachType`, this branch is missed and the engine resets even though  $AAV \leq MMR/2$  still holds.

This scenario is realistic when liquidation realizes larger losses than expected: e.g., slippage, taker fees, funding payments, or negative USDT debt remaining after

## H-17 SLP Handoff Ignores ImmediateSLPTakeover Breach

CONTINUED

`handleAutoExchange()`. Those effects can push the account from "AAV \< MMR" to "AAV \< MMR/2" once positions are gone.

This creates a window where an insolvent account is allowed to submit new orders until the next

`EnforceAccountHealth()`

### — RECOMMENDATION

In `handleAllPositionsLiquidated()`, treat `ImmediateSLPTakeoverBreachType` the same way as `MMRBreachType` (i.e., escalate to SLP takeover) so that accounts below MMR/2 after liquidation don't get incorrectly reset.

### — RESOLUTION

Status: Pending.

MEDIUM **M-01**

PENDING

# Panic Due To Missing SLP Actor

CATEGORY DoS  
LOCATION manager.go  
REVIEW Main Review

## DESCRIPTION

`GetSLPActorForMarket()` function attempts to find a SLP mapping for market, if no mapping exists. After searching for the `defaultSLPActor` the code immediately logs the result

```
``plain text
m.logger.Debug("No SLP mapping for market", "market", marketSymbol, "defaultSLPActor",
defaultSLPActor.subAccount.ID)
`[
```

However if the loop never finds an SLP actor, `defaultSLPActor` remains nil, and dereferencing `.subAccount.ID` panics, so a missing SLP account can crash the trading service.

Compare with `GetSLPActorForCollateral()`, which avoids this by checking for nil before logging

## RECOMMENDATION

Check `defaultSLPActor` is not nil before logging, as is done in `GetSLPActorForCollateral()`

## RESOLUTION

Status: Pending.

MEDIUM **M-02**

PENDING

# Deposit/Withdraw Responses Lacks Idempotency

|          |                                                                                                                                                                                                                                                                                                                                                           |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                                                                                                                                             |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/handlers/account_events.go#L96">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/handlers/account_events.go#L96</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                               |

## — DESCRIPTION

The withdrawal handler applies each WithdrawProcessedEvent to the actor. If the consumer is recreated with DeliverAll or messages are redelivered (e.g. on restart), the same event will be applied twice, double-adjusting PendingWithdraw/Quantity.

The same issue is present with deposits, such that the same DepositProcessedEvent could be applied again, inflating the user's collateral balance.

## — RECOMMENDATION

Track processed request IDs in the Actor and then check if a specific RequestID has already been processed before apply changes in the Actor.

## — RESOLUTION

Status: Pending.

MEDIUM **M-03**

PENDING

## Lock Held For 5 Minutes Blocks Other Tx's

|          |                                                                                                                                                                                                                                                                                                                                                               |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | DoS                                                                                                                                                                                                                                                                                                                                                           |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/relayer/internal/core/transaction_manager.go#L236">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/relayer/internal/core/transaction_manager.go#L236</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                                   |

### — DESCRIPTION

Function `SubmitWithdrawalRequest` holds `signerLock` for the entire duration including `WaitMined()` which has a 5 minute timeout. Because `CastWatcherVote`, `SubmitBatchDisbursal`, and subsequent withdrawal submissions all take the same lock, one slow or congested withdrawal will block and delay every other transaction type for the duration.

### — RECOMMENDATION

Consider releasing the lock after `SendTransaction`.

### — RESOLUTION

Status: Pending.

MEDIUM **M-04**

PENDING

# Subaccount ID Overflow in Relayer

|          |                                                                                                                                                                                                                                                                                                                                                     |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                                                                                                                                       |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/relayer/internal/blockchain/listener.go#L394">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/relayer/internal/blockchain/listener.go#L394</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                         |

## DESCRIPTION

The Relayer listener converts the on-chain `subAccountId`, a `uint256` in the `DepositEntry` struct and `AssetDeposited` event, into a Go `uint64`: `SubaccountID: event.SubAccountId.Uint64()`. Subaccount ID values greater than  $2^{64}-1$  will silently truncate. While the deposit still credits the same beneficiary, different large on-chain sub-account IDs collapse to the same `uint64` within that user, causing misattribution across a user's own sub-accounts.

## RECOMMENDATION

Change the type of `DepositEntry.subAccountId` to `uint64`.

## RESOLUTION

Status: Pending.

MEDIUM **M-05**

PENDING

# Reduce-Only Orders Inflate Margin Calculations

|          |                                                                                                                                                                                                                                                                                                                                                                       |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                                                                                                                                                         |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/margin_calculator.go#L189">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/margin_calculator.go#L189</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                                           |

## — DESCRIPTION

Function `calculateOrderValuesByMarket` sums every open order for a symbol, including reduce-only orders, when computing buy/sell notional for `CalculateOrderMarginRequirement`. Reduce-only orders are not meant to reserve additional margin as they only reduce exposure and they are already excluded from order RMR calculations.

Including them in `calculateOrderValuesByMarket` overstates worst-case notional, causing normal orders to appear riskier, leading to unnecessary rejections or extra order cancellations.

## — RECOMMENDATION

Exclude reduce-only orders in `calculateOrderValuesByMarket`.

## — RESOLUTION

Status: Pending.

MEDIUM **M-06**

PENDING

# Off-chain Withdrawal Minimum Not Validated

CATEGORY Gas Griefing  
LOCATION `withdraw_collateral.go`  
REVIEW Main Review

## — DESCRIPTION

The off-chain system only validates that withdrawal amounts are positive (`amount > 0`), but does not enforce the on-chain `withdrawalMinimum` configured per collateral in the contract's `CollateralConfig`. This allows dust withdrawals to traverse the entire pipeline before reverting on-chain, allowing for the following grieving vector:

1. User submits withdrawal for dust amount
2. Request passes all off-chain validation (API, Actor, Subaccount Service)
3. Funds are moved to `PendingWithdraw` in Actor state
4. Relayer submits on-chain transaction
5. Contract reverts due to `withdrawalMinimum` check
6. Protocol pays gas for the reverted transaction
7. User repeats with minimal cost and continues to drain relayer funds on ETH

## — RECOMMENDATION

Enforce the on-chain minimum at the API layer to reject invalid withdrawals.

## — RESOLUTION

Status: Pending.

<https://github.com/Synthetixio/v4-offchain/pull/852>

MEDIUM **M-07**

PENDING

# Gap In ProcessTransfer Relationship Validation

CATEGORY Access Control  
LOCATION services/subaccount/service.go  
REVIEW Main Review

## DESCRIPTION

`ProcessTransfer()` is designed to enforce master to subaccount relationships (either master to sub or sub to its master).

But when both accounts are masters ( `MasterID == 0` ), no relationship check runs, that means any user who knows another master account's ID can push collateral directly to it, bypassing wallet ownership and intended account-family boundaries, creating an authorization gap.

In a regulated derivatives context, that undermines typical AML/KYC controls: it becomes easier to layer funds by bouncing them across masters, potentially obscuring the source within the protocol.

## RECOMMENDATION

Add an explicit guard for master to master transfers: either disallow them outright or require both masters to share the same `WalletAddress` (or other ownership identifier).

## RESOLUTION

Status: Pending.

MEDIUM **M-08**

PENDING

# Actor Crash Causes Balance Desync

|          |                                                                                                                                                                                                                                                                                                                                                                     |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                                                                                                                                                       |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/actor_withdrawal.go#L179">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/actor_withdrawal.go#L179</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                                         |

## DESCRIPTION

When a user initiates a withdrawal, `handleWithdrawal` deducts from `Quantity` and adds to `PendingWithdraw` in memory. If the actor crashes after the in-memory update (before the NATS message is delivered and persisted in `publishPnlSettledUpdate`), the actor will restart with stale state (`PendingWithdraw = 0`). Meanwhile, the withdrawal request continues through the pipeline and succeeds on-chain.

When `handleWithdrawalResponse` receives the success confirmation, it checks if `PendingWithdraw >= msg.Quantity` and sees `PendingWithdraw (0) < msg.Quantity (Non-Zero)`, it logs a warning and sets `PendingWithdraw = 0` without making any adjustment to `Quantity`.

Consequently, the user receives funds on-chain, but the actor's internal balance remains inflated by the withdrawn amount. The user can withdraw the phantom collateral again or use it for trading.

## RECOMMENDATION

Persist before sending withdrawal request so proper collateral recovery can take place.

## RESOLUTION

Status: Pending.

MEDIUM **M-09**

PENDING

# Collateral Jobs Never Processed

CATEGORY Logical Error  
LOCATION listener.go  
REVIEW Main Review

## — DESCRIPTION

The `handleCollateralChange` function enqueues collateral update jobs into the database through `AddCollateralJob`. However, there is no worker in the codebase that processes these `CollateralJobs`. As a result, collateral add/remove events emitted on-chain are never reflected in the off-chain state. This can lead to inconsistencies where tokens remain usable off-chain after being removed on-chain, or newly added collateral tokens are never recognized by the off-chain services.

## — RECOMMENDATION

Process the jobs added to the DB.

## — RESOLUTION

Status: Pending.

MEDIUM **M-10**

PENDING

# One Failed Publish Bricks TP/SL

CATEGORY Error  
LOCATION actor\_tick.go  
REVIEW Main Review

## DESCRIPTION

When a stop-loss or take-profit fires, the actor converts it into a normal market/limit order before sending it to the matching engine.

```
``plain text
if err := a.convertConditionalToRegularOrder(order); err != nil {
a.logger.Debug("Invalid order type", "orderId", order.ID, "orderType", order.Type, "error",
err)
return false
}
// ...
if err := a.publishOrder(ctx, requestID, chunkOrder); err != nil {
a.logger.Error("Failed to send triggered conditional order", "orderId", order.ID, "quantity",
firstChunkQuantity, "error", err)
return false
}
`
If that send fails (network hiccup, NATS publish error, etc.), the converted order is left in
memory as a normal order instead of a conditional one.
On the next price tick, the system looks at the same order, sees it is the wrong order type,
and will error out.
```

```
``plain text
func (a *Actor) convertConditionalToRegularOrder(order *types.Order) error {
switch order.Type {
case types.OrderTypeStopMarket, types.OrderTypeTakeProfitMarket:
order.Type = types.OrderTypeMarket
case types.OrderTypeStopLimit, types.OrderTypeTakeProfitLimit:
order.Type = types.OrderTypeLimit
default:
return fmt.Errorf("invalid conditional order type: %v", order.Type)
}
return nil
}
```

**M-10** One Failed Publish Bricks TP/SL

CONTINUED

```
}
...
```

The user still sees an active TP/SL, but it will not trigger properly until they delete and re-create it.  
The impact is that a single error can brick a TP/SL

**— RECOMMENDATION**

Do not mutate the conditional order unless the publish succeeds. Either work on a copy, or save the original type and restore it if publish fails.

**— RESOLUTION**

Status: Pending.

MEDIUM **M-11**

PENDING

# Rate Limiter Bypass With Multi-Instance

|          |                                                                                                                                                                                                                                                                                                                 |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | DoS                                                                                                                                                                                                                                                                                                             |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/lib/utils/order_rate_limiter.go#L37">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/lib/utils/order_rate_limiter.go#L37</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                     |

## — DESCRIPTION

The OrderRateLimiter is implemented using in-memory storage (sync.Map) rather than distributed storage, despite accepting a Redis client parameter that is discarded ( `_ any` ). Each API or WebSocket service instance maintains its own independent rate limit counters. In production deployments with multiple instances behind a load balancer, the effective rate limit scales linearly with instance count:

- 1 instance -> 100 orders/sec per subaccount
- 3 instances -> 300 orders/sec per subaccount
- 10 instances -> 1,000 orders/sec per subaccount

An attacker doesn't need to actively exploit this - standard load balancer distribution automatically spreads requests across instances, bypassing the intended rate limit. This could enable order spam, unfair trading advantages, or DoS on downstream systems with resource exhaustion.

## — RECOMMENDATION

Consider implementing distributed rate limiting using the Redis client that is already passed to NewOrderRateLimiter.

## — RESOLUTION

Status: Pending.

MEDIUM **M-12**

PENDING

# Matching Engine Responses Dropped

|          |                                                                                                                                                                                                                                                                                                                                               |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                                                                                                                                 |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/actor.go#L675">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/actor.go#L675</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                   |

## — DESCRIPTION

When the matching engine sends order fill, accept, or cancel responses, `ActorManager.handleMatchingMsg` delivers them through `actor.Send()`, which routes to `sendResponse()`. This function performs a non-blocking send into a 1,000-slot buffered channel.

If the actor's response inbox is full (e.g., actor blocked processing another handler, or flooded with responses), the message is dropped and the caller only logs the error. Consequently, users may have filled orders they never see reflected in their account and positions and balances diverge from the matching engine's state.

## — RECOMMENDATION

Instead of dropping on full, block with timeout or implement a retry mechanism with exponential backoff.

## — RESOLUTION

Status: Pending.

MEDIUM **M-13**

PENDING

# GetCandlesticks Unrestricted Range Enables DoS

|          |                                                                                                                                                                                                                                                                                                                                                           |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | DoS                                                                                                                                                                                                                                                                                                                                                       |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/marketdata/internal/service/marketdata.go#L1836">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/marketdata/internal/service/marketdata.go#L1836</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                               |

## — DESCRIPTION

The MarketData gRPC handler accepts user-supplied startTime, endTime, and limit, but it only caps the limit when the client requests more than 1,000 rows. There is no enforcement when limit is zero or negative, so defaults (zero) silently disable the LIMIT clause, and the full requested time range is forwarded to the service. Downstream the service also lacks any guardrails: it happily truncates the provided range to timeframe boundaries and pipes limit directly to the repository, and the repository only applies LIMIT when limit  $\geq 0$ . An attacker can therefore call GetCandlesticks with a narrow timeframe (e.g., 1m) but startTime months or years in the past while leaving limit unset; the code will scan & return every candle in that enormous range, loading entire tables into memory and saturating the database before replying.

Ultimately, an unauthenticated client can ask for minute-level candles spanning months/years with limit unset. Each request will load the full historical dataset for that range into memory and stream it back, exhausting the MarketData service's RAM/CPU and the PostgreSQL connection pool, ultimately taking the service (and any callers that depend on it) offline.

## — RECOMMENDATION

Enforce sensible caps and pagination. Treat limit values  $\leq 0$  with default value (e.g 100 or 1000) and reject ranges whose duration exceeds a configured maximum per timeframe. Alternatively, paginate via cursors/streaming and reject requests that would read more than a few thousand rows. Ensure the code always applies a LIMIT, even when the client omits it.

## — RESOLUTION

Status: Pending.

MEDIUM **M-14**

PENDING

# Pending Reduce-Only Orders Not Adjusted

|          |                                                                                                                                                                                                                                                                                                                                                   |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                                                                                                                                     |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/orders.go#L3181">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/orders.go#L3181</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                       |

## DESCRIPTION

The `calculateReduceOnlyAdjustments` function correctly counts pending reduce-only orders (in `pendingOps`) toward `totalReduceOnlySize` to detect when combined reduce-only exposure exceeds position size. However, it only selects adjustment targets from `OpenOrders`, not from pending operations. If multiple reduce-only orders are submitted rapidly before the first is accepted by the matching engine, the total reduce size can exceed the existing position, violating the intended behavior detailed with the comment in `calculateReduceOnlyAdjustments`: "This prevents race conditions where multiple reduce-only orders are placed rapidly"

Consider the scenario where a user has a 10BTC position:

- Order A: 10 BTC reduce-only sell is submitted; it sits in `pendingOps` (not yet in `OpenOrders`).
- Before A is accepted, Order B: another 10 BTC reduce-only sell is submitted.
- `calculateReduceOnlyAdjustments` sums size as 10 (pending A) + 10 (new B) = 20 BTC vs 10 BTC position -  $\searrow$  > 10 BTC excess.
- But the adjustment candidates (`reduceOnlyOrders`) are built only from `OpenOrders`; pending orders aren't added there, so it returns no adjustments.
- Order B is allowed through; later both A and B end up in `OpenOrders`, totaling 20 BTC reduce-only against a 10 BTC position.
- If A fills first and fully closes the position, B can still fill; with no position left, that fill is treated as opening a new short (reduce-only reserves zero margin), so the user unintentionally flips into new exposure.

## RECOMMENDATION

When iterating `pendingOps` in `calculateReduceOnlyAdjustments`, append eligible pending reduce-only orders to `reduceOnlyOrders` (not just to `totalReduceOnlySize`) so they can be adjusted when excess exists.

**M-14** Pending Reduce-Only Orders Not Adjusted

CONTINUED

**— RESOLUTION**

Status: Pending.

MEDIUM **M-15**

PENDING

# Pending Orders Ignored In RMR Simulation

|          |                                                                                                                                                                                                                                                                                                                                                                       |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                                                                                                                                                         |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/margin_calculator.go#L437">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/margin_calculator.go#L437</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                                           |

## — DESCRIPTION

Function `calculateSimulatedRMR` within `ValidatePreTradeSafety` only considers OpenOrders for RMR calculation. Pending orders in `pendingOps` are not considered which could lead to underestimating the total margin required and pushing the user to immediate safety margin breach.

## — RECOMMENDATION

Consider pending operation in pre-trade safety calculations. Additionally, extend `calculateOrderValuesByMarket()` to merge pending create operations.

## — RESOLUTION

Status: Pending.

MEDIUM **M-16**

PENDING

# Deposit/Withdrawal Acked Before Processing

|          |                                                                                                                                                                                                                                                                                                                                                             |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                                                                                                                                               |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/handlers/account_events.go#L182">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/handlers/account_events.go#L182</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                                 |

## DESCRIPTION

In `account_events.go`, both `handleDeposit` and `handleWithdrawResponse` ACK the JetStream message immediately after `actor.Send()`, which only enqueues the message. It does not wait for the actor to apply the change. If the actor crashes after the send and before processing, the message is lost (already ACKed) and the state change never occurs.

For deposits, on-chain funds can be received but never credited. For withdrawal responses, `PendingWithdraw` may never be decremented (success) or restored (failure), leaving balances stale.

## RECOMMENDATION

Do not ACK until the actor confirms processing.

## RESOLUTION

Status: Pending.

MEDIUM **M-17**

PENDING

# Teller Role Does Not Cancel Stale Withdrawals

CATEGORY Logical Error

LOCATION Global

REVIEW Main Review

## DESCRIPTION

The [documentation](/2230a80a459e8049a50adcf2fbf46a93?pvs=25#2370a80a459e806483a1ccb0227fbb6a) states that tellers "constantly scan withdrawalRequests that are stale and batch turn them into Rejected". However, the off-chain teller implementation does not invoke `cancelStaleWithdrawals()` on the contract.

The `TransactionManager` in the relayer service only implements `SubmitBatchDisbursal()` for calling `disburseWithdrawals()`. There is no corresponding function to call `cancelStaleWithdrawals()`, and no scheduled task or event listener triggers stale withdrawal cancellation on-chain.

Additionally, the documentation references a `Rejected` status that does not exist in the contract. The actual status enum is:

- None
- Requested
- Validated
- Disbursed
- Denied
- Disputed
- Cancelled
- Expired

## RECOMMENDATION

Consider implementing an off-chain task within the teller that periodically identifies withdrawal requests past the `withdrawalExpiryTimeout` and batch-calls `cancelStaleWithdrawals()` to transition them to `Expired` status to free the user's withdrawal slot, and/or allow an on-chain request to clear the prior requested withdrawal if expired.

Additionally, update the documentation to reflect the correct status enum values used in the contract.

**M-17** Teller Role Does Not Cancel Stale Withdrawals

CONTINUED

**— RESOLUTION**

Status: Pending.

MEDIUM **M-18**

PENDING

# Partial Liquidation Forces SLP Takeover

|          |                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                                                                                                                                                                                       |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/liquidation_engine_process_state.go#L429">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/liquidation_engine_process_state.go#L429</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                                                                         |

## DESCRIPTION

The partial liquidation portion uses `position.Quantity * 25%` and is sent to the matcher without rounding to the market's lot/quantity exponent. The send path truncates extra decimals, so small partials can become zero and be rejected, which immediately triggers SLP takeover.

Consider the following scenarios:

- 2-decimal lot (0.01): Position 0.03 BTC  $\rightarrow 25\% = 0.0075 \rightarrow$  truncates to 0.00  $\rightarrow$  IOC rejected  $\rightarrow$  SLP takeover
- Integer lot: Position 1 contract  $\rightarrow 25\% = 0.25 \rightarrow$  truncates to 0  $\rightarrow$  SLP takeover

## RECOMMENDATION

Before sending the liquidation order, round the partial size to the market's allowed quantity step (and respect `MinTradeAmount`), and enforce a minimum of one lot so the size never truncates to zero.

## RESOLUTION

Status: Pending.

MEDIUM **M-19**

PENDING

# Panic Due To Closed Channel

CATEGORY DoS

LOCATION `services/subaccount/service.go`, `services/subaccount/event_handlers.go`

REVIEW Main Review

## DESCRIPTION

`ProcessTransfer()` stores a channel in `s.transferResponses` and schedules a cleanup that removes the entry and closes the channel as soon as the wait (`time.After(constants.TransferRequestTimeout)`) finishes

```
``plain text
defer func() {
s.transferMutex.Lock()
delete(s.transferResponses, requestID)
close(responseChan)
s.transferMutex.Unlock()
}()
\`
```

If the wait exits due to the timeout, that cleanup runs immediately. Meanwhile the response listener still has the channel pointer and could try to deliver the actual reply after the channel has been closed.

```
``plain text
s.transferMutex.RLock()
responseChan, exists := s.transferResponses[event.RequestID]
s.transferMutex.RUnlock()
```

```
if exists {
responseChan <- &event // panics if channel was closed by timeout cleanup
...
```

If the handler hits a closed channel, Go panics with "send on closed channel," crashing the NATS subscriber goroutine.

## RECOMMENDATION

**M-19** Panic Due To Closed Channel

CONTINUED

Remove the line `close(responseChan)` from the deferred cleanup.

In this pattern the channel has only one receiver ( `ProcessTransfer()` ), the handler either finds the channel (and delivers the response) or doesn't (and logs that nobody is waiting), so there's no resource leak because the channel is small (buffer size 1) and short-lived (cleaned by garbage collector once nothing references it)

**— RESOLUTION**

Status: Pending.

MEDIUM **M-20**

PENDING

# Liquidation Attempt State Resets On Crash

|          |                                                                                                                                                                                                                                                                                                                                               |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                                                                                                                                 |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/actor.go#L374">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/actor.go#L374</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                   |

## — DESCRIPTION

All liquidation state (MMR breach counts/window timers, current action, partial-vs-full attempt) lives only in LiquidationStateDetails in memory. A process crash/restart clears it. A position that had already had a partial cut will be treated as a first attempt again after restart and only partially liquidated instead of proceeding to full liquidation. In-flight liquidations are also forgotten. A position that was on its 2nd liquidation attempt (should do 100%) would restart at attempt 1 (25% partial). This will further delay the liquidation and potentially exacerbate losses.

## — RECOMMENDATION

Make liquidation logic resumable and idempotent: derive the next step from stored state plus current position health; if prior attempt was partial and still under margin, proceed directly to full liquidation. Avoid re-applying partial cuts.

## — RESOLUTION

Status: Pending.

MEDIUM **M-21**

PENDING

# Assets Disappear When SLP Transfer Fails

CATEGORY Unexpected Behavior  
LOCATION liquidation\_engine.go  
REVIEW Main Review

## DESCRIPTION

If `handleSLPTakeover()` fails transfer of collateral to SLP, yet it still removes collateral, regardless of `transferErr`

```
``plain text
transferErr := le.actor.slpTransfer.TransferCollateralToSLP(...)
if transferErr != nil {
le.logger.Warn("Failed to transfer collateral to SLP", ...)
// still proceeds
}
le.publishCollateralDeletion(collateralSymbol)
collateralSymbolsToDelete = append(collateralSymbolsToDelete, collateralSymbol)
`
`plain text
func (le *LiquidationEngine) clearLiquidatedAccount(...) {
for symbol, position := range le.actor.subAccount.Positions {
le.publishPositionCloseEvent(symbol, position)
}
le.actor.subAccount.Positions = make(map[string]types.Position)
...
}
```

Which publishes close events and zeroes every asset. As a result, collaterals are deleted without being transferred, producing permanent loss when SLP transfer fails.

This scenario occurs when `slpActor.Send()` enqueues the transfer message onto the actor's inbox channel, and it can fail when that channel is full. The request inbox capacity is 2,000 entries (`ActorRequestInboxChannelSize``), so a failure scenario is when the SLP actor already has 2,000 pending messages it hasn't processed yet

## RECOMMENDATION

**M-21** Assets Disappear When SLP Transfer Fails

CONTINUED

Abort or retry the takeover when transfer fails, or track which assets actually transferred and only clear those.

**— RESOLUTION**

Status: Pending.

LOW **L-01**

PENDING

# Deposit Spec and Contract Inconsistencies

|          |                              |
|----------|------------------------------|
| CATEGORY | Documentation                |
| LOCATION | SynthetixDepositContract.sol |
| REVIEW   | Main Review                  |

## — DESCRIPTION

The official Deposit contract product specification at [<https://www.notion.so/OFFICIAL-Deposit-Contract-2230a80a459e8049a50adcf2fbf46a93>](<https://www.notion.so/OFFICIAL-Deposit-Contract-2230a80a459e8049a50adcf2fbf46a93>) has multiple small differences with the actual implementation of the contract.

- The spec says Owner/Guardian can pause; contract allows only GUARDIAN\_ROLE to pause deposits/withdrawals (global or per-token) and only OWNER\_ROLE to unpause
- The spec says Watchers can validate/dispute/reject in "Takes Submitted withdrawals and batch turns them into Validated, Disputed or Rejected" but in the contract Watchers can only vote or dispute.
- The spec throughout mentions the states Submitted/Approved/Rejected and no Expired e.g. "while status is Requested or Approved"; the contract uses Requested/Validated/Denied/Disputed/Cancelled/Expired.
- The spec says Teller turns stale requested into Rejected but contract marks stale as Expired, and expiry is enforced opportunistically in multiple handlers
- The spec suggests Teller can reject stale with "Constantly scans withdrawalRequests that are stale and batch turns them into Rejected", however the contract TELLER cannot deny requests -- only expire them with `cancelStaleWithdrawals`. The function `rejectWithdrawals` is reserved for the Relayer.
- The spec mentions user can cancel before validation but does not mention that contract also blocks cancel once status is Disputed.

## — RECOMMENDATION

Update the Deposit spec language to mimic the on-chain contract and reconcile any discrepancies.

## — RESOLUTION

Status: Pending.

LOW **L-02**

PENDING

# Symbol Not Validated At API Layer

|          |                                                                                                                                                                                                                                                                                                                                             |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Validation                                                                                                                                                                                                                                                                                                                                  |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/lib/api/handlers/trade/withdraw_collateral.go#L94">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/lib/api/handlers/trade/withdraw_collateral.go#L94</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                 |

## DESCRIPTION

The API handler passes the Symbol field directly to the Trading Service without validating it against supported collaterals. While the Trading Service correctly rejects non-USDT symbols before any state changes occur, the invalid request still traverses the full API -> gRPC -> Trading Service path unnecessarily.

## RECOMMENDATION

Consider adding symbol validation at the API layer to fail fast.

## RESOLUTION

Status: Pending.

LOW **L-03**

PENDING

# Auto Approve Watcher Votes On Already Voted IDs

|          |                                                                                                                                                                                                                                                                                                                                                                 |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Gas Griefing                                                                                                                                                                                                                                                                                                                                                    |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/relayer/internal/core/auto_approve_watcher.go#L227">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/relayer/internal/core/auto_approve_watcher.go#L227</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                                     |

## DESCRIPTION

The `AutoApproveWatcher` polls every 60 seconds and fetches all withdrawals in Requested status with `getAllWithdrawalRequests()`. For each withdrawal, it performs validation and submits a `castWatcherVotes` transaction. However, there is no tracking of which withdrawals this watcher has already voted on.

The on-chain `castWatcherVotes` function silently skips already-cast votes without reverting, but the relayer undergoes unnecessary gas expenditure on no-op transactions.

## RECOMMENDATION

Consider skipping already voted-on withdrawals.

## RESOLUTION

Status: Pending.

LOW **L-04**

PENDING

# Withdrawal Completion Events Lack NATS Dedupe

CATEGORY Warning  
LOCATION listener.go  
REVIEW Main Review

## — DESCRIPTION

All withdrawal status notifications are published through function `publishWithdrawalEvent` without a Nats-Msg-Id. In contrast, deposit events include this header for JetStream deduplication. If the listener replays a block or resends a WithdrawalStatusChanged log, downstream services can receive duplicate withdrawal status notifications.

## — RECOMMENDATION

Add a deterministic Nats-Msg-Id to withdrawal publishes.

## — RESOLUTION

Status: Pending.

LOW **L-05**

PENDING

## Stale Bindings Present

|          |                                            |
|----------|--------------------------------------------|
| CATEGORY | Warning                                    |
| LOCATION | <code>synthetix_deposit_contract.go</code> |
| REVIEW   | Main Review                                |

### — DESCRIPTION

File `synthetix_deposit_contract.go` has numerous stale bindings, including but not limited to:

- (1) GetAllWithdrawalRequests, GetWithdrawalRequestsByStatus, and GetWithdrawalRequestsByUser are still exposed but were moved to a Lens contract
- (2) The binding exposes an OracleStaleTimeout() getter yet the contract actually has maxOracleStaleTimeout()
- (3) The bindings omit slpVault(), setSLPVault, approveSLPVault, getSLPVaultAllowance, and their events SLPVaultSet/SLPVaultApproved and more.

### — RECOMMENDATION

Regenerate `synthetix_deposit_contract.go` with the latest bindings.

### — RESOLUTION

Status: Pending.

LOW **L-06**

PENDING

# Non-USDT Collateral Not Supported

|          |             |
|----------|-------------|
| CATEGORY | Warning     |
| LOCATION | Global      |
| REVIEW   | Main Review |

## — DESCRIPTION

Currently non-usdt collateral is not supported in margin calculations. This is okay for initial deployment, but there should be changes in the off-chain code if a non-USDT collateral was ever enabled on-chain.

## — RECOMMENDATION

Ensure initial deployment only allows USDT collateral and modify the off-chain code to support multi-collateral if other tokens such as sUSDe and cbBTC end up being supported as collateral.

## — RESOLUTION

Status: Pending.

LOW **L-07**

PENDING

# Zero Address Passes API

|          |                        |
|----------|------------------------|
| CATEGORY | Validation             |
| LOCATION | withdraw_collateral.go |
| REVIEW   | Main Review            |

## DESCRIPTION

The API layer validates that the destination address is a valid hex format using `common.IsHexAddress()`, but this check accepts `0x0000000000000000000000000000000000000000000000000000000000000000` as valid. The downstream services (trading gRPC handler, relayer) perform no additional validation. When a user submits a withdrawal to `address(0)`:

1. API validates with `IsHexAddress("0x0000...000")` -\> passes
2. Request flows through to relayer
3. Relayer calls `requestWithdrawal()` on contract
4. Contract reverts with `ZeroAddress()`

Currently the off-chain layer only supports one `WithdrawalEntry` per `requestWithdrawal` (only one user and one token to withdraw), but if multiples users in `requestWithdrawal` were ever supported, then an attacker can trivially DoS other user withdrawals by passing a zero address beneficiary in a withdrawal request.

## RECOMMENDATION

Add explicit zero address rejection at the API layer.

## RESOLUTION

Status: Pending.

LOW **L-08**

PENDING

# Withdrawals Are Approved Using Stale Margin Data

|          |                                                                                                                                                                                                                                                                                                                                                 |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                                                                                                                                   |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/actor.go#L1224">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/actor.go#L1224</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                     |

## DESCRIPTION

`handleWithdrawal` allows/denies withdrawals using `calculateWithdrawableAmount`, which pulls cached `position.IMR`. Under normal conditions, mark price updates flow through `BroadcastTickUpdate()` -> `actor.handleMarkPriceUpdate()` -> `marginCalc.UpdateAccountState()`, keeping cached IMR values current.

However, both withdrawal requests and mark price updates are delivered to the same actor inbox channel (capacity 2000). Under heavy system load when the inbox approaches capacity, mark price updates may be delayed or dropped while withdrawal requests are still processed. In this scenario, withdrawals are evaluated against stale, lower IMR values. Hence users can withdraw funds that should be locked as margin during periods of system load.

Note that this backpressure can also lead to other areas to operate under stale values.

`calculateSymbolRMR()` recomputes the worst-case IMR using the latest `market.MarkPrice`, but subtracts `position.IMR`. When prices move without a position update, `position.IMR` remains stale, so `worstCaseIMR - currentPositionIMR` mixes live and old data. Hence the difference understates or overstates the true RMR depending on how prices have moved. Contrast this with `CalculateCumulativeIMR()`, which uses the correct value of latest market prices

## RECOMMENDATION

Recompute IMR/RMR from current mark prices to ensure liveness.

## RESOLUTION

Status: Pending.

LOW **L-09**

PENDING

# Silent Failures Return Zero Margin

|          |                                                                                                                                                                                                                                                                                                                                                                       |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Warning                                                                                                                                                                                                                                                                                                                                                               |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/margin_calculator.go#L727">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/margin_calculator.go#L727</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                                           |

## DESCRIPTION

In `calculateSimulatedRMR`, any error during margin computation (e.g. missing/invalid market data or no applicable tier) is logged and the function returns `decimal.Zero`, so `ValidatePreTradeSafety` proceeds with zero RMR.

A misconfigured market or transient lookup error effectively disables the pre-trade margin guardrail instead of blocking the order. Consequently, `ValidatePreTradeSafety` can approve orders that would otherwise breach margin once the real RMR is computed.

Note that this pattern also applies to other paths such as `calculatePerSymbolWithdrawalRequirement` since `GetMarket failure` skips symbol's margin and `calculateSymbolRMR` would also return zero on failure.

## RECOMMENDATION

Consider propagating the error to reject the order.

## RESOLUTION

Status: Pending.

LOW **L-10**

PENDING

# Simultaneous Deposits Can Drop One

|          |                                                                                                                                                                                                                                                                                                                                                             |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                                                                                                                                               |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/handlers/account_events.go#L151">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/handlers/account_events.go#L151</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                                 |

## — DESCRIPTION

In `handleDeposit` (`account_events.go`), if two deposit events for a new subaccount arrive concurrently, both `GetActor` calls can fail before the actor is stored, and both proceed to `CreateActor` with initial balances.

Only one actor is stored and the other deposit isn't sent to an existing actor and is effectively lost. Consequently, a user's on-chain deposit can be missing from off-chain state when their first deposits race.

## — RECOMMENDATION

Serialize actor creation per subaccount.

## — RESOLUTION

Status: Pending.

LOW **L-11**

PENDING

# Wildcard Test Deposit Endpoint

|          |                                                                                                                                                                                                                                                                                                                               |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Warning                                                                                                                                                                                                                                                                                                                       |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/api/handlers/test/deposit.go#L177">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/api/handlers/test/deposit.go#L177</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                   |

## DESCRIPTION

The public API registers /test/deposit (with TestHandler) and HandleTestDeposit simply requires an auth\_secret that is compared against `SNXAPI_COMPETITION_TIME_DEPOSIT_SECRET`

The default docker-compose value for that variable is "\*" so `validateAuthSecret` returns success for any submitted auth\_secret. Once the handler accepts the request it publishes a RelayerEventDeposit to NATS, which downstream services treat as genuine deposits, letting any attacker credit arbitrary subaccounts/wallets without authentication.

There is currently a TODO to remove this, hence this issue is marked as Low severity.

## RECOMMENDATION

Remove or disable the /test/deposit endpoint in production.

## RESOLUTION

Status: Pending.

LOW **L-12**

PENDING

## Cancels Are Not Rate Limited

|          |                                                                                                                                                                                                                                                                                                                                                         |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Warning                                                                                                                                                                                                                                                                                                                                                 |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/handlers/cancel_orders.go#L33">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/handlers/cancel_orders.go#L33</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                             |

### — DESCRIPTION

`CancelOrder` iterates through all requested OrderIds and create a Go routine for each one. Unlike order placement, cancellation does not run through the rate limiter, allowing users to submit large batches of cancellations and spike resource consumption. A user does not necessarily have to submit valid Order ID's to cancel in the request in order to consume resources.

### — RECOMMENDATION

Consider introducing the ability to apply rate limiting on cancellations or enforcing a maximum batch size.

### — RESOLUTION

Status: Pending.

LOW **L-13**

PENDING

## Trade Safety Check Uses Optimistic Fill Prices

|          |                                                                                                                                                                                                                                                                                                                                                                       |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                                                                                                                                                                                                                                                                                                                         |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/margin_calculator.go#L449">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/subaccount/margin_calculator.go#L449</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                                           |

### — DESCRIPTION

`ValidatePreTradeSafety` simulates post-fill account state to prevent trades that would immediately breach margin requirements.

However, it prices orders optimistically:

- Market orders: Uses markPrice
- Limit orders: Uses limit price

Large market orders or aggressive limit orders that walk the book can fill across many price levels at progressively worse prices. The simulation doesn't account for this price impact, hence AAV can be overstated.

### — RECOMMENDATION

Be aware of this risk and/or consider including some form of price impact estimate for pricing.

### — RESOLUTION

Status: Pending.

LOW **L-14**

PENDING

# Worker Pool Panic Causes Permanent Message Loss

|          |                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | DoS                                                                                                                                                                                                                                                                                                                                                                                                     |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/subaccount/internal/services/subaccount/open_order_worker_pool.go#L118">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/subaccount/internal/services/subaccount/open_order_worker_pool.go#L118</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                                                                             |

## DESCRIPTION

The subaccount service uses sharded worker pools to process NATS JetStream events for open orders, positions, and trade history. Each worker pool uses consistent hashing (`subAccountID % numWorkers`) to route all events for a given subaccount to the same worker, ensuring ordering guarantees. However, if a worker goroutine panics, it terminates permanently with no recovery mechanism:

When a worker dies:

- Its channel remains open but unread
- New jobs queue up until the buffer fills
- Subsequent jobs are dropped after a 2ms timeout at submission
- Messages are ACKed to NATS even on handler failure, preventing redelivery

Since consistent hashing routes all events for specific subaccounts to the same worker, a single worker crash causes all subaccounts where `id % numWorkers == deadWorkerID` to lose event persistence indefinitely until the entire service restarts. This creates a durability gap where the trading service (in-memory) and subaccount service (PostgreSQL) diverge. If the trading service subsequently restarts, it syncs from the incomplete database state with function `SyncSubaccountsFromService()`, potentially loading stale positions, orders, or collateral balances.

## RECOMMENDATION

Add panic recovery within the worker loop to prevent goroutine termination.

## RESOLUTION

Status: Pending.

LOW **L-15**

PENDING

# Auto-Exchange Double-Deducts Collateral

|          |                                                                        |
|----------|------------------------------------------------------------------------|
| CATEGORY | Logical Error                                                          |
| LOCATION | internal/autoexchanges/manager.go, internal/subaccount/slp_transfer.go |
| REVIEW   | Main Review                                                            |

## DESCRIPTION

When the auto-exchange manager sells collateral to cover a subaccount's debt, it removes the sold amount twice.

The manager immediately subtracts `quantityToSell` from the source collateral balance:

```
``plain text
newQuantity := sourceCollateral.Quantity.Sub(quantityToSell)
if newQuantity.LessThan(decimal.Zero) {
return fmt.Errorf("insufficient %s balance for auto-exchange", collateralSymbol)
}
```

```
`
then invokes TransferCollateralToSLP() :
`plain text
```

```
if m.slpTransfer != nil {
err := m.slpTransfer.TransferCollateralToSLP(
`
```

which also subtracts the same amount after sending the transfer message (by design the transfer helper handles the deduction only after the send succeeds).

```
`plain text
// Transfer helper assumes balances are untouched until after a successful send
if err := slpActor.Send(transfer); err != nil {
... return err
}
newSourceQuantity = sourceCollateral.Quantity.Sub(amount)
...
st.actor.subAccount.Collaterals[collateralSymbol] = types.Collateral{ ... Quantity:
newSourceQuantity }
`
```

The two calls stack up and the user ends up losing 2x the sold collateral.

No unit test covers this flow; `updateCollateralBalances()` is only called with `slpTransfer == nil`, so the duplicate subtraction went unnoticed.

---

**L-15** Auto-Exchange Double-Deducts Collateral**CONTINUED**

---

**— RECOMMENDATION**

Remove the upfront subtraction in `updateCollateralBalances()` and let `TransferCollateralToSLP()` perform subtraction for auto-exchange flows (like it already does for liquidation).

**— RESOLUTION**

Status: Pending.

LOW **L-16**

PENDING

# No Input Length Validation on Decimal Parsing

|          |                                                                                                                                                                                                                                                                                                                                                                   |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Validation                                                                                                                                                                                                                                                                                                                                                        |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/handlers/withdrawal_handler.go#L58">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/trading/internal/handlers/withdrawal_handler.go#L58</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                                       |

## DESCRIPTION

Several handlers parse user-supplied amount strings into arbitrary-precision decimals using `decimal.NewFromString()` without first validating the input length. This occurs in:

- `lib/api/handlers/trade/withdraw_collateral.go` at the API layer
- `services/trading/internal/handlers/withdrawal_handler.go` at the trading service layer

While gRPC imposes a default 4MB message size limit, a moderately large numeric string (hundreds of thousands of digits) within that limit could still cause elevated CPU and memory usage during parsing and conversion to `big.Int`.

An attacker could submit withdrawal requests with excessively long amount strings to cause minor resource consumption on the API and trading services.

## RECOMMENDATION

Add a simple length check before parsing amount strings.

## RESOLUTION

Status: Pending.

LOW **L-17**

PENDING

## Silent Success Response When NATS Unavailable

|          |                                                                                                                                                                                                                                                                                                                                                                             |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CATEGORY | Warning                                                                                                                                                                                                                                                                                                                                                                     |
| LOCATION | <a href="https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/subaccount/internal/services/subaccount/service.go#L3436">https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/subaccount/internal/services/subaccount/service.go#L3436</a> |
| REVIEW   | Main Review                                                                                                                                                                                                                                                                                                                                                                 |

### — DESCRIPTION

In the subaccount service's withdrawal handling logic, when the NATS client is unavailable, the code logs a warning but continues to return a success response to the caller rather than returning an error. At this point in the flow, the withdrawal has already been persisted to the database in CREATED status, and the trading actor has reserved the requested funds by moving them from available balance to PendingWithdraw. However, without the NATS publish, the withdrawal request never reaches the relayer for on-chain submission, hence the withdrawal remains stuck.

### — RECOMMENDATION

Return an error when NATS is unavailable.

### — RESOLUTION

Status: Pending.

LOW **L-18**

PENDING

# Assets Can Disappear During Liquidation

CATEGORY Unexpected Behavior  
LOCATION liquidation\_engine.go  
REVIEW Main Review

## DESCRIPTION

`handleSLPTakeover()` skips transferring any position or collateral whose SLP actor lookup returns nil, yet it still calls `clearLiquidatedAccount`:

```
``plain text
for symbol, position := range le.actor.subAccount.Positions {
 slpActor := le.actor.manager.GetSLPActorForMarket(symbol)
 if slpActor == nil {
 le.logger.Warn("No SLP actor found for market", ...)
 // Skip this position if no SLP mapping exists
 continue
 }
 ...
}
for collateralSymbol, collateral := range le.actor.subAccount.Collaterals {
 slpActor := le.actor.manager.GetSLPActorForCollateral(collateralSymbol)
 if slpActor == nil {
 le.logger.Warn("No SLP actor found for collateral", ...)
 // Skip this collateral if no SLP mapping exists
 continue
 }
 ...
}
le.clearLiquidatedAccount(collateralSymbolsToDelete)
``plain text
func (le *LiquidationEngine) clearLiquidatedAccount(...) {
 for symbol, position := range le.actor.subAccount.Positions {
 le.publishPositionCloseEvent(symbol, position)
 }
 le.actor.subAccount.Positions = make(map[string]types.Position)
```

**L-18** Assets Can Disappear During Liquidation

CONTINUED

```
...
}
...
```

Which publishes close events and zeroes every asset. As a result, skipped positions/collaterals are deleted without being reassigned, producing permanent loss when SLP lookup fails.

However the scenario is unlikely in production because SLP actors are pre-seeded and cached, so severity reduced to Medium despite high impact.

**— RECOMMENDATION**

Abort or retry the takeover when `slpActor == nil`, or track which assets actually transferred and only clear those.

**— RESOLUTION**

Status: Pending.

LOW **L-19**

PENDING

# Collaterals Not Cleared clearLiquidatedAccount

CATEGORY Unexpected Behavior  
LOCATION liquidation\_engine.go  
REVIEW Main Review

## DESCRIPTION

`clearLiquidatedAccount()` receives `collateralSymbolsToDelete` param but never uses it, so the collateral map is never deleted after liquidation. Each symbol stays in `le.actor.subAccount.Collaterals` with `Quantity = 0`, causing future SLP-takeover runs to keep looping those stale entries and logging warnings.

Impact is low (bounded CPU/log noise, no financial risk), but the function does not perform what it is intended to do as it should clear the collaterals as the natspec suggests.

```
``plain text
// clearLiquidatedAccount clears all positions, collaterals, and balances from the liquidated
account
func (le *LiquidationEngine) clearLiquidatedAccount(collateralSymbolsToDelete []string)
``
```

## RECOMMENDATION

Inside `clearLiquidatedAccount()`, iterate the `collateralSymbolsToDelete` slice (or simply reinitialize `le.actor.subAccount.Collaterals = make(map[string]types.Collateral)`) so collateral keys are actually removed once liquidation completes.

## RESOLUTION

Status: Pending.

LOW **L-20**

PENDING

## TP/SL Event Retry

|          |                                       |
|----------|---------------------------------------|
| CATEGORY | Error                                 |
| LOCATION | services/subaccount/event_handlers.go |
| REVIEW   | Main Review                           |

### — DESCRIPTION

`handlePositionTPSLEvent()` retries only when the DB error string contains "position not found." Any other failure (deadlock, constraint violation, serialization failure, etc.) is treated as non-retryable: the message is ACK'd immediately and never reprocessed, so TP/SL updates are silently lost.

```
``plain text
if !strings.Contains(err.Error(), "position not found") {
s.logger.Warn("Failed to update position TP/SL with non-retryable error", ...)
_ = msg.Ack()
return
}
```

A transient DB hiccup or unexpected validation error discards a user's stop-loss/take-profit configuration, leaving positions unmanaged without alerting.

### — RECOMMENDATION

Introduce proper error classification, retry/Nak transient conditions, and only Ack when the payload is definitively invalid. Emit alerts for permanent failures.

### — RESOLUTION

Status: Pending.

LOW **L-21**

PENDING

# Database Init Error Handling

|          |                                |
|----------|--------------------------------|
| CATEGORY | Suggestion                     |
| LOCATION | services/subaccount/service.go |
| REVIEW   | Main Review                    |

## DESCRIPTION

`InitDB()` treats failures while preparing critical collateral columns (`ltv`, `lltv`) as warnings and then keeps going.

for example:

```
`plain text`
if err := db.Exec("ALTER TABLE collaterals ADD COLUMN ltv DECIMAL(5,2) DEFAULT 0.8").Error; err
!= nil {
 logger.Warn("Failed to add ltv column, may already exist", "error", err)
}
`
```

`AutoMigrate()` for the collateral tables runs immediately afterward and expects those constraints to be in place, so the service can run against an inconsistent database.

## RECOMMENDATION

Treat these preparatory migrations as fatal: return the error (or wrap it) instead of logging and continuing

## RESOLUTION

Status: Pending.

LOW L-22

PENDING

# Unbounded Rebroadcast Loop

CATEGORY Warning  
LOCATION watcher.go  
REVIEW Main Review

## DESCRIPTION

The watcher rebroadcasts any stuck tx every 15s once it ages past `stuckTimeout`, each time bumping the gas price. There's no cap on attempts or gas ceiling. If a tx is repeatedly stuck, the loop will continue rebroadcasting indefinitely, potentially pushing replacement gas prices higher than intended and consuming operator funds unnecessarily.

## RECOMMENDATION

Consider a max gas price, a cap on rebroadcast attempts, or some larger back-off period.

## RESOLUTION

Status: Pending.

LOW **L-23**

PENDING

# Inconsistent Behaviour In recoveryMode

CATEGORY Validation  
LOCATION orders.go  
REVIEW Main Review

## DESCRIPTION

`publishCancelOrder()` and `publishModifyOrder()` skip publishing if `a.recoveryMode` is `true` (returns `nil`, simulating success). But `publishOrder()` has no such check

```
``plain text
// Skip publishing in recovery mode
if a.recoveryMode {
return nil
}
`
```

Therefore during recovery mode new orders could error while cancels/modifies returns `nil`, causing inconsistent behaviour.

## RECOMMENDATION

Add recovery mode validation in `publishOrder()` like the other functions

## RESOLUTION

Status: Pending.

LOW **L-24**

PENDING

# Price Feed Limitations During Liquidations

CATEGORY Configuration

LOCATION subaccount/liquidation\_engine.go, subaccount/liquidation\_engine\_process\_state.go

REVIEW Main Review

## DESCRIPTION

Liquidations halt when the cached mark price is zero, causing the liquidation flow to error and retry indefinitely.

`calculateBankruptcyPrice()` returns `decimal.Zero`, and the caller aborts with "invalid bankruptcy price," so `ProcessLiquidationState()` retries forever without placing orders.

```
``plain text
markPrice := market.MarkPrice
if markPrice.IsZero() {
le.logger.Error("No mark price available for bankruptcy calculation",
"subAccountID", le.actor.subAccount.ID,
"symbol", symbol,
)
return decimal.Zero
}
`
`plain text
bankruptcyPrice := le.calculateBankruptcyPrice(*position, market, symbol)
if !bankruptcyPrice.IsPositive() {
le.logger.Warn("Invalid bankruptcy price calculated", ...)
return decimal.Zero, fmt.Errorf("invalid bankruptcy price: %s", bankruptcyPrice.String())
}
``
```

## RECOMMENDATION

Clearly document this risk and monitor/alert for lacking mark prices.

## RESOLUTION

Status: Pending.

LOW L-25

PENDING

## calculateReduceOnlyAdjustment Race Condition

CATEGORY Unexpected Behavior

LOCATION orders.go

REVIEW Main Review

### DESCRIPTION

In `calculateReduceOnlyAdjustments()` there is a loop to prevent race condition for multiple `types.OperationCreateOrder` that may exceed `position.Quantity` which is a cap to ensure that reductions do not exceed position size and end up flipping the trade. This does not take into account pending `types.OperationCancelOrder`, hence can cause unnecessary adjustments.

For example:

1. User has 10 ETH long position
2. User places reduce-only sell 8 ETH (Order A) → added to `OpenOrders`
3. User cancels Order A → stays in `OpenOrders` with `State = OrderStateCancel`
4. User places reduce-only sell 10 ETH (Order B)
5. `calculateReduceOnlyAdjustments()` sees: 8 (Order A, pending cancel) + 10 (Order B) = 18 > 10
6. System adjusts Order A (which is already being cancelled)

### RECOMMENDATION

Document that cancellations are not considered when adjusting reduce-only, creating a more conservative behavior.

### RESOLUTION

Status: Pending.

INFORMATIONAL I-01

PENDING

## safeAddOnelfGreater Unused Function

CATEGORY Best Practices

LOCATION <https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e0105776cb36682919bd18a7a5b5b95655b2e807/services/relayer/internal/blockchain/listener.go#L299>

REVIEW Main Review

### DESCRIPTION

The `safeAddOneIfGreater` function is unused throughout the off-chain code. Consider removing the unused code.

### RECOMMENDATION

Consider removing the unused code.

### RESOLUTION

Status: Pending.

## INFORMATIONAL I-02

PENDING

# Incomplete Response On Withdraw

CATEGORY Best Practices

LOCATION [https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/lib/api/handlers/trade/withdraw\\_collateral.go#L149](https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/lib/api/handlers/trade/withdraw_collateral.go#L149)

REVIEW Main Review

## DESCRIPTION

The WithdrawCollateralResponse struct has timestamp fields that are never populated in `Handle_withdrawCollateral`.

```
``plain text
response := WithdrawCollateralResponse{
RequestID: ctx.RequestId, // Use API request ID for end-to-end correlation
Symbol: Symbol(grpcResp.Symbol),
Amount: grpcResp.Amount,
Destination: grpcResp.Destination,
Status: grpcResp.Status.String(), // Use actual status from database ("created",
"requestedOnChain", "completed", "failed")
}
```

## RECOMMENDATION

Either populate timestamp fields or remove them from response struct.

## RESOLUTION

Status: Pending.

INFORMATIONAL I-03

PENDING

## Certain Transactions Don't Wait For Mining

CATEGORY Warning  
LOCATION transaction\_manager.go  
REVIEW Main Review

### — DESCRIPTION

Unlike `SubmitWithdrawalRequest`, functions `CastWatcherVote` and `SubmitBatchDisbursal` return immediately after `SendTransaction` without waiting for confirmation. This inconsistency between the functions should be clearly documented, since if there is an issue with the tx the caller would proceed as if it was successful.

### — RECOMMENDATION

Clearly document this behavior.

### — RESOLUTION

Status: Pending.

INFORMATIONAL **I-04**

PENDING

# ERC-1271 Signatures Unsupported

CATEGORY Logical Error

LOCATION eip712.go

REVIEW Main Review

## — DESCRIPTION

The off-chain signature verification only supports EOA signatures. `isValidSignature(bytes32 hash, bytes signature)` is not called anywhere hence smart contract signatures documented in ERC-1271 are not supported.

## — RECOMMENDATION

Clearly document this restriction.

## — RESOLUTION

Status: Pending.

INFORMATIONAL I-05

PENDING

# handleWithdrawal Processes Negative Quantities

CATEGORY Informational

LOCATION actor\_withdrawal.go

REVIEW Main Review

## DESCRIPTION

The actor does not check that `withdrawal.Quantity` is positive.

```
`plain text`
```

```
// Deduct from quantity and move to pending withdrawal
```

```
collateral, exist := a.subAccount.Collaterals[withdrawal.Collateral]
```

```
...
```

```
collateral.Quantity = collateral.Quantity.Sub(withdrawal.Quantity)
```

```
collateral.PendingWithdraw = collateral.PendingWithdraw.Add(withdrawal.Quantity)
```

```
`
```

A negative value will mint balance.

The public REST handler enforces `amount > 0`, but the gRPC endpoint

( `HandleWithdrawCollateral()` ) does not. Attack requires a compromised service that can call gRPC directly, this is unlikely however a sanity check is still recommended.

## RECOMMENDATION

Validate `withdrawal.Quantity` is greater than zero.

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-06

PENDING

# Inconsistent InitialMargin Value

CATEGORY Informational  
LOCATION internal/handlers/grpc\_handler.go  
REVIEW Main Review

## DESCRIPTION

For the `InitialMargin` value, the `GetSubaccount()` function returns only position IMR, but `ListSubaccounts()` return IMR + RMR, so clients see two different "initial margin" totals for the same subaccount depending on which function they call.

```
``plain text
/subaccount/internal/handlers/grpc_handler.go
// GetSubaccount
InitialMargin: subAccount.CumulativeIMR.String(),
...
// ListSubaccounts
InitialMargin: subAccount.CumulativeIMR.Add(subAccount.CumulativeRMR).String(),
``
```

## RECOMMENDATION

Have `GetSubaccount()` report `CumulativeIMR + CumulativeRMR` so every API reflects the total margin locked by positions plus open orders, consistent with trading-engine calculations.

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-07

PENDING

# GetPositionsBySymbol Response Not Populated

|          |                                              |
|----------|----------------------------------------------|
| CATEGORY | Informational                                |
| LOCATION | subaccount/internal/handlers/grpc_handler.go |
| REVIEW   | Main Review                                  |

## DESCRIPTION

The gRPC handler `GetPositionsBySymbol()` allocates `LongPositions` and `ShortPositions` slices but never fills them. The loops are empty:

```
```plain text
respLongPositions := make([]*grpc.PositionItem, len(longPositions))
respShortPositions := make([]*grpc.PositionItem, len(shortPositions))

for i := 0; i != len(longPositions); i++ {
}

for i := 0; i != len(shortPositions); i++ {
}
```
```

So the returned response every entry is nil. Dashboards and other consumers relying on this receive empty data.

## RECOMMENDATION

Replace the empty loops with a call to the existing `convertPositionItems` helper, mirroring `GetPositions`:

```
```plain text
respLongPositions := h.convertPositionItems(longPositions)
respShortPositions := h.convertPositionItems(shortPositions)
```
```

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-08

PENDING

# Skipped Transactions Can Undercount Totals

CATEGORY Informational  
LOCATION services/subaccount/service.go  
REVIEW Main Review

## DESCRIPTION

`GetFundingPayments()` drops any transaction whose metadata fails to unmarshal.

```
``plain text
if err := json.Unmarshal(tx.Metadata, &metadata); err != nil {
s.logger.Warn("Failed to parse funding payment metadata", "error", err, "transaction_id",
tx.ID)
continue
}
`
```

Those entries don't reach the aggregation loop, so `totalFundingReceived`, `totalFundingPaid`, and the returned `FundingHistory` silently omit them. Users may see incomplete summaries with no indication that data was skipped.

## RECOMMENDATION

Either fail the request when metadata parsing fails (so clients can retry/correct) or include the transaction while flagging the row/response as incomplete. At minimum, surface a warning in the API response indicating partial aggregation so consumers know totals may be understated.

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-09

PENDING

# Inconsistent Rounding Between Deposits and Pnl

CATEGORY Informational  
LOCATION actor.go  
REVIEW Main Review

## DESCRIPTION

Deposits and withdrawals update USDT collateral with six-decimal precision ( `updateCollateral()` ), but realized PnL settlements and funding-rate adjustments are rounded to `types.MarginPrecision` (2 dp) before they ever hit the collateral ledger. Any gain/loss smaller than a cent is silently discarded, so cumulative rounding always favors the protocol and makes on-chain/off-chain accounting diverge.

## RECOMMENDATION

While rounding in favor of the protocol is best practice for DeFi, it is usually only on the wei factor, not 2 decimal factor. Consider aligning all balance changes on a single precision (6) and document the rounding policy. This avoids the perception of hidden protocol spread.

## RESOLUTION

Status: Pending.

INFORMATIONAL I-10

PENDING

# Missing Collateral Due To Unhandled Error

CATEGORY Informational  
LOCATION manager.go  
REVIEW Main Review

## DESCRIPTION

In `mapGRPCToSubAccount()` if `decimal.NewFromString()` returns `err`, it simply logs it and skips adding the collateral entry.

There is a TODO statement:

```
``plain text
pendingWithdraw, err := decimal.NewFromString(col.PendingWithdrawalAmount)
if err != nil {
m.logger.Error("Failed to parse pending withdrawal amount", "error", err, "collateral",
col.Collateral)
continue // TODO: should we return an error here?
}
``
```

This can cause collateral entry to be skipped resulting in shrinking the available margin.

## RECOMMENDATION

This is informational as there is already a `TODO` statement, but it is recommended to return `err` instead of `continue`

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-11

PENDING

# Bulk Order Cancellation May Lose Results

CATEGORY Informational

LOCATION orders.go

REVIEW Main Review

## DESCRIPTION

The `handleCancelAllOpenOrdersBySymbol` function uses a 30-second timeout that abandons result collection early, causing successful order cancellations completed after the timeout to be lost and incorrectly marked as "failed" in the response. This provides incomplete and inaccurate information about order states.

## RECOMMENDATION

Consider using `sync.WaitGroup` to reliably return all states.

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-12

PENDING

# Margin Not Cleared On Error

CATEGORY Best Practices

LOCATION orders.go

REVIEW Main Review

## DESCRIPTION

In `handleOrder` (after `a.marginCalc.ReserveMarginForOrder` call)  
If `publishOrder` fails (e.g., NATS error), the function returns an error response but does not unreserve/rollback the margin previously reserved via `a.marginCalc.ReserveMarginForOrder`.  
However impact is low because these stale values don't affect actual margin calculations since failed orders are never added to `pendingOps` or `OpenOrders`.  
This is more a state consistency issue that may confuse debugging but doesn't break any system behavior. Contrast to `modifyOpenOrder`, which does rollback on publish failure.

## RECOMMENDATION

```
``plain text
err = a.publishOrder(ctx, request.RequestID, request.Order)
if err != nil {
 // Rollback margin reservation since order was not successfully submitted
 request.Order.RMR = decimal.Zero
 request.Order.MarginReserved = decimal.Zero
 request.Order.IMR = decimal.Zero

 respond.Error(request.Order.ID, request.RequestID, err)
 return
}
``
```

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-13

PENDING

# Legacy Code Not Cleaned Up

CATEGORY Informational  
LOCATION `margin_calculator.handleIndexTick()`  
REVIEW Main Review

## DESCRIPTION

There is a block of commented out legacy code in `margin_calculator.handleIndexTick()`

```
``plain text
// // Recalculate values with new price
``
```

## RECOMMENDATION

Remove commented out code before production to reduce confusion.

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-14

PENDING

# Unused Parameters

CATEGORY Informational  
LOCATION /subaccount/events.go  
REVIEW Main Review

## DESCRIPTION

`publishIndexPriceUpdate(ctx context.Context, symbol string, collateral types.Collateral)`, the parameters `symbol` and `collateral` are passed but never used. The function loops over all collaterals in `a.subAccount.Collaterals` instead.

`publishPnlSettledUpdate(ctx context.Context, collateralName string, newQuantity, pendingWithdraw decimal.Decimal)`, the parameters `newQuantity` and `pendingWithdraw` are unused. The function pulls values from `a.subAccount.Collaterals` instead.

## RECOMMENDATION

Clean up unused params

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-15

PENDING

# SLP Collateral Logs Zero

|          |                 |
|----------|-----------------|
| CATEGORY | Informational   |
| LOCATION | slp_transfer.go |
| REVIEW   | Main Review     |

## DESCRIPTION

`TransferCollateralToSLP()` hard-codes `HaircutRate: decimal.NewFromInt(1)` so the SLP side always computes `post_haircut_value = 0`, even when the collateral should retain value.

Balances aren't impacted but logs sees a 100% haircut for every transfer.

```
``plain text
transfer := &SLPCollateralTransfer{
ID: fmt.Sprintf(collateralTransferIDFormat, ...),
SourceAccountID: st.actor.subAccount.ID,
CollateralSymbol: collateralSymbol,
Quantity: amount,
PricePerUnit: pricePerUnit,
HaircutRate: decimal.NewFromInt(1),
...
}
```

## RECOMMENDATION

Compute the actual haircut via `GetCollateralHaircutRate()`

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-16

PENDING

# created\_at Overwritten On Every Event

CATEGORY Informational  
LOCATION internal/subaccount/events.go  
REVIEW Main Review

## DESCRIPTION

`sendOpenOrderEvent()` sets both `CreatedAt` and `UpdatedAt` to `snx_lib_utils_time.Now()` for every action (create, modify, cancel, fill, TP/SL insertion, leverage adjustment, etc.).

```
``plain text
UpdatedAt: snx_lib_utils_time.Now(),
CreatedAt: snx_lib_utils_time.Now(),
``
```

Since the subaccount service reuses this helper for all open-order state changes, the true creation time is overwritten when order is modified.

## RECOMMENDATION

Only write the `UpdatedAt` for edits to preserve the original `CreatedAt` timestamp.

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-17

PENDING

# Unreachable Code Risk Of Stuck Withdraws

CATEGORY Informational

LOCATION event\_handlers.go

REVIEW Main Review

## DESCRIPTION

The withdraw flow contains error-handling branches that can never be reached, because `marginCalc.UpdateAccountState()` does not return errors. As a result, withdrawals that fail on-chain are always refunded correctly right now, but the dead code masks a latent risk: if future refactors make `UpdateAccountState()` start surfacing errors, previously completed withdrawals could become permanently stuck in `WITHDRAWAL_STATUS_AWAITING_ACTOR_UPDATE`.

The scenario is described below:

If there is an onchain withdraw failure, and then the actor rollback also fails. Funds will be permanently stuck in `WITHDRAWAL_STATUS_AWAITING_ACTOR_UPDATE`.

1. On-chain failure path: When a withdrawal fails on-chain (or the cleanup job detects it expired), `handleWithdrawCompletedEvent()` marks the row `WITHDRAWAL_STATUS_AWAITING_ACTOR_UPDATE`, records a `failure_reason`, and sends a `WithdrawProcessedEvent()` with Error to trading:

```
``plain text
services/subaccount/internal/services/subaccount/event_handlers.go
updates := map[string]any{
 "status": snx_grpc.WithdrawalStatus_WITHDRAWAL_STATUS_AWAITING_ACTOR_UPDATE,
 "request_id": requestID,
}
if !isSuccess {
 updates["failure_reason"] = errorMsg
}
`
```

1. Actor rollback failure: The trading actor should restore `PendingWithdraw` to available balance for failed withdrawals, but if that restoration path also errors (e.g., `marginCalc.UpdateAccountState()` fails), `_publishWithdrawActorUpdated()` is called with a non-empty Error:

2. Dead-end state: `handleWithdrawActorUpdatedEvent()` sees both the original `failure_reason` and the actor error, and intentionally leaves the record in `WITHDRAWAL_STATUS_AWAITING_ACTOR_UPDATE` expecting a retry that does not exist anywhere else in the codebase:

As a result, both the withdrawal and the user's collateral remain permanently stuck; no

**I-17** Unreachable Code Risk Of Stuck Withdraws

CONTINUED

job or handler reprocesses `WITHDRAWAL_STATUS_AWAITING_ACTOR_UPDATE`, so funds never return to the user. In the current implementation since `marginCalc.UpdateAccountState()` does not return errors, it is just an issue with superfluous since the error handling is unreachable. However there is risk that future refactor will surface this risk.

**— RECOMMENDATION**

Document the dependency: withdrawals rely on `UpdateAccountState()` never returning an error. Remove the unreachable branches or on refactor add a reconciliation/cleanup job that can re-drive `WITHDRAWAL_STATUS_AWAITING_ACTOR_UPDATE` rows if actor rollbacks ever start failing in future

**— RESOLUTION**

Status: Pending.

## INFORMATIONAL I-18

PENDING

# Unrealized PnL Considered Withdrawable

CATEGORY Logical Error

LOCATION actor.go

REVIEW Main Review

## DESCRIPTION

The `calculateWithdrawableAmount` function includes unrealized PnL when computing how much a user can withdraw, via `CalculateBalance()`. This is a deliberate design choice that allows traders to withdraw "paper profits" before positions are closed.

As documented in dYdX's post-mortem of the SUSHI/YFI incident, allowing withdrawal of unrealized PnL creates potential for:

- Socialized losses: If a trader withdraws unrealized gains and the position later moves against them into insolvency, the protocol's insurance fund (or other users) absorbs the loss.
- Market manipulation incentive: Attackers can pump mark price, withdraw "profits," then let the position liquidate - extracting value that was never truly realized.
- Liquidity drain during volatility: In extreme market conditions, many users withdrawing unrealized gains simultaneously can stress protocol solvency.

## RECOMMENDATION

Clearly document this design decision and risks. Carefully monitor margin requirements.

## RESOLUTION

Status: Pending.

## INFORMATIONAL I-19

PENDING

# Authentication Can Be Disabled

CATEGORY Warning

LOCATION <https://github.com/GuardianOrg/v4-offchain-team1-1764269086710/blob/e6a7d6ceea4c8525040a1e2a4b3fc0fa65c508d8/services/api/cmd/server/main.go#L253>

REVIEW Main Review

## DESCRIPTION

When `cfg.Auth.Enabled = false`, the `AccountAuthenticator` is never instantiated, causing `h.authenticator.Enabled()` to return `false`. In this state, the `authenticateRequest` function bypasses all signature verification and ownership checks:

```
``plain text
if !h.authenticator.Enabled() {
walletAddress := req.WalletAddress()
if walletAddress == "" {
walletAddress = "0x0000000000000000000000000000000000000000000000000000000000000001"
}
return walletAddress, subAccountId, nil
}
``
```

When authentication is disabled, no EIP-712 signature verification occurs and any user can impersonate any wallet and act on any subaccount.

## RECOMMENDATION

Ensure production deployments always have `Auth.Enabled = true`

## RESOLUTION

Status: Pending.

INFORMATIONAL I-20

PENDING

# Missing End-to-End On-Chain Integration Tests

|          |             |
|----------|-------------|
| CATEGORY | Warning     |
| LOCATION | Global      |
| REVIEW   | Main Review |

## — DESCRIPTION

The off-chain repo has no automated E2E tests that exercise the full stack (API → trading/subaccount → relayer → Anvil-deployed contracts → event consumption) for on-chain flows.

Relayer benchmarks simulate chain events and deposits have sample scripts, but there's no round trip that starts from the public API with real signatures, drives a contract call on a local chain, and asserts off-chain state updates.

Withdrawals are notably untested end-to-end: there's no flow that submits `withdrawCollateral`, runs the relayer against `SynthetixDepositContract` on Anvil, observes `WithdrawalRequested`, and verifies DB/NATS/WebSocket status transitions.

## — RECOMMENDATION

Add CI-friendly integration tests that spin up Anvil, deploy `SynthetixDepositContract`, start API/trading/subaccount/relayer, and run both deposit and withdrawal scenarios end-to-end. Use real signed requests, wait for the relayer to submit on-chain transactions, consume emitted events, and assert off-chain persistence and status updates. Add documentation in the README on how to run these tests locally.

## — RESOLUTION

Status: Pending.

## § 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).