Quantstamp

Dipcoin Perpetual

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Perpetual DEX
Timeline	2025-10-27 through 2025-11-28
Language	Move
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• dipcoinlab/dipcoin-perpetual #419edec
Auditors	• Tim Sigl Auditing Engineer • Adrian Koegl Senior Auditing Engineer • Hamed Mohammadi Auditing Engineer

Documentation quality	Medium
Test quality	Medium
Total Findings	24 Fixed: 13 Acknowledged: 9 Mitigated: 2
High severity findings	2 Acknowledged: 2
Medium severity findings	3 Fixed: 2 Acknowledged: 1
Low severity findings	12 Fixed: 7 Acknowledged: 3 Mitigated: 2
Undetermined severity findings	0
Informational findings	7 Fixed: 4 Acknowledged: 3

Summary of Findings

The **Dipcoin** protocol is a perpetual futures exchange built on the Sui blockchain. The system employs a hybrid architecture with an off-chain orderbook and matching engine, while settlement, margin management and order validation remain on-chain. The `exchange` module acts as the central coordinator, routing operations to specialized modules for trading, margin management, and risk enforcement. User funds are held in the `bank` module, separated from trading positions which are tracked in the `Perpetual` object. The `settlement` module manages the flow of funds between traders and the protocol on trading, margin adjustments, funding rate application, liquidations and auto-deleveraging (ADL).

Trading logic is split into three key modules: `isolated_trading` handles standard order execution, `isolated_liquidation` manages the liquidation of below maintenance ratio positions, and `isolated_adl` resolves bankruptcy situations by force-matching profitable positions against underwater positions. Trading is permissioned; standard trades can only be submitted by settlement operators holding a `SettlementCap`, effectively keeping the matching logic under the control of the Dipcoin team.

The protocol integrates with the Pyth Network for real-time price feeds. These oracle prices are critical for the system's risk management: they determine the execution price for liquidations and ADL events, drive the funding rate accumulation in the `funding_rate` module, and are used to calculate mark prices for margin requirements.

During the audit, a high-severity issue was identified where signature verification relies solely on off-chain systems (DIP-1). Several medium-severity issues were also identified, including an expiry buffer extending orders beyond the user-intended lifetime (DIP-2), immediate liquidation risk via MMR updates (DIP-3), cases where the insurance position can be increased through regular trading (DIP-4), a position size increase allowed below IMR (DIP-5), and a transaction denial-of-service via tx_hash front-running (DIP-6).

The repository contains two distinct test suites: a Sui Move unit test suite and a TypeScript integration test suite. Since coverage of both test suites combined cannot easily be calculated, it can only be estimated. Several core modules, including `isolated_trading` and `isolated_liquidation`, currently have minimal test coverage. The protocol's security posture would significantly benefit from the addition of fuzz tests and invariant tests to cover edge cases and ensure system stability under stress.

Fix-Review Update 2025-12-15:
In the fix review, the client resolved 13 of 24 findings, mitigated 2, and acknowledged 9 as accepted risk or design decisions. Overall, the implementation shows meaningful progress, but several items, especially those relying on off chain trust assumptions such as signature and zkLogin verification paths, remain intentionally unaddressed.

ID	DESCRIPTION	SEVERITY	STATUS
DIP-1	Missing On-Chain ZK Proof Verification	High	Acknowledged
DIP-2	Skipped Signature Verification for Privileged Addresses	High	Acknowledged
DIP-3	Expiry Buffer Extends Orders Beyond User-Intended Lifetime	Medium	Fixed
DIP-4	Immediate Liquidation Risk via MMR Update	Medium	Acknowledged
DIP-5	Transaction Denial-of-Service via `tx_hash` Front-Running	Medium	Fixed
DIP-6	Misleading Documentation Regarding Liquidations in Delisted Markets	Low	Fixed
DIP-7	Inconsistent Rounding in `apply_isolated_margin()` Functions Favors Users in some Cases	Low	Mitigated
DIP-8	Revoked Exchange Manager Caps Retain Partial Access	Low	Fixed
DIP-9	Funding Rates Can Continue to Accrue After Delisting	Low	Fixed
DIP-10	Hardcoded Oracle Staleness Threshold	Low	Acknowledged
DIP-11	Transaction Hash Index Pollution	Low	Mitigated
DIP-12	Multiple `S128` Helpers Can Produce Negative Zero Representation	Low	Fixed
DIP-13	Unwithdrawable Dust Accumulation in `Bank`	Low	Fixed
DIP-14	Incorrect Margin Ratio for `price = 0`	Low	Fixed
DIP-15	Several `Bank`s Can Exist for the Same Currency	Low	Acknowledged
DIP-16	Insurance-First Liquidation Design Differs From Industry Standard	Low	Acknowledged
DIP-17	Error Code 1 Used Twice	Low	Fixed
DIP-18	The `trade_margin_settlement()` Function Uses Incorrect Error Code for Missing Accounts	Informational	Fixed
DIP-19	Insurance Transfer Limits Are Adjustable by the Same `ExchangeManagerCap` that Performs the Transfers	Informational	Fixed
DIP-20	Asymmetric Funding Debt Handling Between Long and Short Positions	Informational	Acknowledged
DIP-21	Redundant Signature Verification on Order Cancellation	Informational	Fixed
DIP-22	Hardcoded 6-Decimal Assumption in Bank Conversion	Informational	Acknowledged
DIP-23	Possibility of Error Code Collision	Informational	Acknowledged
DIP-24	Redundant Additional Check for Insurance Pool	Informational	Fixed

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

> **Disclaimer**
> Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

Repo: https://github.com/dipcoinlab/dipcoin-perpetual/tree/release/mainnet-beta/(419edecfe37e8e9904fab8b9012a396f93e3c93b)

Files: contracts/sources/*

Files Excluded

Repo: https://github.com/dipcoinlab/dipcoin-perpetual/tree/release/mainnet-beta/(419edecfe37e8e9904fab8b9012a396f93e3c93b)

Files: submodules/*, pyth/*, scripts/*, deployments/*, docs/*, tests/*, src/*

Operational Considerations

- **Per-Leverage Open Interest Caps Must Be Fully Populated** The `evaluator::verify_oi_open_for_account()` function skips the open-interest check whenever the computed leverage index is greater than or equal to the length of `checks.max_allowed_oi_open`. Operators must ensure that `set_max_oi_open` is configured with limits covering every leverage tier that can be reached by the chosen IMR values; otherwise, high-leverage positions may bypass open-interest caps entirely.
- **Rotate Insurance Accounts Together With Balances** When changing the insurance fund's `active_account` or `security_account` via `insurance_fund::set_active_account()` or `insurance_fund::set_security_account()`, only the configured addresses are updated; existing balances remain at the old accounts. Operators should transfer funds from the old account to the new one as part of any rotation, otherwise the new accounts may be empty or underfunded when needed.
- **Insurance Fund Transfer Limits Are Per-Transaction, Not Cumulative** The `transfer_amount_limit` in `insurance_fund.move` restricts the maximum amount of a single transfer, not the cumulative volume over a period. Operators should be aware that multiple

transfers up to the limit can be executed sequentially as long as the `transfer_interval` (minimum time between transfers) is respected. This differs from a typical "quota per epoch" model and requires careful tuning of both the interval and the per-transaction limit to achieve the desired throughput cap.

- **Batch Size Limit in `remove_empty_positions`** The `position::remove_empty_positions()` function iterates through a user-provided vector of addresses. Since each table access counts towards the Sui dynamic field access limit (1,000 per transaction), operators must ensure that the `pos_keys` vector does not exceed this limit (minus overhead).
- **Funding-Window Timing** Because funding is settled at discrete window boundaries, users can try to time entries and exits just before or after those boundaries to slightly capture funding when they would receive it or avoid a charge when they would pay.
- **Oracle Trust Assumption** The protocol assumes the Pyth oracle provides accurate and timely price data. Oracle inaccuracies or latency directly affect liquidations, funding, and margin calculations.
- **Upgrade Governance Risk** All contract upgrades are assumed to undergo proper audits and review. Operators and admins must ensure disciplined upgrade processes to prevent introducing critical vulnerabilities.
- **Settlement Operator Signature Integrity** Users must trust that settlement operators do not forge signatures or exploit the `ZK_WALLET` bypass when validating off-chain approvals.
- **Settlement Operator Execution Fairness** Settlement operators are trusted not to front-run, extract MEV, censor orders, or manipulate order matching. They are expected to provide best execution and honest matching.
- **Funding Rate Operator Honesty** Funding operators are trusted to set fair and non-manipulative funding rates that reflect true market conditions.
- **ADL Target Selection Fairness** Deleveraging operators are assumed to select ADL targets fairly and without preferential or malicious bias.
- **Admin Non-Abuse of Emergency Controls** Admins must not abuse pause functionality (e.g., indefinitely freezing withdrawals or trading).
- **Admin Parameter Tampering Risk** Admins are trusted not to maliciously alter parameters (e.g., IMR/MMR, caps) to force user liquidations.
- **Insurance Fund Solvency and Management** The insurance fund is assumed to be properly funded, secured, and managed to cover bad debt and liquidation shortfalls.

Key Actors And Their Capabilities

Exchange Admin (`ExchangeAdminCap`)

Responsibilities:

- Mint and assign `ExchangeManagerCap`, `SettlementCap`, `DeleveragingCap`, and `FundingRateCap` (root role appointing all others).
- Upgrade modules / deploy new implementations.
- Pause and unpause the entire protocol (global emergency control).

Trust Assumptions: Root of trust. Compromise enables full protocol takeover, including replacing operators, upgrading code, force-delisting markets, or freezing operations indefinitely.

Exchange Manager (`ExchangeManagerCap`)

Responsibilities:

- **Market Management**: Create markets, update market status, *force delist markets*, and configure oracle price feeds.
- **Scoped Emergency Controls**: Pause / resume **trading and withdrawals per market**.
- **Risk Parameter Configuration**: Set IMR, MMR, fees, open-interest limits, funding caps (`max_funding`), order constraints (tick size, lot size, min quantity).
- **Insurance Fund Operations**: Update insurance fund accounts and execute transfers between active/security pools.
- **Operator & Privilege Assignment**: Grant/revoke `SettlementCap` roles and manage the privileged-address whitelist.

Trust Assumptions: High trust. Can force liquidations via parameter changes, drain or mismanage insurance funds, delist markets, or escalate privileges to malicious actors.

Settlement Operator (`SettlementCap`)

Responsibilities:

- **Trade Execution**: Submit matched maker–taker orders on-chain.
- **Execution Discretion**: Choose fill price and quantity within user-defined constraints.
- Typically held by **multiple operators**.

Trust Assumptions: Trusted not to censor orders, front-run, extract MEV, manipulate execution, or exploit the `ZK_WALLET` signature-bypass path. A malicious operator can legally execute at the worst price allowed by the user's order.

Deleveraging Operator (`DeleveragingCap`)

Responsibilities:

- **Auto-Deleveraging (ADL)**: Select ADL targets and force position closures **at bankruptcy price**.
- Typically held by a **single operator** for coordinated deleveraging.

Trust Assumptions: Trusted to trigger ADL only when legitimate and to select targets fairly, without bias or favoritism. On-chain checks limit abuse but do not remove operator discretion.

Funding Operator (`FundingRateCap`)

Responsibilities:

- **Funding Rate Updates**: Set funding rates and update funding index, bounded by `max_funding`.
- Typically held by a **single operator**.

Trust Assumptions: Trusted to publish honest, non-manipulative funding rates aligned with real market conditions.

Privileged Addresses

Responsibilities:

- **Signature Bypass**: Skip on-chain signature verification for faster execution.
- **Enhanced Limits**: Operate with up to **5× higher open-interest caps**.

Trust Assumptions: Operationally privileged but still constrained by protocol invariants (margin, balances, liquidation logic). Increased market power and increased risk due to amplified OI limits.

Traders (Regular Users)

Responsibilities:

- **Collateral Management**: Deposit / withdraw margin.
- **Trading**: Submit signed off-chain orders for execution.
- **Delisted Market Exit**: Close positions in delisted markets.

Trust Assumptions: Untrusted. All invariants should be enforced trustlessly by the protocol.

Findings

DIP-1 Missing On-Chain ZK Proof Verification

High | Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
Since we operate as an on-chain/off-chain hybrid system, our off-chain infrastructure requires user trust. Additionally, ZK proof verification exclusively applies to ZKLogin users and does not impact wallet users.
```

File(s) affected: `library.move`, `order.move`, `isolated_trading.move`

Description: In `library::verify_signature()`, when the scheme byte equals `SIGNED_USING_ZK_WALLET (3)`, the function sets `is_verified = true` without validating any zkLogin-related proofs or checking whether `public_key` corresponds to the claimed zkLogin identity.

The team indicated that these zkLogin proofs are verified off-chain, but the on-chain contracts do not check that such verification actually happened. As a result, any bug in the off-chain verification pipeline would make the on-chain authentication bypassable, since scheme `3` is accepted unconditionally.

Recommendation: For zkLogin / ZK wallets, the strongest option is to verify ZK proofs on-chain. If on-chain verification is not wanted, it should be clearly documented that zkLogin security is enforced entirely by off-chain infrastructure and that the contracts themselves do not verify any zkLogin signatures or proofs.

DIP-2 Skipped Signature Verification for Privileged Addresses

High | Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
To enhance efficiency, privileged addresses trade off a small degree of security. The security of privileged addresses is fully guaranteed by the off-chain system. If a privileged address is compromised, only the address itself will be affected—other users remain unaffected.
```

File(s) affected: `order.move`, `library.move`, `isolated_trading.move`

Description: `order::verify_order_signature()` skips signature verification when the `creator` is a privileged address listed in `ProtocolConfig.privileged_addresses`. By placing a privileged address into the `creator` field, any user can submit an order that is treated as if it were signed by that privileged account.

The team indicated that off-chain systems are used to prevent such misuse, but these protections are not visible or enforceable from the on-chain contracts. Because this behavior relies entirely on off-chain systems to prevent misuse, on-chain authentication becomes bypassable if any future function depends solely on the contract's signature verification logic.

Recommendation: Remove the unconditional skip for privileged addresses. All orders and cancellations should require a valid signature from the actual signer, regardless of privilege status. While skipping verification may save some gas costs for privileged users, we believe the security trade-off is not worth the added risk of weaker authentication.

DIP-3
Expiry Buffer Extends Orders Beyond User-Intended Lifetime

Medium | Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `00ecd36379ce6496c3bc8a5247f4ac224b9d52bf`.

File(s) affected: `order.move`

Description: The protocol applies an internal "expiry buffer" on top of the user-specified order expiry, effectively extending the validity window without the user's knowledge or control. This conflicts with common expectations where users already account for network and inclusion delays when setting expiries. As a result, orders can remain valid longer than intended, exposing users to additional market and counterparty risk after they believe the order has expired.

Exploit Scenario:
An advanced user wants a very short-lived order (e.g., "valid for 2 minutes") around a volatile event and sets the expiry timestamp accordingly, already pricing in expected transaction latency. The protocol then silently adds its own buffer (e.g., another minute), keeping the order fillable beyond the original 2-minute window. If the market moves sharply in that extra time, a relayer or counterparty can still match and settle the order at a now-unfavorable price, even though the user assumed the order was no longer executable.

Recommendation: Remove the protocol-side expiry buffer and treat the on-chain expiry as an exact, user-controlled boundary.

DIP-4 Immediate Liquidation Risk via MMR Update

Medium | Acknowledged

Update

Marked as "Acknowledged" by the client.
Addressed in: `6d1e07cd32e541c1a8502a21cde137ca1dec03e9`.
The client provided the following explanation:

> We added documentation for this function.

File(s) affected: `perpetual.move`

Description: The `perpetual::set_maintenance_margin_required()` function allows the Exchange Manager to update the Maintenance Margin Ratio (MMR) for a perpetual market. This change takes effect immediately for all existing positions. If the MMR is increased, positions that were previously safe (margin ratio > old MMR) but are below the new MMR (margin ratio < new MMR) become immediately liquidatable. This can lead to mass liquidations without user action or market movement, potentially causing significant user losses and system instability.

Exploit Scenario:

1. A user has a position with a margin ratio of 6%. The current MMR is 5%. The position is safe.
2. The Exchange Manager calls `set_maintenance_margin_required()` to increase the MMR to 7%.
3. The user's position (6%) is now less than the new MMR (7%).
4. The liquidator calls `exchange::liquidate()` on the user's position.
5. The user is liquidated immediately, losing their position and paying liquidation penalties.

Recommendation: Implement a timelock or a grace period for MMR increases. For example, announce the new MMR but only apply it after a delay (e.g., 7 days), allowing users to adjust their positions or add margin. Alternatively, apply the new MMR only to new positions or gradually increase it. Another option is to remove the function altogether since it presents a significant risk to users.

DIP-5 Transaction Denial-of-Service via `tx_hash` Front-Running

Medium | Fixed

Update

Marked as "Mitigated" by the client.
Addressed in: `7d2cb0ed1c29ae8e666fea30b63fb04171da5103`.
The client provided the following explanation:

```
We do not registry tx_hash through functions directly callable by user
```

File(s) affected: `sub_accounts.move`, `exchange.move`, `order.move`, `bank.move`

Description: All protocol operations requiring unique transaction hashes can be permanently blocked through front-running. The `validate_unique_tx` function accepts arbitrary user-supplied `tx_hash` values without verifying their relationship to the transaction content. Any observer who learns a transaction's `tx_hash` can submit it first, causing the legitimate transaction to revert permanently.

This affects all operations using the transaction indexer including trades, liquidations, deleveraging, order cancellations, margin operations, and fund transfers. Time-sensitive operations are particularly vulnerable. Users attempting to cancel orders before being filled or add margin before liquidation can have their transactions blocked, forcing them to retry with a new `tx_hash` while market conditions worsen.

The impact depends on transaction visibility. In fully private operator infrastructure with trusted transaction submission, this risk is mitigated. However, any public mempool visibility, RPC logging, or blockchain explorer access before transaction finalization exposes the `tx_hash` to potential front-runners. Additionally, a malicious or compromised operator could selectively deny service to specific users.

Recommendation: This issue would be mitigated by not registering any `tx_hash` through functions directly callable by the user, i.e. `cancel_order()`. To resolve this issue, consider computing the `tx_hash` on-chain by hashing a unique set of transaction parameters.

DIP-6
Misleading Documentation Regarding Liquidations in Delisted Markets

• Low Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `85359e49d94a7d10ee986e7c97b7323199aab339`.

File(s) affected: `perpetual.move`

Description: The documentation for `perpetual::delist_perpetual()` states that "Liquidations continue (prevents bankruptcy)" after a market is delisted. However, the code explicitly blocks liquidations via `assert!(!perpetual::delisted(perp))` in `exchange::liquidate()`. The developers have confirmed that liquidations are indeed intended to stop, and the documentation is incorrect.

Recommendation: Update the documentation for `perpetual::delist_perpetual()` to remove the documentation saying that liquidations continue even after market delisting.

DIP-7
Inconsistent Rounding in `apply_isolated_margin()` Functions Favors Users in some Cases

• Low Mitigated

Update

Marked as "Fixed" by the client.
Addressed in: `54e5e80878a4681706922b665ad5f471d91878d5`, `979b9692417943b0544086f9d38fb516aadcf390`.

Alert

The client adjusted the rounding behavior for the specific examples highlighted in this finding. However, additional calculations within `apply_isolated_margin()` still rely on implicit integer division rounding and were not updated. We strongly recommend reviewing **all** divisions in `apply_isolated_margin()` and explicitly defining the intended rounding direction for each calculation. Even when rounding down is the correct behavior, this should be clearly documented to make the assumption explicit and prevent future inconsistencies or regressions.

File(s) affected: `isolated_trading.move`, `isolated_liquidation.move`, `isolated_adl.move`

Description: In the `apply_isolated_margin` functions across `isolated_trading`, `isolated_liquidation`, and `isolated_adl`, integer arithmetic is used for margin calculations. While rounding generally favors the protocol (e.g., when reducing positions, the released margin is rounded down), there are cases where it favors the user.

For example, in `isolated_trading::apply_isolated_margin` (Case 1: Open/Increase):

```
// margin per unit = price * mro
margin_per_unit = library::base_mul(fill.price, mro);

// funds = qty * (margin_per_unit + fee_per_unit)
funds_flow = s128::from(library::base_mul(fill.quantity, margin_per_unit + fee_per_unit), true);
```

`library::base_mul()` rounds down. This means `margin_per_unit` is rounded down, and `funds_flow` (the amount the user pays) is further rounded down. Consequently, the user pays slightly less margin than the exact theoretical requirement.

While the discrepancy is very small, consistent rounding favoring the protocol is a standard security practice to prevent any potential for systematic exploitation or invariant breakage over time. As demonstrated by the recent Balancer hack, even small rounding errors, when exploited at scale or in specific edge cases, can lead to significant protocol losses.

Recommendation: Review all margin calculations in `apply_isolated_margin()` and ensure that rounding always favors the protocol.

- When calculating amounts the user **pays** (e.g., opening positions), round **up**.
- When calculating amounts the user **receives** (e.g., closing positions, reducing margin), round **down**.

DIP-8 Revoked Exchange Manager Caps Retain Partial Access

• Low ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `8d82558b66daf90dd2700f5b4555e213752db4f0`.

File(s) affected: `roles.move`, `perpetual.move`

Description: The `roles::set_exchange_manager()` function issues a new `ExchangeManagerCap` without invalidating the previous one. While most privileged functions validate the cap against access control list in the `ProtocolConfig`, the functions `perpetual::transfer_insurance_fund_from_active_to_security()` and `perpetual::transfer_insurance_fund_from_security_to_active()` only check for the cap's existence, allowing holders of revoked caps to continue performing these operations. This effectively creates a two-tier permission system where access to insurance fund transfers can never be revoked, and it is difficult to retrace which actors retain partial access.

Recommendation: If the intent is to have a distinct role for insurance fund operations, introduce a dedicated capability (e.g., `InsuranceOperatorCap`). Otherwise, ensure all functions requiring `ExchangeManagerCap` call `roles::check_manager_validity()` to verify the cap is currently active.

DIP-9 Funding Rates Can Continue to Accrue After Delisting

• Low ⓘ Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `61a9e639219efd051f063728c7b09160f107b6e3`.

File(s) affected: `perpetual.move`

Description: The documentation of the `perpetual::delist_perpetual()` function states that "Funding rates stop accruing" after a market is delisted. However, `perpetual::set_funding_rate()` does not check the `delisted` status. A funding rate operator can continue to call this function, updating the global funding index. Since `exchange::close_position()` settles funding against this global index, users closing positions later could pay significantly more funding than those closing earlier, depending on operator actions. Currently users have to trust the funding rate operator to not update funding rate after delisting.

Recommendation: In the `perpetual::set_funding_rate()` function, enforce that the perp is not delisted when updating the funding rate.

DIP-10 Hardcoded Oracle Staleness Threshold

• Low ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
ORACLE_PRICE_MAX_AGE is not used on-chain to ensure the freshness of oracle prices, but rather as a
final safeguard to confirm that the oracle provider is not down. When executing critical functions such
as trade, liquidate, and deleverage, our operator will check the freshness of oracle prices and update
```

```
the oracle price data. Currently, we only utilize the Pyth oracle, which fully leverages Pyth's price update mechanism.
```

File(s) affected: `contracts/sources/library.move`

Description: The protocol uses a global constant `ORACLE_PRICE_MAX_AGE` of 60 seconds for all Pyth price feeds in `library::get_oracle_price()`. Different assets may have different update frequencies or network conditions. A single hardcoded value might be too strict for less liquid assets (causing transaction failures) or too loose for highly volatile assets.

Recommendation: Store the max age threshold in the `Perpetual` struct or `ProtocolConfig` to allow per-market configuration. Additionally, implement administrative functions to update this threshold, allowing operators to adjust it if the initial value proves too strict or too loose.

DIP-11 Transaction Hash Index Pollution

Low Mitigated

Update

Marked as "Mitigated" by the client.
Addressed in: `7d2cb0ed1c29ae8e666fea30b63fb04171da5103`.
The client provided the following explanation:

```
similar to DIP-6, not registry tx_hash through functions directly callable by user
```

File(s) affected: `sub_accounts.move`

Description: The protocol maintains a platform-level global transaction index through `TxIndexer` to track all on-chain interactions. However, since `tx_hash` values are arbitrary user input without content verification, malicious or careless users can pollute this index with garbage values. The `TxIndexer.map` stores every submitted `tx_hash` permanently in an unbounded table with no cleanup mechanism.

While the development team intends `tx_hash` to serve as unique identifiers for legitimate transactions, nothing prevents users from submitting random or sequential values or even replaying the same payload with a different hash. An attacker could spam the system with bogus `tx_hash` values to bloat storage.

This issue does not create direct security vulnerabilities but impacts operational costs and index quality. Off-chain indexers and monitoring systems relying on this global transaction index may need to filter or validate entries to distinguish legitimate transaction identifiers from noise.

Recommendation: For order-related operations, calculate the order hash using the existing `library::get_hash()` function, which is cryptographically bound to the signed order. For other replay-sensitive operations, similarly construct a hash on-chain from the signed parameters (similar to recommendation in DIP-6).

DIP-12 Multiple `S128` Helpers Can Produce Negative Zero Representation

Low Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `41dad46bd1bb99ca2af41ac9506ee2a28513a238`.

File(s) affected: `signed_int128.move`

Description: Several helpers in `s128` can produce a 'negative zero' value, which is inconsistent with treating zero as always positive. This inconsistency can lead to subtle bugs where downstream code assumes all zero values have `sign == true`.

Recommendation: Normalize zero across all helpers so that `value == 0` always implies `sign == true`. Adjust the branch conditions or add a normalization step to ensure consistency.

DIP-13 Unwithdrawable Dust Accumulation in `Bank`

Low Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `3b3b82a1416250f7394a5767b4a23f4ba94160bc`.

File(s) affected: `bank.move`

Description: When users call `withdraw_all()`, some dust will remain in the contract that can never be withdrawn. The dust accrues due to the fact that 3 decimals of precision are removed. While the dust is negligible for a single user operation, dust may accumulate over time.

Recommendation: We recommend removing the rounding and working with asset-specific decimals.

DIP-14 Incorrect Margin Ratio for `price = 0`

Low Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `1129233c721840a8ea4746fb2b9bf11f43fb39f8`.

File(s) affected: `position.move`

Description: When the asset price is zero, the `compute_margin_ratio()` function will incorrectly return a margin of 1.0. This means that long positions, which are actually underwater (MR of minus infinity), will appear to have a margin of 1.0. While this should not occur in practice, edge cases may exist.

Recommendation: We recommend specifically handling this edge case of `position_value == 0`, instead of only calculating the `margin_ratio` when `position_value > 0`.

DIP-15 Several `Bank`s Can Exist for the Same Currency

Low Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
We can ensure create_bank can only be called once by the project administrator during contract
deployment, and will not be called again thereafter.
```

File(s) affected: `bank.move`

Description: The `create_bank()` function allows the `ExchangeManagerCap` to create a shared `Bank` object for a `Coin T`. However, there is no check to ensure that a `Bank` for `T` already exists. If several `Bank`s for the same `Coin T` existed, this would lead to fund and protocol fragementation.

Recommendation: We recommend creating a Singleton Registry to ensure a `Bank` object exists only once for each `Coin T`.

DIP-16 Insurance-First Liquidation Design Differs From Industry Standard

Low Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
This is a solution we have discussed after careful deliberation. We believe this solution is safer than
opening up liquidation permissions, because the insurance fund can accumulate positive premiums,
thereby having more capital to resist the directional risk . Under the solution you proposed, the
insurance fund only takes over insolvent positions; the directional risk  still remain, and the
insurance fund is exposed to higher risks.
```

File(s) affected: `exchange.move`, `isolated_liquidation.move`

Description: The protocol routes all liquidations through the insurance fund, which takes over positions and relies on an off-chain automated engine to close them via regular trades. This differs from industry-standard perpetual DEXs (GMX, dYdX, Hyperliquid) where liquidations are matched against orderbook liquidity first, with insurance funds only absorbing unmatched bad debt. Under the insurance-first design, market makers never see liquidation flow, potentially disadvantaging them compared to traditional orderbook-first systems. The insurance fund accumulates directional risk whenever liquidation closures lag behind new liquidations, even during normal volatility when orderbook liquidity exists.

The team's rationale centers on concerns about thin orderbook liquidity, and their off-chain engine is designed to immediately match insurance positions against the orderbook. However, this creates a critical operational dependency where the insurance fund becomes the required intermediary for all liquidations rather than a last-resort backstop. If the off-chain engine experiences delays (network congestion, orderbook

depth constraints), the insurance fund accumulates positions and directional exposure that would otherwise be absorbed directly by market makers.

Recommendation: Consider implementing hybrid liquidation where liquidations with positive premiums (remaining equity) can be executed permissionlessly by external liquidators competing for the premium, while only negative premium liquidations (bad debt) route through the insurance fund. This would align incentives with industry standards, reduce unnecessary directional risk for the insurance fund, and provide market makers access to liquidation flow.

DIP-17 Error Code 1 Used Twice

Low Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `f16ab4b812a1ddf5cf168df59f556d89dc700627`.

File(s) affected: `error.move`

Description: The system assigns error code **1** to two different errors: `invalid_protocol_version()` and `min_price_greater_than_zero()`. This duplication can cause confusion and make it harder to correctly identify the source of an error.

Recommendation: Consider assigning a different error code to one of these errors to ensure each error remains uniquely identifiable.

DIP-18
The `trade_margin_settlement()` Function Uses Incorrect Error Code for Missing Accounts

Informational Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `06c96c4cf68cf7c15b8d9d24f161707af977c9cd`.

File(s) affected: `settlement.move`

Description: In `settlement::trade_margin_settlement()`, the function checks if the maker and taker have bank accounts using `bank.exists_account()`. However, if this check fails, it aborts with `error::not_enough_balance_in_bank()` instead of the more appropriate `error::user_has_no_bank_account()`.

Recommendation: Update the assertions to use `error::user_has_no_bank_account()` when checking for account existence.

DIP-19
Insurance Transfer Limits Are Adjustable by the Same `ExchangeManagerCap` that Performs the Transfers

Informational Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `8d82558b66daf90dd2700f5b4555e213752db4f0`.
> The client provided the following explanation:
>
> ```
> Same with [DIP-9]
> ```

File(s) affected: `perpetual.move`

Description: The function `perpetual::transfer_insurance_fund_from_security_to_active()` is restricted by amount limits and time intervals to protect the security pool. However, the same `ExchangeManagerCap` required to call this function can also be used to modify these limits via `perpetual::set_insurance_fund_transfer_amount_limit()` and `perpetual::set_insurance_fund_transfer_interval()`. This means a compromised or malicious manager can bypass the safety controls by first relaxing the limits and then draining the pool.

Recommendation: Consider separating the roles for fund transfers and parameter configuration.

DIP-20

Asymmetric Funding Debt Handling Between Long and Short Positions

Informational | Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

> `Although the handling of long and short positions differs, the underlying logic is correct. We do not plan to make modifications for the time being.`

File(s) affected: `exchange.move`

Description: The protocol handles unpaid funding debt asymmetrically between long and short positions during ADL execution (flag=3). While long positions can accumulate unbounded debt by inflating `oi_open`, short positions have their debt capped at `oi_open` due to the unsigned integer type constraint.

Long Positions (exchange.move:1160-1162):

```
if (is_long) {
    // Debt deferred indefinitely by inflating oi_open
    position::set_oi_open(user_pos, oi_open + settlement_amount);
}
```

For longs, unpaid funding increases `oi_open`, which reduces future PnL and increases bankruptcy price. This effectively defers the debt to be paid when the position eventually closes.

Short Positions (exchange.move:1163-1175):

```
else {
    if (settlement_amount > oi_open) {
        // Debt exceeding oi_open is forgiven
        position::set_oi_open(user_pos, 0);
        emit(SettlementAmountNotPaidCompletelyEvent {
            amount: settlement_amount - oi_open,
        });
    } else {
        position::set_oi_open(user_pos, oi_open - settlement_amount);
    }
}
```

For shorts, `oi_open` (unsigned u128) cannot go negative. If funding debt exceeds `oi_open`, the excess is written off, meaning counterparty traders owed this funding would not receive it.

While this would negatively impact the protocol, the condition `settlement_amount > oi_open` is practically unreachable because positions are liquidated long before sufficient funding debt can accumulate. At the protocol's maximum 1% hourly funding rate, a position would need 100+ hours to accumulate debt exceeding `oi_open`, but margin depletion triggers liquidation within ~5 hours.

Nevertheless, the code asymmetry between long and short handling creates potential confusion.

Recommendation: While this code path is practically unreachable, if it were somehow triggered, the debt should not be written off. We recommend applying the same unbounded debt-tracking approach to shorts as used for longs (though this would require changing `oi_open` to a signed integer type, representing a significant architectural change).

DIP-21

Redundant Signature Verification on Order Cancellation

Informational | Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `f81c70f7e7ad18f925b52f2f38fbb56914d00534`.

File(s) affected: `order.move`

Description: The `cancel_order()` function verifies that the `creator` or a sub-account has created a valid signature (without replay protection). However, the function already verifies that the `creator` or a subaccount has submitted the order cancellation on-chain. As a result, the signature check does not add any security feature on top.

Recommendation: We recommend removing the signature verification.

DIP-22
Hardcoded 6-Decimal Assumption in Bank Conversion

Informational | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client.
> The client provided the following explanation:
>
> ```
> create_bank is invoked by the official party. We will ensure that the token has 6 decimal places, and additional parameters need to be added to CoinMetadata. Note that subsequent contract upgrades will be incompatible with this entry method.
> ```

File(s) affected: `bank.move`, `library.move`

Description: The `Bank<T>` contract is generic over coin type `T` but hardcodes decimal conversion assuming all coins have 6 decimals. The `convert_usdc_to_base_decimals()` function unconditionally multiplies amounts by 1000 to convert from 6 decimals to the internal 1e9 base units.

If a bank is created for coins with different decimal counts (e.g., SUI with 9 decimals), the internal accounting will be inflated or deflated by powers of 10:

- 9-decimal coins: Internal balances 1000x inflated
- 18-decimal coins: Internal balances divided by 1 billion

While this does not create an exploitable vulnerability (the system remains mathematically consistent since all users share the same conversion factor), it creates several operational risks:

1. **Misconfigured parameters**: Default insurance fund transfer limits are set assuming 6-decimal coins. For 9-decimal coins, the effective limit would be 1000x smaller than intended.
2. **Off-chain confusion**: Events and indexers would display balances that appear 1000x larger for 9-decimal coins.
3. **Misleading generics**: The `Bank<T>` generic type parameter suggests any coin type is supported, but only 6-decimal coins work as designed.

The issue stems from the function name `convert_usdc_to_base_decimals()` explicitly referencing USDC, yet being used for all coin types in the generic `Bank<T>` contract.

The team has confirmed that they intend to only support USDC as an asset for now.

Recommendation: Since only USDC should be supported, we recommend adding validation in `create_bank()` to ensure only 6-decimal coins can be used:

```
let decimals = coin::get_decimals<T>();
assert!(decimals == 6, error::unsupported_coin_decimals());
```

DIP-23 Possibility of Error Code Collision

Informational | Acknowledged

> **Update**
> Marked as "Acknowledged" by the client.
> The client provided the following explanation:
>
> ```
> Complete checks have been implemented for all locations where special values need to be passed in.
> ```

File(s) affected: `error.move`

Description: The `dipcoin::error` module assigns unique error codes to each error by combining a base error code with a variable component derived from specific error conditions. However, these variable components are all unbounded `u64` values, provided as a function parameter. This creates a risk of error code collisions when an unexpected or invalid number is passed to an error code function.

For example, `error::order_is_canceled()` assumes the `is_taker` parameter is either 0 or 1, which is true under normal conditions. But if a mistake leads to passing a value like 2, the resulting error code may overlap with an entirely different error such as `error::order_has_invalid_signature()`. This ambiguity makes debugging significantly harder and reduces the maintainers' ability to accurately diagnose issues during the system's operation.

Recommendation: Consider using enums and booleans instead of raw `u64` parameters. Even though the system defines many different error codes, the variable components come from a small set of conceptual categories. Introducing enums (e.g., an `Offset` enum) and converting

flags like `isTaker` or `isMaker` into booleans would eliminate this class of collision. Alternatively, strict boundary checks could be added to validate error parameters before constructing the final error code.

DIP-24 Redundant Additional Check for Insurance Pool

Informational · Fixed

Description: The `isolated_trading::apply_isolated_margin()` function makes sure that the insurance fund cannot create new positions, increase existing positions, or flip the position direction. Nevertheless, `exchange::trade()` employs addtional checks that are ineffective. These checks would allow the insurance fund to increase their position, which is unintended:

```
if (insurance_active_pool == maker_address || insurance_active_pool == taker_address) {
    assert!(perp.positions().borrow(insurance_active_pool).size() > 0,
            error::insurance_active_account_can_not_open_new_position());
};
```

Recommendation: While the insurance fund cannot effectively increase their position due to the checks in `apply_isolated_margin()`, we recommend removing the redundant code in `exchange::trade()`.

Auditor Suggestions

S1 Validate Input Parameters

Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `b0119017473969b7d4afe68c350a0113a9f81113`.

File(s) affected: `exchange.move`, `perpetual.move`, `evaluator.move`, `position.move`

Description: Several functions lack input validation for critical parameters, which could lead to unexpected behavior, execution failures, or invalid state configurations.

- `exchange::apply_funding_rate()`: The `flag` parameter is not validated to be within the expected range (0-3). Invalid values default to the ADL branch. It should be validated with `assert!(flag <= 3)`.
- `perpetual::set_max_oi_open()` and `evaluator::set_max_oi_open()`: The length of the `max_limit` vector is not checked against the maximum leverage allowed by the IMR. If the vector is too short, high-leverage positions might bypass open-interest checks. It should be validated that `max_limit` covers all possible leverage tiers defined by the IMR.
- `position::remove_empty_positions()`: The `pos_keys` vector length is not capped, potentially causing the transaction to hit the dynamic field access limit (1000) and fail. It should be validated with `assert!(pos_keys.length() <= 1000)`.
- `perpetual::set_maker_fee()` and `perpetual::set_taker_fee()`: Fees are not validated to be within 0-100% (0 to 1e9), allowing for invalid fee configurations. They should be validated with `assert!(fee <= 1_000_000_000)`.

Recommendation: Add input validation checks to the functions above.

S2 Combine Oracle Price and Exponent Retrieval

Fixed

> **Update**
> Marked as "Fixed" by the client.
> Addressed in: `d6a291b3762f8914e82fe0012f04f2df7b35e15a`, `5f8d30618ad7d4463c5bff9668a641dd23ece674`.

File(s) affected: `library.move`, `perpetual.move`

Description: The `library` module splits price retrieval into `get_oracle_price()` and `get_oracle_base()`. The code comments claim this allows caching the exponent to optimize gas. However, in `perpetual::update_oracle_price()`, both functions are called sequentially for every update, meaning no caching is actually performed. Furthermore, both functions independently call `pyth::get_price_no_older_than()`, causing the protocol to pay for the staleness check and data retrieval twice for every price update.

Recommendation: Combine these into a single function that returns both the price and the exponent from a single Pyth call to reduce gas costs.

S3 Shared Event Definition Causes Aborts and Unnecessary State Creation

Mitigated

> **Update**

Marked as "Fixed" by the client.
Addressed in: `f87bfa9a7ee371bc2534db3dcc55920b7dceaa16`.

File(s) affected: `bank.move`

Description: The `BankBalanceUpdateEvent` is used for deposits, withdrawals, and internal transfers. It requires emitting both `src_balance` and `dest_balance` regardless of the operation type.

- Fixed In `withdraw()` and `withdraw_all()`, the code attempts to read `dest_balance` from the accounts table. If the destination address (e.g., a user's wallet) does not have a registered bank account, this lookup fails and the transaction aborts. This prevents users from withdrawing to addresses that have not interacted with the bank before.
- Unresolved In `deposit_internal()`, the code forces the creation of a bank account for the `sender` (via `create_account_if_not_exist`) even if they are just depositing coins and don't need an internal balance state. This is likely done just to satisfy the event's requirement to emit `src_balance`.

Recommendation: Split the `BankBalanceUpdateEvent` into separate events (`DepositEvent`, `WithdrawEvent`, `TransferEvent`) that only include fields relevant to the specific action. This removes the need to look up or create accounts for addresses that are not participating in the internal accounting of that specific operation.

S4
Simplify Role Management by Removing Redundant Capabilities or Centralizing Access Control

Acknowledged

Update
Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
The main point is that permission-related function calls should take the corresponding Cap as the first parameter — this is the standard practice in Sui. That said, whether you make this change or not doesn't have a big impact.
```

File(s) affected: `roles.move`, `protocol.move`

Description: The current system uses a hybrid approach where roles (Exchange Manager, Deleveraging Operator, Funding Rate Operator) are managed by minting new Capability objects and storing their IDs in `ProtocolConfig`. This adds unnecessary complexity. Since the `ExchangeAdmin` controls these assignments, a simpler Access Control List (ACL) storing authorized addresses in `ProtocolConfig` would achieve the same security with less overhead.

Recommendation: Refactor role management to either:

1. Use a pure ACL pattern where `ProtocolConfig` stores the addresses of authorized operators, removing the need for `ExchangeManagerCap`, `DeleveragingCap`, and `FundingRateCap` objects. Or
2. Use standard transferable Capabilities where the current holder can rotate the key by transferring the object, removing the need for the admin to mint new ones and update the config.

S5 Use BCS Encoding Instead of JSON for Order Serialization

Acknowledged

Update
Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
It is designed as JSON primarily to cooperate with the frontend, so that when the user's wallet pops up, order details can be displayed through the JSON string.
```

File(s) affected: `order.move`

Description: The current implementation manually constructs a JSON string for order serialization in `order::get_serialized_order()`. This is gas-inefficient and prone to formatting errors (e.g., whitespace or field ordering mismatches). Sui Move provides native support for Binary Canonical Serialization (BCS), which is faster, cheaper, and deterministic.

Recommendation: Replace the manual JSON string construction with BCS encoding. This will significantly reduce the gas cost of order verification and simplify the codebase.

S6 Redundant `Order` Flags

Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
Because considering that deleting extra fields during SUI contract upgrades is not allowed, this impact is minimal.
```

File(s) affected: `order.move`

Description: The `Order` struct has a redundant encoding of all flags, thereby redundantly including the data twice.

Recommendation: We recommend removing flag encoding if it remains unused.

S7 Rename `library::get_public_address()` Local Variable

Fixed

Update

Marked as "Fixed" by the client.
Addressed in: `13222acb4cdc8dcd5def9c1545c8a87a59c04e4c` .

File(s) affected: `library.move`

Description: The `library::get_public_address()` function declares a local variable named `address` of type `address` . Since the variable name exactly matches the `address` type keyword, this can create ambiguity and may be disallowed in future compiler versions.

Recommendation: Consider renaming the local variable to avoid conflicts and improve clarity.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not pose an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Test Suite Results

Test data output was obtained with `sui move test --skip-fetch-latest-git-deps` . The Sui test suite contains 82 tests which execute successfully.

Fix Review Update

The Sui test suite was expanded from 82 tests to 275 tests during the audit. Therefor we increased the test quality score from low to medium.

```
sui move test -p contracts --skip-fetch-latest-git-deps
INCLUDING DEPENDENCY Pyth
INCLUDING DEPENDENCY Wormhole
INCLUDING DEPENDENCY Sui
INCLUDING DEPENDENCY MoveStdlib
BUILDING dipcoin
Running Move unit tests
[ PASS    ] dipcoin::library_tests::test_add_intent_bytes_and_hash_changes_with_input
```

```
[ PASS    ] dipcoin::funding_rate_tests::test_compute_global_index_multiple_hours
[ PASS    ] dipcoin::insurance_fund_tests::test_create_insurance_fund_active_account_zero
[ PASS    ] dipcoin::adl_tests::test_adl_all_or_nothing_failure_maker
[ PASS    ] dipcoin::bank_tests::test_account_existence
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_all_or_nothing_failure
[ PASS    ] dipcoin::isolated_trading_tests::test_flip_position
[ PASS    ] dipcoin::exchange_tests::test_add_margin_balance_conservation
[ PASS    ] dipcoin::library_tests::test_base_div
[ PASS    ] dipcoin::funding_rate_tests::test_compute_global_index_no_time_change
[ PASS    ] dipcoin::library_tests::test_base_div_by_zero
[ PASS    ] dipcoin::funding_rate_tests::test_compute_global_index_one_hour
[ PASS    ] dipcoin::insurance_fund_tests::test_create_insurance_fund_same_accounts
[ PASS    ] dipcoin::adl_tests::test_adl_all_or_nothing_failure_taker
[ PASS    ] dipcoin::bank_tests::test_create_bank
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_liquidator_has_long_position_add_position
[ PASS    ] dipcoin::isolated_trading_tests::test_mtb_violation_should_fail
[ PASS    ] dipcoin::exchange_tests::test_add_margin_increases_position_margin
[ PASS    ] dipcoin::library_tests::test_base_mul
[ PASS    ] dipcoin::funding_rate_tests::test_compute_global_index_partial_hour
[ PASS    ] dipcoin::library_tests::test_ceil
[ PASS    ] dipcoin::insurance_fund_tests::test_create_insurance_fund_security_account_zero
[ PASS    ] dipcoin::funding_rate_tests::test_compute_global_index_price_impact
[ PASS    ] dipcoin::adl_tests::test_adl_all_or_nothing_success
[ PASS    ] dipcoin::bank_tests::test_deposit_and_balance
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_liquidator_has_long_position_flip_position
[ PASS    ] dipcoin::exchange_tests::test_add_margin_no_position_should_fail
[ PASS    ] dipcoin::isolated_trading_tests::test_open_and_increase_same_side
[ PASS    ] dipcoin::library_tests::test_ceil_with_zero_m
[ PASS    ] dipcoin::funding_rate_tests::test_funding_rate_initialization
[ PASS    ] dipcoin::library_tests::test_compute_mro
[ PASS    ] dipcoin::insurance_fund_tests::test_create_insurance_fund_success
[ PASS    ] dipcoin::adl_tests::test_adl_insurance_fund_insufficient_for_bankruptcy
[ PASS    ] dipcoin::bank_tests::test_deposit_insufficient_coin
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_liquidator_has_long_position_flip_position_loss
[ PASS    ] dipcoin::exchange_tests::test_add_margin_when_delisted_should_fail
[ PASS    ] dipcoin::isolated_trading_tests::test_post_only_ioc_restrictions_fail
[ PASS    ] dipcoin::funding_rate_tests::test_funding_rate_lifecycle
[ PASS    ] dipcoin::library_tests::test_compute_mro_zero_leverage
[ PASS    ] dipcoin::funding_rate_tests::test_large_time_gaps
[ PASS    ] dipcoin::insurance_fund_tests::test_create_insurance_fund_zero_interval
[ PASS    ] dipcoin::library_tests::test_convert_usdc_to_base_decimals
[ PASS    ] dipcoin::adl_tests::test_adl_insurance_fund_zero_balance
[ PASS    ] dipcoin::bank_tests::test_deposit_to_zero_address
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_liquidator_has_long_position_reduce_position
[ PASS    ] dipcoin::exchange_tests::test_add_margin_wrong_sender_should_fail
[ PASS    ] dipcoin::isolated_trading_tests::test_reduce_only_partial_close
[ PASS    ] dipcoin::funding_rate_tests::test_maximum_funding_rate
[ PASS    ] dipcoin::library_tests::test_get_hash
[ PASS    ] dipcoin::insurance_fund_tests::test_create_insurance_fund_zero_limit
[ PASS    ] dipcoin::adl_tests::test_adl_maker_not_underwater
[ PASS    ] dipcoin::funding_rate_tests::test_new_index_creation
[ PASS    ] dipcoin::library_tests::test_get_public_address_format
[ PASS    ] dipcoin::bank_tests::test_deposit_when_bank_paused
[ PASS    ]
dipcoin::liquidation_tests::test_liquidation_liquidator_has_long_position_reduce_position_loss
[ PASS    ] dipcoin::exchange_tests::test_add_margin_zero_amount_should_fail
[ PASS    ] dipcoin::isolated_trading_tests::test_self_trade_fees_only
[ PASS    ] dipcoin::library_tests::test_min
[ PASS    ] dipcoin::funding_rate_tests::test_precision_limits
[ PASS    ] dipcoin::evaluator_tests::test_mtb_long_violation_should_fail
[ PASS    ] dipcoin::insurance_fund_tests::test_multiple_parameter_updates
[ PASS    ] dipcoin::adl_tests::test_adl_multiple_rounds_insurance_fund
[ PASS    ] dipcoin::bank_tests::test_deposit_with_change
[ PASS    ] dipcoin::library_tests::test_round
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_liquidator_has_short_position_add_position
[ PASS    ] dipcoin::exchange_tests::test_add_margin_zero_size_position_should_fail
[ PASS    ] dipcoin::funding_rate_tests::test_zero_funding_rate
[ PASS    ] dipcoin::evaluator_tests::test_price_below_min_should_fail
[ PASS    ] dipcoin::library_tests::test_round_down
[ PASS    ] dipcoin::insurance_fund_tests::test_set_active_account_same_as_security
[ PASS    ] dipcoin::evaluator_tests::test_qty_below_min_should_fail
[ PASS    ] dipcoin::bank_tests::test_deposit_zero_amount
```

```
[ PASS    ] dipcoin::adl_tests::test_adl_normal_case
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_liquidator_has_short_position_flip_position
[ PASS    ] dipcoin::protocol_tests::test_cap_getters_and_settlement_ids_getter
[ PASS    ] dipcoin::library_tests::test_sub
[ PASS    ] dipcoin::exchange_tests::test_adjust_leverage_balance_conservation
[ PASS    ] dipcoin::evaluator_tests::test_step_size_violation_should_fail
[ PASS    ] dipcoin::library_tests::test_to_1x9_vec
[ PASS    ] dipcoin::insurance_fund_tests::test_set_active_account_success
[ PASS    ] dipcoin::evaluator_tests::test_tick_size_violation_should_fail
[ PASS    ] dipcoin::bank_tests::test_get_balance_nonexistent_account
[ PASS    ] dipcoin::adl_tests::test_adl_same_side_positions
[ PASS    ] dipcoin::protocol_tests::test_check_basic_should_fail_when_paused
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_liquidator_has_short_position_flip_position_loss
[ PASS    ] dipcoin::exchange_tests::test_adjust_leverage_margin_excess_branch
[ PASS    ] dipcoin::library_tests::test_verify_signature_branches_return_status_and_prefix
[ PASS    ] dipcoin::evaluator_tests::test_verify_functions_on_trade_checks
[ PASS    ] dipcoin::insurance_fund_tests::test_set_active_account_zero_address
[ PASS    ] dipcoin::s128_tests::test_add
[ PASS    ] dipcoin::bank_tests::test_large_amount_operations
[ PASS    ] dipcoin::adl_tests::test_adl_taker_underwater
[ PASS    ] dipcoin::protocol_tests::test_check_is_normal_should_fail_when_paused
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_liquidator_has_short_position_reduce_position
[ PASS    ] dipcoin::exchange_tests::test_adjust_leverage_margin_shortfall_branch
[ PASS    ] dipcoin::s128_tests::test_add_u128
[ PASS    ] dipcoin::roles_tests::test_add_privileged_addresses_empty_should_fail
[ PASS    ] dipcoin::insurance_fund_tests::test_set_amount_limit_success
[ PASS    ] dipcoin::s128_tests::test_comparison_between_s128
[ PASS    ] dipcoin::bank_tests::test_multi_user_operations
[ PASS    ] dipcoin::adl_tests::test_adl_with_insurance_fund_as_maker
[ PASS    ] dipcoin::protocol_tests::test_check_version_ok_and_privileged_getter_contains
[ PASS    ]
dipcoin::liquidation_tests::test_liquidation_liquidator_has_short_position_reduce_position_loss
[ PASS    ] dipcoin::exchange_tests::test_adjust_leverage_no_position_should_fail
[ PASS    ] dipcoin::roles_tests::test_check_deleveraging_operator_validity_success
[ PASS    ] dipcoin::s128_tests::test_comparison_with_u128
[ PASS    ] dipcoin::insurance_fund_tests::test_set_amount_limit_zero
[ PASS    ] dipcoin::bank_tests::test_multiple_deposits
[ PASS    ] dipcoin::adl_tests::test_adl_with_insurance_fund_as_maker_with_enough_balance
[ PASS    ] dipcoin::protocol_tests::test_pause_and_resume
[ PASS    ] dipcoin::s128_tests::test_div_u128
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_partial_liquidation_long_position_at_mmr
[ PASS    ] dipcoin::exchange_tests::test_adjust_leverage_updates_mro
[ PASS    ] dipcoin::roles_tests::test_check_funding_rate_operator_validity_success
[ PASS    ] dipcoin::insurance_fund_tests::test_set_security_account_same_as_active
[ PASS    ] dipcoin::s128_tests::test_edge_cases
[ PASS    ] dipcoin::bank_tests::test_multiple_withdrawals
[ PASS    ] dipcoin::adl_tests::test_adl_with_insurance_fund_as_taker
[ PASS    ] dipcoin::protocol_tests::test_set_funding_rate_cap_id_and_remove_settlement_cap_id
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_partial_liquidation_long_position_below_mmr
[ PASS    ] dipcoin::s128_tests::test_from
[ PASS    ] dipcoin::exchange_tests::test_adjust_leverage_when_delisted_should_fail
[ PASS    ] dipcoin::roles_tests::test_check_insurance_operator_validity_invalid_cap_should_fail
[ PASS    ] dipcoin::insurance_fund_tests::test_set_security_account_success
[ PASS    ] dipcoin::s128_tests::test_from_subtraction
[ PASS    ] dipcoin::bank_tests::test_set_withdrawal_status
[ PASS    ] dipcoin::protocol_tests::test_set_maker_taker_gas_fee
[ PASS    ]
dipcoin::apply_funding_rate_tests::test_apply_funding_rate_for_insurance_active_position_pay_funding_fee
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_partial_liquidation_short_position_at_mmr
[ PASS    ] dipcoin::exchange_tests::test_adjust_leverage_wrong_sender_should_fail
[ PASS    ] dipcoin::roles_tests::test_check_insurance_operator_validity_success
[ PASS    ] dipcoin::s128_tests::test_mul_u128
[ PASS    ] dipcoin::insurance_fund_tests::test_set_security_account_zero_address
[ PASS    ] dipcoin::s128_tests::test_negate
[ PASS    ] dipcoin::protocol_tests::test_settlement_operator_add_remove
[ PASS    ] dipcoin::bank_tests::test_transfer
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_partial_liquidation_short_position_below_mmr
[ PASS    ]
dipcoin::apply_funding_rate_tests::test_apply_funding_rate_for_insurance_active_position_receive_funding_
fee
[ PASS    ] dipcoin::exchange_tests::test_adjust_leverage_zero_should_fail
[ PASS    ] dipcoin::roles_tests::test_check_settlement_operator_validity_success
```

```
[ PASS    ] dipcoin::insurance_fund_tests::test_set_transfer_interval_success
[ PASS    ] dipcoin::s128_tests::test_negative_number
[ PASS    ] dipcoin::perpetual_admin_tests::test_delist_perpetual
[ PASS    ] dipcoin::bank_tests::test_transfer_insufficient_balance
[ PASS    ] dipcoin::s128_tests::test_one
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_permission_insurance_fund_only_1
[ PASS    ] dipcoin::exchange_tests::test_close_position_after_delist
[ PASS    ] dipcoin::roles_tests::test_create_settlement_operator
[ PASS    ] dipcoin::insurance_fund_tests::test_set_transfer_interval_zero
[ PASS    ] dipcoin::apply_funding_rate_tests::test_apply_funding_rate_for_normal_position
[ PASS    ] dipcoin::s128_tests::test_positive_number
[ PASS    ] dipcoin::perpetual_admin_tests::test_delist_twice_should_fail
[ PASS    ] dipcoin::bank_tests::test_withdraw
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_permission_insurance_fund_only_2
[ PASS    ] dipcoin::s128_tests::test_positive_value
[ PASS    ] dipcoin::exchange_tests::test_close_position_no_position_should_fail
[ PASS    ] dipcoin::roles_tests::test_insurance_operator_update_replaces_old_cap
[ PASS    ] dipcoin::insurance_fund_tests::test_transfer_from_active_to_security
[ PASS    ] dipcoin::apply_funding_rate_tests::test_apply_funding_rate_no_position
[ PASS    ] dipcoin::s128_tests::test_sub
[ PASS    ] dipcoin::perpetual_admin_tests::test_insurance_fund_limits_and_intervals
[ PASS    ] dipcoin::bank_tests::test_withdraw_all
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_permission_insurance_fund_only_3
[ PASS    ] dipcoin::exchange_tests::test_close_position_requires_delist_should_fail
[ PASS    ] dipcoin::s128_tests::test_sub_u128
[ PASS    ] dipcoin::roles_tests::test_manager_updates_and_privileged_addresses
[ PASS    ] dipcoin::insurance_fund_tests::test_transfer_from_active_to_security_exceeds_original_limit
[ PASS    ]
dipcoin::apply_funding_rate_tests::test_apply_funding_rate_normal_position_insufficient_margin
[ PASS    ] dipcoin::perpetual_admin_tests::test_remove_empty_positions_noop
[ PASS    ] dipcoin::s128_tests::test_zero
[ PASS    ] dipcoin::bank_tests::test_withdraw_all_disabled
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_self_liquidation_forbidden
[ PASS    ] dipcoin::roles_tests::test_pause_and_resume_success
[ PASS    ] dipcoin::exchange_tests::test_close_position_zero_size_should_fail
[ PASS    ] dipcoin::insurance_fund_tests::test_transfer_from_active_to_security_insufficient_balance
[ PASS    ] dipcoin::apply_funding_rate_tests::test_apply_funding_rate_same_index
[ PASS    ] dipcoin::perpetual_admin_tests::test_set_all_risk_params
[ PASS    ] dipcoin::settlement_tests::test_get_margin_left_short_position
[ PASS    ] dipcoin::bank_tests::test_withdraw_all_to_zero_address
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_trigger_long_position_at_bankruptcy_price
[ PASS    ] dipcoin::roles_tests::test_pause_twice_should_fail
[ PASS    ] dipcoin::insurance_fund_tests::test_transfer_from_active_to_security_zero_amount
[ PASS    ] dipcoin::exchange_tests::test_create_perpetual_creates_accounts
[ PASS    ] dipcoin::apply_funding_rate_tests::test_apply_funding_rate_sequential_applications
[ PASS    ] dipcoin::perpetual_admin_tests::test_set_fee_and_gas_pool_addresses
[ PASS    ] dipcoin::settlement_tests::test_get_max_removeable_margin_short_position
[ PASS    ] dipcoin::bank_tests::test_withdraw_all_with_remainder
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_trigger_long_position_at_mmr
[ PASS    ] dipcoin::roles_tests::test_remove_privileged_addresses_empty_should_fail
[ PASS    ] dipcoin::insurance_fund_tests::test_transfer_from_security_to_active_after_interval
[ PASS    ] dipcoin::exchange_tests::test_create_perpetual_zero_fee_pool_should_fail
[ PASS    ] dipcoin::perpetual_admin_tests::test_set_funding_rate_success
[ PASS    ] dipcoin::settlement_tests::test_get_target_margin_short_position
[ PASS    ]
dipcoin::apply_funding_rate_tests::test_apply_funding_rate_sequential_applications_comprehensive
[ PASS    ] dipcoin::bank_tests::test_withdraw_all_with_zero_balance
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_trigger_long_position_below_bankruptcy_price
[ PASS    ] dipcoin::roles_tests::test_remove_settlement_operator_success
[ PASS    ] dipcoin::insurance_fund_tests::test_transfer_from_security_to_active_at_limit
[ PASS    ] dipcoin::exchange_tests::test_create_perpetual_zero_gas_pool_should_fail
[ PASS    ] dipcoin::perpetual_admin_tests::test_set_insurance_pool_addresses
[ PASS    ] dipcoin::settlement_tests::test_trade_margin_settlement_maker_account_not_exist_should_fail
[ PASS    ] dipcoin::apply_funding_rate_tests::test_insurance_active_position_pay_funding_fee_adl_maker
[ PASS    ] dipcoin::bank_tests::test_withdraw_disabled
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_trigger_long_position_below_mmr
[ PASS    ] dipcoin::roles_tests::test_resume_without_pause_should_fail
[ PASS    ] dipcoin::insurance_fund_tests::test_transfer_from_security_to_active_exceeds_limit
[ PASS    ] dipcoin::exchange_tests::test_deleverage_executes
[ PASS    ] dipcoin::perpetual_admin_tests::test_set_max_allowed_funding_rate
[ PASS    ] dipcoin::settlement_tests::test_trade_margin_settlement_taker_account_not_exist_should_fail
[ PASS    ] dipcoin::apply_funding_rate_tests::test_insurance_active_position_pay_funding_fee_adl_taker
```

```
[ PASS    ] dipcoin::bank_tests::test_withdraw_insufficient_balance
[ PASS    ] dipcoin::roles_tests::test_set_exchange_admin_success
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_trigger_short_position_above_bankruptcy_price
[ PASS    ] dipcoin::insurance_fund_tests::test_transfer_from_security_to_active_insufficient_balance
[ PASS    ] dipcoin::exchange_tests::test_deleverage_when_delisted_should_fail
[ PASS    ] dipcoin::sub_accounts_tests::test_set_sub_account
[ PASS    ] dipcoin::perpetual_admin_tests::test_set_special_fee_and_get_fee_priority
[ PASS    ]
dipcoin::apply_funding_rate_tests::test_insurance_active_position_pay_funding_fee_liquidation_maker
[ PASS    ] dipcoin::bank_tests::test_withdraw_to_zero_address
[ PASS    ] dipcoin::sub_accounts_tests::test_set_sub_account_zero_address
[ PASS    ] dipcoin::roles_tests::test_set_exchange_admin_zero_address_should_fail
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_trigger_short_position_at_bankruptcy_price
[ PASS    ] dipcoin::insurance_fund_tests::test_transfer_from_security_to_active_interval_not_met
[ PASS    ] dipcoin::exchange_tests::test_deleverage_when_trading_not_permitted_should_fail
[ PASS    ] dipcoin::sub_accounts_tests::test_validate_unique_tx
[ PASS    ] dipcoin::perpetual_admin_tests::test_set_special_fee_zero_address_should_fail
[ PASS    ]
dipcoin::apply_funding_rate_tests::test_insurance_active_position_pay_funding_fee_liquidation_taker
[ PASS    ] dipcoin::bank_tests::test_withdraw_without_account
[ PASS    ] dipcoin::roles_tests::test_set_exchange_manager_success
[ PASS    ] dipcoin::sub_accounts_tests::test_validate_unique_tx_replay
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_trigger_short_position_at_mmr
[ PASS    ] dipcoin::insurance_fund_tests::test_transfer_from_security_to_active_zero_amount
[ PASS    ] dipcoin::exchange_tests::test_liquidate_executes
[ PASS    ]
dipcoin::apply_funding_rate_tests::test_insurance_active_position_pay_funding_fee_trading_maker
[ PASS    ] dipcoin::perpetual_admin_tests::test_special_fee_priority_and_zero_address_failures
[ PASS    ] dipcoin::position_tests::test_compute_pnl_per_unit_and_bankruptcy_price
[ PASS    ] dipcoin::bank_tests::test_withdraw_zero_amount
[ PASS    ] dipcoin::roles_tests::test_set_funding_rate_operator_success
[ PASS    ] dipcoin::liquidation_tests::test_liquidation_trigger_short_position_below_mmr
[ PASS    ] dipcoin::order_tests::test_cancel_order_success_and_verify_status
[ PASS    ] dipcoin::position_tests::test_create_position_and_remove_empty_positions_cleanup
[ PASS    ] dipcoin::exchange_tests::test_liquidate_when_delisted_should_fail
[ PASS    ]
dipcoin::apply_funding_rate_tests::test_insurance_active_position_pay_funding_fee_trading_taker
[ PASS    ] dipcoin::perpetual_admin_tests::test_trading_permit_toggle_and_default_fees
[ PASS    ] dipcoin::position_tests::test_getters_setters_and_average_price_long_short
[ PASS    ] dipcoin::roles_tests::test_set_insurance_operator_success
[ PASS    ] dipcoin::liquidation_tests::test_multiple_liquidations
[ PASS    ] dipcoin::order_tests::test_create_order_idempotent_and_verify_state
[ PASS    ] dipcoin::exchange_tests::test_liquidate_when_trading_not_permitted_should_fail
[ PASS    ] dipcoin::position_tests::test_is_undercollat_and_verify_collat_checks_paths
[ PASS    ] dipcoin::roles_tests::test_set_insurance_operator_zero_address_should_fail
[ PASS    ] dipcoin::position_tests::test_verify_collat_checks_fail_direction_or_size_rule
[ PASS    ] dipcoin::order_tests::test_expiry_and_leverage_checks_success
[ PASS    ] dipcoin::exchange_tests::test_liquidate_with_taker_margin_transfer
[ PASS    ] dipcoin::order_tests::test_expiry_should_fail_when_too_old
[ PASS    ] dipcoin::exchange_tests::test_remove_margin_balance_conservation
[ PASS    ] dipcoin::order_tests::test_flag_and_pack_and_getters_and_to_1x9
[ PASS    ] dipcoin::exchange_tests::test_remove_margin_decreases_position_margin
[ PASS    ] dipcoin::order_tests::test_get_serialized_order_deterministic
[ PASS    ] dipcoin::exchange_tests::test_remove_margin_no_position_should_fail
[ PASS    ] dipcoin::order_tests::test_get_serialized_order_orderbook_only_true_false
[ PASS    ] dipcoin::exchange_tests::test_remove_margin_when_delisted_should_fail
[ PASS    ] dipcoin::order_tests::test_leverage_mro_mismatch_should_fail
[ PASS    ] dipcoin::exchange_tests::test_remove_margin_wrong_sender_should_fail
[ PASS    ] dipcoin::order_tests::test_leverage_zero_should_fail
[ PASS    ] dipcoin::exchange_tests::test_remove_margin_zero_amount_should_fail
[ PASS    ] dipcoin::order_tests::test_verify_and_fill_order_overfill_should_fail
[ PASS    ] dipcoin::exchange_tests::test_remove_margin_zero_size_position_should_fail
[ PASS    ] dipcoin::order_tests::test_verify_and_fill_order_qty_success_and_overfill
[ PASS    ] dipcoin::exchange_tests::test_sum_transaction_balance_variants
[ PASS    ] dipcoin::order_tests::test_verify_and_fill_reduce_only_invalid_should_fail
[ PASS    ] dipcoin::exchange_tests::test_trade_executes_positions_update
[ PASS    ] dipcoin::order_tests::test_verify_and_fill_reduce_only_rules
[ PASS    ] dipcoin::exchange_tests::test_trade_when_protocol_paused_should_fail
[ PASS    ] dipcoin::exchange_tests::test_trade_with_gas_fee_transfer
Test result: OK. Total tests: 275; passed: 275; failed: 0
```

Code Coverage

The repository includes two separate test suites: a Sui Move unit test suite and a TypeScript-based integration suite. Coverage metrics can only be generated for the Move tests and therefore do **not** reflect repository-wide coverage. Several core modules, including `isolated_trading` and `isolated_liquidation`, currently have minimal test coverage. We strongly recommend extending the Move test suite to cover these critical paths to improve security guarantees and catch bugs in future releases.

Fix Review Update

Test coverage improved substantially, with total Move coverage increasing from **30.91% to 89.11%**. Previously untested or minimally covered core modules such as `isolated_trading`, `isolated_liquidation`, `perpetual`, and `order` were expanded to ~90–96% coverage. This significantly strengthens confidence in critical trading, liquidation, and settlement paths and reduces the risk of regressions.

Module	Coverage %
error	45.16
protocol	84.97
library	89.08
roles	88.37
sub_accounts	87.26
bank	95.76
coin	0.00
evaluator	79.96
s128	97.97
funding_rate	91.25
position	88.94
settlement	93.27
insurance_fund	100.00
perpetual	88.53
order	89.67
isolated_trading	96.12
isolated_liquidation	95.25
isolated_adl	95.35
exchange	92.92
initialize	100.00
test	0.00
Total Move Coverage	89.11

Changelog

- 2025-12-01 - Initial report
- 2025-12-16 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT